DAFTAR PUSTAKA

Deo, Valiandro. (2020). “Pengamanan Jawaban Ujian *Computer Based Test* (CBT) dengan menggunakan algoritma RSA dan fungsi HMAC berbasis algoritma SHA-1”. Skripsi. Makassar: Universitas Hasanuddin.

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

Nigel P. Smart. (2016). Cryptography Made Simple. doi: https://doi.org/10.1007/978-3-319-21936-3

O. Ali, A. Jaradat, A. Kulakli and A. Abuhalimeh, "A Comparative Study: Blokchain Technology Utilization Benefits, Challenges and Functionalities," in IEEE Access, vol. 9, pp. 12730-12749, 2021, doi: 10.1109/ACCESS.2021.3050241.

Pankaj Dutta, Tsan-Ming Choi, Surabhi Somani, Richa Butala, Blokchain technology in supply chain operations: Applications, challenges and research opportunities, Transportation Research Part E: Logistics and Transportation Review, Volume 142, 2020, 102067, ISSN 1366-5545, https://doi.org/10.1016/j.tre.2020.102067.

R. Johari, V. Kumar, K. Gupta et al., BLOSOM: Blokchain technology for Security Of Medical records, ICT Express (2021), https://doi.org/10.1016/j.icte.2021.06.002.

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

Shuyun Shi, Debiao He, Li Li, Neeraj Kumar, Muhammad Khurram Khan, Kim-Kwang Raymond Choo, Applications of blokchain in ensuring the security and privacy of electronic health record systems: A survey, Computers & Security, Volume 97, 2020, 101966, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2020.101966.

Zaghloul, E., Li, T., Mutka, M. W., & Ren, J. (2020). Bitcoin and Blokchain: Security and Privacy. IEEE Internet of Things Journal, 1–1. doi:10.1109/jiot.2020.3004273

LAMPIRAN

Lampiran 1. Node.java

```
public class Node {
    private Node left;
    private Node right;
    private String hash;

    public Node(Node left, Node right, String hash) {
        this.left = left;
        this.right = right;
        this.hash = hash;
    }

    public Node getLeft() {
        return left;
    }

    public Node getRight() {
        return right;
    }

    public void setRight(Node right) {
        this.right = right;
    }

    public void setLeft(Node left) {
        this.left = left;
    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

}
```

Lampiran 2. MerkleTree.java

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

public class MerkleTree {

    private Hash hash= new Hash();
    private int i = 1;

    public Node generateTree(ArrayList<String> data) throws
Exception{
        ArrayList<Node> childNodes = new ArrayList<>();

        for (String message : data) {
            childNodes.add(new Node(null, null,
hash.generateHash(message)));
        }

        return buildTree(childNodes);
    }

    private Node buildTree(ArrayList<Node> child) throws Exception {
        ArrayList<Node>parents = new ArrayList<>();

        if (child.size() == 1) {
            return child.get(0);
        }

        while (child.size() != 1) {
            int index = 0, length = child.size();
            while (index < length) {

                Node leftChild = child.get(index);
                Node rightChild = null;

                if ((index + 1) < length) {
                    rightChild = child.get(index + 1);
                } else {
                    rightChild = new Node(null, null,
leftChild.getHash());
                }

                String parentHash =
hash.generateHash(leftChild.getHash() + rightChild.getHash());
```

```
                parents.add(new Node(leftChild, rightChild,
parentHash));
                index += 2;
            }
            child = parents;
            parents = new ArrayList<>();
        }
        return child.get(0);
    }

    public void printMerkleTree(Node tree) {

        if (tree == null) {
            return;
        }

        if (tree.getLeft() == null && tree.getRight() == null) {
            System.out.println(tree.getHash());
            return;
        }

