Pembangkitan Kombinasi Kode pada Google Authenticator

Aisyah Dzulqaidah 13510005
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510005@std.stei.itb.ac.id

***Abstrak*—Penggunaan nama-pengguna dan sandi-lewat masih dirasa kurang cukup aman untuk otentikasi suatu akun. Sekarang telah dibuat metode baru untuk mengamankan suatu akun yaitu dengan *2-step verification* yang disediakan oleh Google. Dengan *2-step verification*, pengguna tidak hanya memasukkan nama-pengguna dan sandi-lewat untuk masuk ke dalam suatu akun tetapi juga memasukkan kode sebanyak enam digit. Kode ini akan selalu berganti dan dikirimkan ke pengguna melalui pesan singkat maupun suatu aplikasi berbasis *mobile* yang bernama Google Authenticator. Tidak hanya untuk mengamankan akun-akun pada situs yang diusung oleh Google, layanan *2-step verification* juga bisa digunakan pada situs lain seperti Facebook, Dropbox, Wordpress dan situs-situs lainnya. Pada makalah ini akan dibahas mengenai pembangkitan kombinasi enam digit kunci yang menggunakan dua algoritma yaitu HMAC-Based One-time Password (HOTP) dan algoritma Time-based One-time Password (TOTP) yang digunakan pada aplikasi Google Authenticator. Aplikasi Google Authenticator ini diimplementasikan pada tiga platform mobile dengan implementasiyang berbeda, yaitu Android, iOS, dan BlackBerry.**

***Kata Kunci*—2-step verification, Google Authenticator, algoritma HOTP, algoritma TOTP.**

I. PENDAHULUAN

Perilaku pembajakan suatu akun baik akun sosial maupun akun penting lainnya di internet semakin marak. Otentikasi berupa nama-pengguna dan sandi-lewat yang bisa disebut *1-step verification* masih mudah untuk ditembus oleh pembajak yang ingin mencuri data-data penting dalam suatu akun atau membajak suatu akun tersebut. Kombinasi sandi-lewat dan nama pengguna sesulit apapun masih dapat diretas oleh pembajak dengan berbagai metode yang digunakan.

Perusahaan teknologi informasi raksasa di dunia yaitu Google mengenalkan suatu layanan baru untuk pengecekan kebenaran pengguna yaitu *2-step verification*. Layanan ini menyediakan satu kolom tambahan selain kolom nama pengguna dan sandi-lewat yang harus dimasukkan oleh pengguna untuk masuk pada akun Google. Kolom ini harus diisi dengan enam digit angka yang telah dikirimkan ke pengguna. Setelah memasukkan ketiga kolom tersebut, pengguna dapat masuk ke dalam suatu akun. Jika telah berjalan tiga puluh hari (jika *cookies* disediakan) atau setelah *log-out* dari akun tersebut, pengguna harus memasukkan kembali enam digit kunci bersama nama-pengguna dan sandi-lewat untuk dapat mengakses kembali akun tersebut. Penggunaan 2-step verification ini akan meningkatkan tingkat keamanan dari akun yang dimiliki pengguna.

Layanan *2-step verification* ini sebenarnya sudah lama dikenalkan oleh Google, yaitu pada tahun 2011, namun baru marak pada pertengahan 2013. *2-step verification* ini mulai banyak digunakan setelah muncul suatu aplikasi berbasis mobile yaitu Google Authenticator yang diimplementasikan pada ketiga platform yaitu Android, Blackberry, dan iOS, dan pada *pluggable* PAM ().Aplikasi ini akan memberikan enam digit kunci yang diminta pada *2-step verification.* Tanpa harus terhubung ke internet, aplikasi ini dapat tetap membangkitkan kombinasi enam digit kunci tersebut

.

