Implementing SHA-3 *for Secure One Time Password Authentication in Mobile Banking*

Verawati Esteria S. Simatupang (18220002)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
verawati4553@gmail.com

***Abstract*—Makalah ini bertujuan untuk menjelaskan bagaimana SHA-3 dapat digunakan untuk mengautentikasi *One Time Password* pada layanan *mobile banking*. Pada perkembangan teknologi saat ini, masalah keamanan *cyber* menjadi permasalahan yang selalu dibahas dan semakin kompleks. Oleh sebab itu, diperlukan keamanan yang kuat dalam mengatasi tantangan tersebut, salah satunya di sektor perbankan. Pada makalah ini, autentikasi *One Time Password* pada *mobile banking* akan dilakukan pada saat registrasi akun dan penarikan tunai. Hal ini disebabkan oleh kedua kegiatan tersebut memerlukan data kredensial pengguna. Oleh sebab itu, dilakukan proses autentikasi *One Time Password*. Hasil dari penelitian ini menunjukkan bahwa SHA-3 memberikan solusi yang kuat dan andal dalam mengamankan autentikasi *One Time Password* di *mobile banking* dan memastikan kerahasiaan informasi pengguna.**

Keywords—SHA-3, One Time Password, Autentikasi, Mobile Banking

I. Pendahuluan

Pada saat ini, penggunaan aplikasi *mobile* sangat banyak digunakan oleh masyarakat. Oleh sebab itu, keamanan harus selalu diperhatikan saat menggunakan aplikasi. Salah satu aspek yang sangat penting terkait keamanan adalah di bidang finansial karena itu menyangkut uang yang digunakan untuk memenuhi kebutuhan. Saat ini, layanan perbankan melalui aplikasi *mobile* menjadi salah satu cara utama bagi pengguna untuk mengakses rekening, melakukan transaksi, dan mengelola keuangan secara efisien. Kemudahan dalam menggunakan aplikasi *mobile*, disertai dengan meningkatnya penggunaan aplikasi *mobile.* Hal ini dapat membawa risiko keamanan yang semakin kompleks, terutama terkait dengan informasi sensitif seperti kata sandi dan informasi keuangan. Oleh karena itu, dalam melindungi pengguna dari ancaman keamanan seperti pencurian identitas dan serangan peretas, diperlukan metode autentikasi yang kuat dan aman.

One Time Password (*OTP*) merupakan salah satu metode autentikasi yang kuat dan aman karena dapat memberikan lapisan keamanan terhadap data yang akan disimpan pada aplikasi. *OTP* adalah kode yang hanya digunakan satu kali untuk mengamankan proses autentikasi dan transaksi. Namun, untuk memastikan keamanan yang optimal, diperlukan implementasi algoritma kriptografi yang kuat, salah satunya adalah SHA-3 (Secure Hash Algorithm 3). SHA-3 dikenal luas karena kemampuannya dalam menghasilkan nilai hash yang unik sehingga tahan terhadap serangan keamanan *cyber*. Pada makalah ini, akan dijelaskan mengenai implementasi SHA-3 dalam autentikasi *One Time Password* pada aplikasi *mobile banking* saat melakukan registrasi dan penarikan uang tunai.

II. Metode

A. Literature Review

Penulis memulai pembuatan makalah ini dengan melakukan *literature review*, yang melibatkan pencarian informasi dari sumber - sumber digital, seperti jurnal dan paper tentang topik - topik berikut : SHA-3, *cardless transaction*, *One Time Password*, dan cara kerja *Secure Cardless with One Time Password*.

B. Eksperimen

Selanjutnya, penulis melakukan eksperimen dengan membuat program aplikasi sederhana berbasis desktop *mobile banking*. Program ini ditulis dengan menggunakan Python dengan *library* tkinter.

Pada percobaan ini, implementasi *One Time Password* akan menggunakan algoritma hash SHA-3. Terdapat 2 jenis autentikasi dengan *One Time Password* yang digunakan, yaitu pada saat melakukan registrasi dan tarik tunai dengan durasi penggunaan *OTP* adalah 5 menit.

III. Landasan Teori

A. SHA-3 (Secure Hash Algorithm 3)

SHA-3 juga dikenal sebagai Secure Hash Algorithm 3, adalah fungsi *hash* kriptografi yang dirancang untuk menghasilkan nilai *hash* berukuran tetap dari *input* data dengan panjang sembarang. SHA-3 didasarkan pada konstruksi berbasis permutasi. SHA-3 menawarkan tingkat keamanan dan ketahanan yang tinggi terhadap berbagai serangan kriptografi, termasuk serangan *collision*, *preimage*, dan *second preimage*. Algoritma ini memberikan panjang keluaran yang fleksibel dan mampu memproses data secara efisien dan mampu memproses data secara efisien. SHA-3 telah distandarisasi oleh NIST (National Institute of Standards

and Technology) dan menawarkan solusi andal untuk mentransmisikan integritas data, autentikasi, dan transmisi pesan yang aman di berbagai aplikasi dan protokol. [1]

SHA-3 merupakan fungsi *hash* kriptografi yang menggunakan *sponge construction*. *Sponge construction* menggabungkan dua fungsi yang berbeda, yaitu fungsi penyerapan (*absorbing function*) dan fungsi pemerasan (*squeezing function*), untuk menghasilkan fungsi yang aman dan efisien. [1]

Berikut adalah ilustrasi implementasi fungsi *hash* SHA-3.