        Queue<Node> queue = new LinkedList<>();
        queue.add(tree);
        queue.add(null);

        while (!queue.isEmpty()) {

            Node node = queue.poll();
            if (node != null) {
                System.out.println("Hash-" + i + " = " +
node.getHash());
                i++;
            } else {
                if (queue.isEmpty()) {
                    queue.add(null);
                }
            }
            if (node != null && node.getLeft() != null) {
                queue.add(node.getLeft());
                System.out.println();}

            if (node != null && node.getRight() != null) {
                queue.add(node.getRight());}
        }
    }
}
```

Lampiran 3. Hash.java

```
import java.security.MessageDigest;

public class Hash {

    MessageDigest md;

    public String generateHash(String message) throws Exception {
        String hash = "";
        md = MessageDigest.getInstance("SHA-256");
        byte[] bytes = md.digest(message.getBytes("UTF-8"));

        StringBuffer stringBuffer = new StringBuffer();
        for (int i = 0; i < bytes.length; i++) {
            String hex = Integer.toHexString(0xff & bytes[i]);
            if (hex.length() == 1) {
                stringBuffer.append('0');
            }
            stringBuffer.append(hex);
        }
        hash += stringBuffer;
        return hash;
    }
}
```

Lampiran 4. RSA.java

```
import java.util.Scanner;
public class RSA {
    static Scanner in = new Scanner(System.in);
    // Check Relative Prime
    public long gcd(long a, long b) {
        long gcd = 1;
        for (long i = 1; i <= a && i <= b; i++) {
            if (a % i == 0 && b % i ==0) {
                gcd = i;
            }
        }
        return gcd;
    }

    // Check Prime Number
    public boolean isPrime(long number) {
        if (number == 0 || number == 1) {
            return false;
        }
        for (int i = 2; i <= number / 2; ++i) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }

    // KeyGeneration
    public long[] generateKey() {
        long n = 0,e = 0,d = 0;
        long totient = 0;

        System.out.println("Choose different prime numbers : ");
        System.out.print("p : ");
        long p = in.nextLong();
        if (!isPrime(p)) {
            System.out.println("p is not prime");
            System.exit(0);
        }
        System.out.print("q : ");
        long q = in.nextLong();
        if (!isPrime(q)) {
            System.out.println("q is not prime");
            System.exit(0);
```

```
        }
        if (p == q) {
            System.out.println("p and q must different");
            System.exit(0);
        }
        n = p*q;
        totient = (p-1)*(q-1);
        System.out.print("Enter Public Key : ");
        e = in.nextLong();
        if (!isPrime(e)) {
            System.out.println("e is not prime");
            System.exit(0);
        }

        // Check if totient and e are relative prime
        boolean check = gcd(e, totient) == 1 ? true : false;

        if (check) {
            d = generatePrivateKey(e, totient);
        } else {
            System.out.println("e and totient not relative prime");
            System.exit(0);
        }

        long[] key = {n,e,d};
        return key;
    }

    // Private Key Computation
    public long generatePrivateKey(long e, long totient) {
        long d = 0;
        int i = 1;
        while (true) {
            d = (1 + i * totient) / e;
            i++;
            if ((d*e) % totient == 1) {
                return d;
            }
        }

    }

    // Encryption
    public String encrypt(String message, long e, long n) {
        String s = "";
        long length = message.length();
```

```
        for (int i = 0; i < length; i++) {
            String result = "";
            long m = message.charAt(i);
            long cipher = 1;
            for (int j = 0; j < e; j++) {
                cipher *= m;
                cipher = cipher%n;
            }
            result = String.valueOf(cipher);

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

            if (result.length() < nLenght) {
                result = nol.repeat(nLenght - result.length()) +
result;
            }

            s += result;
        }
        return s;
    }

    // Decryption
    public String decrypt(String message, long d, long n) {
        String s = "";
        String result = "";
        int nLength = String.valueOf(n).length();
        for (int i = 0; i < message.length(); i += nLength) {
            long plain = 1;
            String str = message.substring(i, i + nLength);
            long c = Integer.parseInt(str);
                for (int j = 0; j < d; j++) {
                    plain *= c;
                    plain = plain % n;
                }
            result = String.valueOf((char)
Integer.parseInt(String.valueOf(plain)));
            s += result;
        }
        return s;
    }
}
```

Lampiran 5. Blok.java