Gambar 1 Logo Google Authenticator

Pada *platform* Android, Google Authenticator mendukung:

- Akun majemuk
- 30 detik kode TOTP
- Kode HOTP
- Penyediaan kunci dengan memindai kode QR
- Masukan string kunci RFC 3548 base32 secara manual

Pada *platform* iOS, Google Authenticator mendukung:

- Akun majemuk
- 30 detik kode TOTP

- Kode HOTP
- Penyediaan kunci dengan memindai kode QR
- Masukan string kunci RFC 3548 base32 secara manual

Pada *platform* BlackBerry, Google Authenticator mendukung:

- Akun majemuk
- 30 detik kode TOTP
- Kode HOTP
- Masukan string kunci RFC 3548 base32 secara manual

Pada *pluggable* PAM, Google Authenticator mendukung:

- *Per-user secret* dan *status file* yang disimpan pada direktori *home*.
- 30 detik kode TOTP
- Kode awal darurat
- Penyediaan kunci dengan memindai kode QR
- Masukan string kunci RFC 3548 base32 secara manual
- Perlindungan terhadap serangan balik

Algoritma yang digunakan untuk membangkitkan enam digit kunci itu ada dua yaitu algoritma HMAC-Based One-time Password (HOTP) dan algoritma Time-based One-time Password (TOTP) yang akan dijelaskan pada bagian dasar Teori

II. DASAR TEORI

A. Kombinatorial

Kombinatorial adalah cabang matematika untuk menghitung jumlah penyusunan objek-objek tanpa harus mengenumerasi semua kemungkinan susunannya[1]. Terdapat dua macam cara menhitung pada kombinatorial, yaitu permutasi dan kombinasi.

A.1 Permutasi

Permutasi adalah jumlah urutan berbeda dari pengaturan objek-objek.Permutasi merupakan bentuk khusus aplikasi kaidah perkalian.

Misalkan jumlah objek adalah *n*, maka

1. urutan pertama dipilih dari *n* objek,
2. urutan kedua dipilih dari $n - 1$ objek,
3. urutan ketiga dipilih dari $n - 2$ objek,
4. …
5. urutan terakhir dipilih dari 1 objek yang tersisa.

Menurut kaidah perkalian, permutasi dari *n* objek adalah

$$n(n-1)(n-2)\ldots(2)(1) = n! \text{ [1]}$$

Pada permutasi urutan hasil pengaturan diperhatikan

B.1 Kombinasi

Permutasi adalah jumlah urutan berbeda dari pengaturan objek-objek.Permutasi merupakan bentuk khusus aplikasi kaidah perkalian.

Misalkan jumlah objek adalah *n*, maka

1. urutan pertama dipilih dari *n* objek,
2. urutan kedua dipilih dari $n - 1$ objek,
3. urutan ketiga dipilih dari $n - 2$ objek,
4. …
5. urutan terakhir dipilih dari 1 objek yang tersisa.

Menurut kaidah perkalian, permutasi dari *n* objek adalah

$$n(n-1)(n-2)\ldots(2)(1) = n! \text{ [1]}$$

Pada Kombinasi, urutan pada hasil tidak diperhatikan.

B. Algoritma Time-Based One-time Password (TOTP)

Algoritma TOTP ini digunakan untuk membangkitkan kunci setiap 30 detik. Dengan algoritma ini kunci akan berubah setiap 30 detik. Berikut pseudocode yang digunakan untuk membangkitkan kunci:

```
function GoogleAuthenticatorCode(string secret)
    key := base32decode(secret)
    message := counter encoded on 8 bytes
    hash := HMAC-SHA1(key, message)
    offset := last nibble of hash
    truncatedHash := hash[offset..offset+3]
    Set the first bit of truncatedHash to zero
code := truncatedHash mod 1000000
    pad code with 0 until length of code is 6
    return code
```

C. Algoritma HMAC-Based One-time Password (HOTP)

Algoritma ini digunakan untuk membangkitkan enam kunci yang dipicu suatu event tertentu. Dengan Algoritma ini tidak harus menunggu 30 detik untuk membangkitkan suatu kunci. Berikut *pseudocode* dari algoritma ini:

Function **GoogleAuthenticatorCode**

(string secret)

key := base32decode(secret)

message := floor(current Unix time / 30)

hash := HMAC-SHA1(key, message)

offset := last nibble of hash

truncatedHash := hash[offset..offset+3]

Set the first bit of truncatedHash to zero

code := truncatedHash mod 1000000

pad code with 0 until length of code is 6

return code

III. PENERAPAN PADA PEMBANGKITAN KUNCI

A. Kemungkinan Kombinasi Kunci

Kemungkinan kunci yang dapat dibangkitkan pada setiap pembangkitan dapat dihitung dengan teori kombinatorial.

Kunci yang digunakan pada Google Authenticator terdiri dari enam digit angka. Domain dari kunci tersebut yaitu bilangan asli dari 0 hingga 9 yang berarti 10 kemungkinan angka.Angka boleh berulang namun dapat memengaruhi kemungkinan.

Banyak kombinasi angka yang dapat dibangkitkan pada suatu waktu yaitu

10	10	10	10	10	10

10x10x10x10x10x10= 1.000.000

Jadi, kemungkinan banyak digit yang bisa dibangkitkan yaitu 1.000.0000. Dari kemungkinan ini pembajak harus menebak satu kunci dari 1.000.000 kunci selama 30 detik. Hal ini sangat sulit untuk dilakukan sehingga keamanan meningkat.