```
import java.util.ArrayList;

public class Blok {
    private String timeStamp;
    private String hash;
    private String previousHash;
    private ArrayList<String> data = new ArrayList<>();
    private int nonce = 0;
    private Hash generateHash = new Hash();

    public Blok(String timeStamp, ArrayList<String> data, String
previousHash) {
        this.data = data;
        this.timeStamp = timeStamp;
        this.previousHash = previousHash;
        this.hash = calculateHash();
    }

    public Blok(){}

    public String getHash() {
        return hash;
    }

    public String getTimeStamp() {
        return timeStamp;
    }

    public ArrayList<String> getData() {
        return data;
    }

    public int getNonce() {
        return nonce;
    }

    public String getPreviousHash() {
        return previousHash;
    }

    public void mineBlok(int diff) {
        String target = new String(new char[diff]).replace('\0',
'0');
        while (!hash.substring(0,diff).equals(target)) {
```

```
            nonce++;
            hash = calculateHash();
        }
    }

    public String calculateHash() {

        String s = "";
        try {

            MerkleTree mTree = new MerkleTree();
            Node root = mTree.generateTree(data);

            String plaintext =
                timeStamp +
                root.getHash() +
                previousHash +
                Integer.toString(nonce);

            s += generateHash.generateHash(plaintext);

        } catch (Exception e) {
            System.out.println("Error Occured on Hashing");
        }

        return s;
    }
}
```

Lampiran 6. BlokChain.java

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;

public class BlokChain {
    private ArrayList<Blok> blokchain = new ArrayList<>();
    private ArrayList<String> genesisData = new ArrayList<>();
    private int diff = 0;
    String genesisHash;

    public BlokChain(long e, long d, long n, int diff) {
        genesisData.add("");
        genesisHash = new String(new char[diff]).replace('\0', '0');

        this.diff = diff;

        // Create Genesis Blok
        Blok blok = new Blok(
            date(),
            genesisData,
            genesisHash
        );
        blokchain.add(blok);
        blok.mineBlok(diff);
    }

    public void addBlok(ArrayList<String> data) {
        Blok blok = new Blok(
            date(),
            data,
            blokchain.get(blokchain.size() - 1).getHash()
        );
        blokchain.add(blok);
        blok.mineBlok(diff);

    }

    public boolean isBlokChainValid() {
        Blok currentBlok;
        Blok previousBlok;

        for (int i = 1; i < blokchain.size(); i++) {
            currentBlok = blokchain.get(i);
            previousBlok = blokchain.get(i - 1);
```

```
            if
(!previousBlok.getHash().equals(currentBlok.getPreviousHash())){
                System.out.println("BlokChain are Invalid, check the
hash");
                return false;
            }
            if
(!currentBlok.getHash().equals(currentBlok.calculateHash())) {
                System.out.println("Blok- " + " " + i + "hash has
changed, check your data");
                return false;
            }
            // System.out.println("Blok-" + i + " dan Blok-" + (i-1)
+ " = terverifikasi");
        }
        return true;
    }

    public void getListOfBlokChain() {
        for (int i = 0; i < blokchain.size(); i++) {
            System.out.println();
            System.out.println("Blok #" + i);
            System.out.println("Timestamp : " +
blokchain.get(i).getTimeStamp());
            System.out.println();
            System.out.println("Data : ");
                for (int j = 0; j <
blokchain.get(i).getData().size(); j++) {
                    System.out.println("Data-" + (j + 1) + ": " +
blokchain.get(i).getData().get(j));
                }
            System.out.println();
            System.out.println("Hash : " +
blokchain.get(i).getHash());
            System.out.println("Previous Hash : " +
blokchain.get(i).getPreviousHash());
            System.out.println("Nonce : " +
blokchain.get(i).getNonce());
        }
    }

    public ArrayList<Blok> getBlokChain() {
        return blokchain;
    }

    private String date() {
```

```
        DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
        LocalDateTime now = LocalDateTime.now();
        return dtf.format(now);
    }
}
```

Lampiran 7. Main.java

```
import java.util.Scanner;
import java.util.ArrayList;

public class Test {
    static Scanner in = new Scanner(System.in);
    static RSA rsa = new RSA();
    static long publicKey = 1;
    static long privateKey = 1;
    static long n = 1;
    static int dataCount = 5;
    static int diff = 2;
    static BlokChain senderblokChain;
    static BlokChain receiverBlokChain;
    static ArrayList<String> dataBlok = new ArrayList<>();
    static ArrayList<String> answer = new ArrayList<>();

    public static void main(String[] args) throws Exception {
        ArrayList<String> cipherData = new ArrayList<>();
        ArrayList<String> plainData = new ArrayList<>();

        generateKey();
        senderblokChain = new BlokChain(publicKey, privateKey, n,
diff);
        receiverBlokChain = senderblokChain;
        boolean validate = false;

        while (true) {
            System.out.println("\nChoose :\n1. Sender Side\n2.
Receiver Side\n3. Check Validity Blokchain\n4. Exit");
            int n = in.nextInt();
            switch (n) {
                case 1:
                    cipherData = sender();
                    senderblokChain.addBlok(dataBlok);
                    senderblokChain.getListOfBlokChain();
                    System.out.println("Message Sent");
                    printArrayList(cipherData);
                    System.out.println();
                    break;
                case 2:
                    plainData = receiver(cipherData);
                    printArrayList(plainData);
                    System.out.println();
                    if (verificationData(plainData)) {
```

```
                        System.out.println("Data verified");
                    } else {
                        System.out.println("Data not verified");
                    }
                    break;
                case 3:
                    validate = senderblokChain.isBlokChainValid();
                    System.out.println("Blokchain valid? " +
validate);
                    break;
                case 4:
                    System.exit(0);
                default:
                    System.out.println("Wrong input");;
            }
        }

    }

    private static boolean verificationData(ArrayList<String>data) {
        boolean verified = false;
        for (int i = 0; i < senderblokChain.getBlokChain().size();
i++) {
            if
(data.equals(senderblokChain.getBlokChain().get(i).getData())) {
                verified = true;
            }
        }
        return verified ;
    }

    private static void printArrayList(ArrayList<String> list) {
        for (int i = 0; i < list.size(); i++) {
            System.out.println("Data-" + (i+1) + ": " +
list.get(i));
        }
    }

    private static void generateKey() {
        RSA rsa = new RSA();
        long key[] = rsa.generateKey();
        n = key[0];
        publicKey = key[1];
        privateKey = key[2];
```

```
        System.out.println("public key e : (" + publicKey + ", " + n
+ ")");
        System.out.println("private key d : (" + privateKey + ", " +
n + ")");
    }

    private static ArrayList<String> inputData() {
        ArrayList<String> data = new ArrayList<>();
        data.clear();
        String s = "";
        in.nextLine();
        for (int i = 0; i < dataCount; i++) {
            System.out.print("Data-" + (i+1) + " : ");
            s = in.nextLine();
            data.add(s);
        }
        return data;
    }

    private static ArrayList<String> sender() {
        ArrayList<String> list = new ArrayList<>();
        ArrayList<String> cipherList = new ArrayList<>();

        System.out.println("Masukkan data : ");

        list = inputData();
        dataBlok = list;
        long e = publicKey;

        for (int i = 0; i < list.size(); i++) {
            String encryptMessage = "";
            encryptMessage = rsa.encrypt(list.get(i), e, n);
            cipherList.add(encryptMessage);
        }

        return cipherList;
    }

    private static ArrayList<String> receiver(ArrayList<String>
cipherData) {
        ArrayList<String> plainMessage = new ArrayList<>();

        long d = privateKey;

        for (int i = 0; i < cipherData.size(); i++) {
            String decryptMessage = "";
```

```
            decryptMessage = rsa.decrypt(cipherData.get(i), d, n);
            plainMessage.add(decryptMessage);
        }

        return plainMessage;
    }
}
```