B. Implementasi Algoritma TOTP

Berikut *reference algorithm* dari RFC6238[6]:

```
/**
 Copyright (c) 2011 IETF Trust and the persons identified as
 authors of the code. All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, is permitted pursuant to, and subject to the license
 terms contained in, the Simplified BSD License set forth in Section
 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
 (http://trustee.ietf.org/license-info).
 */

 import java.lang.reflect.UndeclaredThrowableException;
 import java.security.GeneralSecurityException;
 import java.text.DateFormat;
 import java.text.SimpleDateFormat;
 import java.util.Date;
 import javax.crypto.Mac;
 import javax.crypto.spec.SecretKeySpec;
 import java.math.BigInteger;
 import java.util.TimeZone;


 /**
  * This is an example implementation of the OATH
  * TOTP algorithm.
  * Visit www.openauthentication.org for more information.
  *
  * @author Johan Rydell, PortWise, Inc.
  */

 public class TOTP {

     private TOTP() {}

     /**
      * This method uses the JCE to provide the crypto algorithm.
      * HMAC computes a Hashed Message Authentication Code with the
      * crypto hash algorithm as a parameter.
      *
      * @param crypto: the crypto algorithm (HmacSHA1, HmacSHA256,
      *                                 HmacSHA512)
      * @param keyBytes: the bytes to use for the HMAC key
      * @param text: the message or text to be authenticated
      */


     private static byte[] hmac_sha(String crypto, byte[] keyBytes,
             byte[] text){
         try {
             Mac hmac;
             hmac = Mac.getInstance(crypto);
             SecretKeySpec macKey =
                 new SecretKeySpec(keyBytes, "RAW");
             hmac.init(macKey);
             return hmac.doFinal(text);
         } catch (GeneralSecurityException gse) {
             throw new UndeclaredThrowableException(gse);
         }
     }


     private static byte[] hexStr2Bytes(String hex){
         // Adding one byte to get the right conversion
         // Values starting with "0" can be converted
         byte[] bArray = new BigInteger("10" + hex,16).toByteArray();

         // Copy all the REAL bytes, not the "first"
         byte[] ret = new byte[bArray.length - 1];
         for (int i = 0; i < ret.length; i++)
             ret[i] = bArray[i+1];
         return ret;
     }
```

```
    private static final int[] DIGITS_POWER
    // 0 1  2   3    4     5      6       7
8
    =
{1,10,100,1000,10000,100000,1000000,10000000,100
000000 };


    /**
     * This method generates a TOTP value for
the given
     * set of parameters.
     *
     * @param key: the shared secret, HEX
encoded
     * @param time: a value that reflects a
time
     * @param returnDigits: number of digits to
return
     *
     * @return: a numeric String in base 10
that includes
     *              {@link truncationDigits}
digits
     */

    public static String generateTOTP(String
key,
            String time,
            String returnDigits){
        return generateTOTP(key, time,
returnDigits, "HmacSHA1");
    }


    /**
     * This method generates a TOTP value for
the given
     * set of parameters.
     *
     * @param key: the shared secret, HEX
encoded
     * @param time: a value that reflects a
time
     * @param returnDigits: number of digits to
return
     *
     * @return: a numeric String in base 10
that includes
     *              {@link truncationDigits}
digits
     */

    public static String generateTOTP256(String
key,
            String time,
            String returnDigits){
        return generateTOTP(key, time,
returnDigits, "HmacSHA256");
    }



    /**
     * This method generates a TOTP value for
the given
     * set of parameters.
     *
     * @param key: the shared secret, HEX
encoded
     * @param time: a value that reflects a
time
     * @param returnDigits: number of digits to
return
     *
     * @return: a numeric String in base 10
that includes
     *              {@link truncationDigits}
digits
     */

    public static String generateTOTP512(String
key,
            String time,
            String returnDigits){
        return generateTOTP(key, time,
returnDigits, "HmacSHA512");
    }


    /**
     * This method generates a TOTP value for
the given
     * set of parameters.
     *
     * @param key: the shared secret, HEX
encoded
     * @param time: a value that reflects a
time
     * @param returnDigits: number of digits to
return
     * @param crypto: the crypto function to
use
     *
     * @return: a numeric String in base 10
that includes
     *              {@link truncationDigits}
digits
     */

    public static String generateTOTP(String
key,
            String time,
            String returnDigits,
            String crypto){
        int codeDigits =
Integer.decode(returnDigits).intValue();
        String result = null;

        // Using the counter
        // First 8 bytes are for the
movingFactor
        // Compliant with base RFC 4226 (HOTP)
        while (time.length() < 16 )
            time = "0" + time;

        // Get the HEX in a Byte[]
        byte[] msg = hexStr2Bytes(time);
        byte[] k = hexStr2Bytes(key);

        byte[] hash = hmac_sha(crypto, k, msg);

        // put selected bytes into result int
        int offset = hash[hash.length - 1] &
0xf;

        int binary =
            ((hash[offset] & 0x7f) << 24) |
            ((hash[offset + 1] & 0xff) << 16) |
            ((hash[offset + 2] & 0xff) << 8) |
            (hash[offset + 3] & 0xff);

        int otp = binary %
DIGITS_POWER[codeDigits];

        result = Integer.toString(otp);
        while (result.length() < codeDigits) {
            result = "0" + result;
        }
        return result;
    }

    public static void main(String[] args) {
        // Seed for HMAC-SHA1 - 20 bytes
        String seed =
"3132333435363738393031323334353637383930";
        // Seed for HMAC-SHA256 - 32 bytes
        String seed32 =
"3132333435363738393031323334353637383930" +
        "313233343536373839303132";
```

```
        // Seed for HMAC-SHA512 - 64 bytes
        String seed64 =
"3132333435363738393031323334353637383930" +

"3132333435363738393031323334353637383930" +

"3132333435363738393031323334353637383930" +
        "31323334";
        long T0 = 0;
        long X = 30;
        long testTime[] = {59L, 1111111109L,
1111111111L,
                1234567890L, 2000000000L,
20000000000L};

        String steps = "0";
        DateFormat df = new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

df.setTimeZone(TimeZone.getTimeZone("UTC"));


        try {
            System.out.println(
                    "+---------------+---------
--------------+" +
            "------------------+--------+------
--+");
            System.out.println(
                    "|  Time(sec)    |   Time
(UTC format)   " +
            "| Value of T(Hex)  |  TOTP  | Mode
|");
            System.out.println(
                    "+---------------+---------
--------------+" +
            "------------------+--------+------
--+");

            for (int i=0; i<testTime.length;
i++) {
                long T = (testTime[i] - T0)/X;
                steps =
Long.toHexString(T).toUpperCase();
                while (steps.length() < 16)
steps = "0" + steps;
                String fmtTime =
String.format("%1$-11s", testTime[i]);
                String utcTime = df.format(new
Date(testTime[i]*1000));
                System.out.print("|  " +
fmtTime + "  |  " + utcTime +
                        "  | " + steps + " |");

System.out.println(generateTOTP(seed, steps,
"8",
                "HmacSHA1") + "| SHA1   |");
                System.out.print("|  " +
fmtTime + "  |  " + utcTime +
                        "  | " + steps + " |");

System.out.println(generateTOTP(seed32, steps,
"8",
                "HmacSHA256") + "| SHA256 |");
                System.out.print("|  " +
fmtTime + "  |  " + utcTime +
                        "  | " + steps + " |");

System.out.println(generateTOTP(seed64, steps,
"8",
                "HmacSHA512") + "| SHA512 |");

                System.out.println(
                        "+---------------+-----
------------------+" +
                "------------------+--------+--
------+");
            }
        }catch (final Exception e){
            System.out.println("Error : " + e);
        }
    }
}
```

C. Implementasi Algorima HOTP

Berikut *reference algorithm* dari RFC4226[4]:

```
/*
    * OneTimePasswordAlgorithm.java
    * OATH Initiative,
    * HOTP one-time password algorithm
    *
    */

   /* Copyright (C) 2004, OATH.  All rights
reserved.
    *
    * License to copy and use this software is
granted provided that it
    * is identified as the "OATH HOTP Algorithm"
in all material
    * mentioning or referencing this software or
this function.
    *
    * License is also granted to make and use
derivative works provided
    * that such works are identified as
    *  "derived from OATH HOTP algorithm"
    * in all material mentioning or referencing
the derived work.
    *
    * OATH (Open AuTHentication) and its members
make no
    * representations concerning either the
merchantability of this
    * software or the suitability of this
software for any particular
    * purpose.
    *
    * It is provided "as is" without express or
implied warranty
    * of any kind and OATH AND ITS MEMBERS
EXPRESSaLY DISCLAIMS
    * ANY WARRANTY OR LIABILITY OF ANY KIND
relating to this software.
    *
    * These notices must be retained in any
copies of any part of this
    * documentation and/or software.
    */

   package org.openauthentication.otp;

   import java.io.IOException;
   import java.io.File;
   import java.io.DataInputStream;
   import java.io.FileInputStream ;
   import
java.lang.reflect.UndeclaredThrowableException;

   import
java.security.GeneralSecurityException;
   import
java.security.NoSuchAlgorithmException;
   import java.security.InvalidKeyException;

   import javax.crypto.Mac;
   import javax.crypto.spec.SecretKeySpec;


   /**
    * This class contains static methods that
are used to calculate the
    * One-Time Password (OTP) using
    * JCE to provide the HMAC-SHA-1.
    *
    * @author Loren Hart
    * @version 1.0
    */
```

```
   public class OneTimePasswordAlgorithm {
       private OneTimePasswordAlgorithm() {}

       // These are used to calculate the check-
sum digits.
       //                                 0  1  2
3  4  5  6  7  8  9
       private static final int[] doubleDigits =
                        { 0, 2, 4, 6, 8, 1, 3, 5,
7, 9 };

       /**
        * Calculates the checksum using the
credit card algorithm.
        * This algorithm has the advantage that
it detects any single
        * mistyped digit and any single
transposition of
        * adjacent digits.
        *
        * @param num the number to calculate the
checksum for
        * @param digits number of significant
places in the number
        *
        * @return the checksum of num
        */
       public static int calcChecksum(long num,
int digits) {
           boolean doubleDigit = true;
           int     total = 0;
           while (0 < digits--) {
               int digit = (int) (num % 10);
               num /= 10;
               if (doubleDigit) {
                   digit = doubleDigits[digit];
               }
               total += digit;
               doubleDigit = !doubleDigit;
           }
           int result = total % 10;
           if (result > 0) {
               result = 10 - result;
           }
           return result;
       }

       /**
        * This method uses the JCE to provide
the HMAC-SHA-1

       * algorithm.
        * HMAC computes a Hashed Message
Authentication Code and
        * in this case SHA1 is the hash
algorithm used.
        *
        * @param keyBytes   the bytes to use for
the HMAC-SHA-1 key
        * @param text       the message or text
to be authenticated.
        *
        * @throws NoSuchAlgorithmException if no
provider makes
        *       either HmacSHA1 or HMAC-SHA-1
        *       digest algorithms available.
        * @throws InvalidKeyException
        *       The secret provided was not a
valid HMAC-SHA-1 key.
        *
        */

       public static byte[] hmac_sha1(byte[]
keyBytes, byte[] text)
           throws NoSuchAlgorithmException,
InvalidKeyException
       {
   //        try {
             Mac hmacSha1;
             try {
                 hmacSha1 =
Mac.getInstance("HmacSHA1");
             } catch (NoSuchAlgorithmException
nsae) {
                 hmacSha1 =
Mac.getInstance("HMAC-SHA-1");
             }
             SecretKeySpec macKey =
         new SecretKeySpec(keyBytes, "RAW");
             hmacSha1.init(macKey);
             return hmacSha1.doFinal(text);
   //        } catch (GeneralSecurityException
gse) {
   //            throw new
UndeclaredThrowableException(gse);
   //        }
       }

       private static final int[] DIGITS_POWER
     // 0 1  2   3    4     5      6       7
8
     =
{1,10,100,1000,10000,100000,1000000,10000000,100
000000};

       /**
        * This method generates an OTP value for
the given
        * set of parameters.
        *
        * @param secret       the shared secret
        * @param movingFactor the counter, time,
or other value that
        *                     changes on a per
use basis.
        * @param codeDigits   the number of
digits in the OTP, not
        *                     including the
checksum, if any.
        * @param addChecksum  a flag that
indicates if a checksum digit


        *                     should be appended
to the OTP.
        * @param truncationOffset the offset
into the MAC result to
        *                     begin truncation.
If this value is out of
        *                     the range of 0 ...
15, then dynamic
        *                     truncation  will
be used.
        *                     Dynamic truncation
is when the last 4
        *                     bits of the last
byte of the MAC are
        *                     used to determine
the start offset.
        * @throws NoSuchAlgorithmException if no
provider makes
        *                     either HmacSHA1 or
HMAC-SHA-1
        *                     digest algorithms
available.
        * @throws InvalidKeyException
        *                     The secret
provided was not
        *                     a valid HMAC-SHA-1
key.
        *
        * @return A numeric String in base 10
that includes
        * {@link codeDigits} digits plus the
optional checksum
        * digit if requested.
        */
       static public String generateOTP(byte[]
secret,
                  long movingFactor,
              int codeDigits,
```

```
                boolean addChecksum,
              int truncationOffset)
            throws NoSuchAlgorithmException,
InvalidKeyException
        {
            // put movingFactor value into text
byte array
      String result = null;
      int digits = addChecksum ? (codeDigits + 1)
: codeDigits;
            byte[] text = new byte[8];
            for (int i = text.length - 1; i >= 0;
i--) {
                text[i] = (byte) (movingFactor &
0xff);
                movingFactor >>= 8;
            }

            // compute hmac hash
            byte[] hash = hmac_sha1(secret,
text);

            // put selected bytes into result int
            int offset = hash[hash.length - 1] &
0xf;
      if ( (0<=truncationOffset) &&
             (truncationOffset<(hash.length-4)) )
{
          offset = truncationOffset;
      }
            int binary =
                ((hash[offset] & 0x7f) << 24)
                | ((hash[offset + 1] & 0xff) <<
16)
                | ((hash[offset + 2] & 0xff) <<
8)

                | (hash[offset + 3] & 0xff);

            int otp = binary %
DIGITS_POWER[codeDigits];
      if (addChecksum) {
          otp =  (otp * 10) + calcChecksum(otp,
codeDigits);
      }
      result = Integer.toString(otp);
      while (result.length() < digits) {
          result = "0" + result;
      }
      return result;
        }
    }
```

V. KESIMPULAN

Dari penjelasan mengenai 2-step verification yang telah dituliskan di atas dapat disimpulkan bahwa:

1. *2-step verification* dapat meningkatkan keamanan akun dan mencegah pengambilan data-data pribadi atau pembajakan dari pihak-pihak yang tidak berhak dan untuk meretasnya perlu memikirkan 1.000.000 kemungkinan kunci yang tidak mungkin dilakukan selama 30 detik.
2. Kunci untuk tahap verifikasi kedua dibangun dari kombinasi enam kunci yang berubah setiap 30 detik. Kombinasi ini bila dihitung dengan kombinatorial yaitu terdapat 1.000.000 kemungkinan kunci setiap 30 detik.
3. Terdapat dua algoritma untuk membangkitkan kunci tersebut yaitu HMAC-Based One-time Password (HOTP) untuk membangkitakan kunci yang dipicu oleh *event* tertentu dan algoritma Time-based One-time Password (TOTP) untuk membangkitkan kunci sesuai dengan waktunya yaitu setiap 30 detik.

VII. UCAPAN TERIMAKASIH

Penulis ingin mengucapkan terima kasih kepada pihak-pihak yang telah membantu saya dalam mengerjakan makalah berikut ini:

1. Pengajar matakuliah IF 2120 Matematika Diskrit, Bapak Rinaldi Munir dan Ibu Harlili yang telah mengajar selama satu semester di matakuliah ini.
2. Teman-teman khususnya Setyo dan Arief yang telah memberikan ide inspirasi untuk membuat makalah mengenai *2-step verification* ini
3. Dan seluruh pihak-pihak yang telah membantu yang tidak bisa disebutkan satu persatu.

DAFTAR PUSTAKA

[1] Munir, Rinaldi, Diktat Kuliah IF2091 – Struktur Diskrit. Bandung: Program Studi Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2008.
[2] https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2 , 16 Desember 2013, 10.05 WIB
[3] http://tools.ietf.org/pdf/rfc4226.pdf ,16 Desember 2013, 11.05 WIB
[4] https://tools.ietf.org/html/rfc4226 ,16 Desember 2013, 11.05 WIB
[5] http://tools.ietf.org/html/rfc3548 ,16 Desember 2013, 11.05 WIB
[6] https://tools.ietf.org/html/rfc6238 ,16 Desember 2013, 11.05 WIB
[7] https://code.google.com/p/google-authenticator/ ,16 Desember 2013, 11.05 WIB
[8] http://mbmccormick.com/2013/01/use-facebooks-2-factor-authentication-with-third-party-totp-generators/ ,16 Desember 2013, 11.05 WIB
[9] http://www.techrepublic.com/blog/google-in-the-enterprise/secure-your-google-account-with-two-step-authentication/1322/,16 Desember 2013, 11.05 WIB
[10] https://support.google.com/accounts/answer/1066447?hl=id ,16 Desember 2013, 11.05 WIB

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Desember 2013

Aisyah Dzulqaidah 13510005