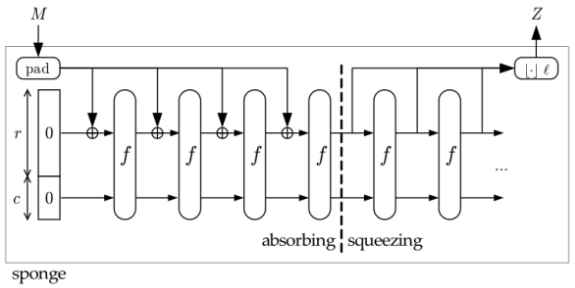


Fig. 1. Ilustrasi Implementasi Fungsi *Hash* SHA-3 (sumber: [2])

Berikut adalah langkah praproses pada fungsi hash SHA-3 [2] :

1. Tentukan terlebih dahulu panjang *digest* yang diinginkan misalnya d bit.
2. Masukan berupa pesan M, akan ditambahkan dengan bit - bit *padding* yang kemudian akan menjadi *string* S sehingga habis dibagi dengan r.
3. P dipecah menjadi blok - blok P_i berukuran r-bit
4. b-bit merupakan gabungan dari r-bit dan c-bit. Peubah status (*state*) S yang berukuran b-bit diinisialisasi ke nol.
5. *Sponge construction* akan berlangsung pada dua fase yaitu fase penyerapan (*absorbing*) dan fase pemerasan (*squeezing*).

Berikut adalah ilustrasi fase penyerapan (*absorbing*) dan fase pemerasan (*squeezing*).

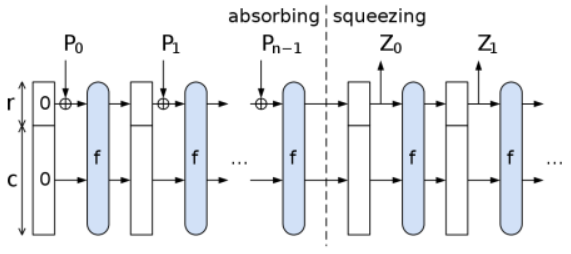


Fig. 2. Ilustrasi Fase *Absorbing* dan Fase *Squeezing* (sumber: [2])

Berikut adalah proses pada fase penyerapan (*absorbing*) [2] :

1. Untuk setiap blok masukan P_i yang berukuran r-bit, akan dilakukan XOR dengan r-bit pertama dari *state* S.
2. Hasil tersebut akan dimasukkan ke dalam fungsi permutasi f sehingga menghasilkan *state* baru S.
3. Proses ini akan terus berlangsung hingga semua blok masukan selesai diproses.
4. Selanjutnya, *sponge construction* akan beralih ke fase pemerasan (*squeezing*)

Berikut adalah proses pada fase pemerasan (*squeezing*) [2] :

1. *Message digest* akan disimpan di dalam Z.
2. Z akan diinisialisasi dengan *string* kosong *(null string*)
3. r-bit pertama dari *state* S akan disambungkan (*append*) ke Z.
4. Hasilnya akan dimasukkan ke dalam fungsi permutasi f menghasilkan *state* baru S.
5. Proses tersebut akan terus dilakukan hingga panjang Z sama dengan d.

B. *Cardless Transaction*

Menurut Kitti Phothikitti [4], *Cardless ATM* adalah sebuah layanan teknologi baru yang menawarkan keunggulan berupa kenyaman kepada pengguna dengan menggunakan teknologi *smartphone* untuk melakukan penarikan tunai melalui akun mereka sendiri. Ketika melakukan penarikan tunai di *ATM* tanpa kartu, aplikasi pada *smartphone* (*mobile banking*) akan menghasilkan kunci autentikasi yang dapat berupa kode numerik yang akan dimasukkan atau memindai kode *QR* pada *ATM*. Setelah melakukan hal tersebut, *ATM* akan memproses transaksi dan mengeluarkan uang tunai tanpa memasukkan kartu.

C. *One Time Password*

One Time Password merupakan *password* yang hanya berlaku untuk satu kali / sesi. *One Time Password* akan selalu berubah walaupun pengguna melakukan *input* yang sama. Hal ini yang membuat *One Time Password* menjadi salah satu metode autentikasi yang terpopuler saat ini karena memberikan autentikasi yang aman. [3]

Terdapat dua pendekatan dalam menghasilkan *OTP*, yaitu berdasarkan token yang disinkronkan dengan waktu dan berdasarkan algoritma matematika. [3]

Pertama, *OTP* berdasarkan token yang disinkronkan dengan waktu. *OTP* yang disinkronkan dengan waktu menggunakan *hardware* disebut dengan token keamanan. Token ini biasanya hanya berlaku untuk jangka waktu yang singkat. Pada sistem *OTP* ini, waktu merupakan bagian yang penting dalam pembangkitan kode *OTP* karena pembuatan *OTP* didasarkan pada waktu saat ini. [3]

Kedua, *OTP* berdasarkan algoritma matematika. *OTP* berdasarkan pendekatan algoritma matematika, menggunakan dua metode, yaitu berdasarkan kata sandi sebelumnya dan berdasarkan tantangan. *OTP* yang dihasilkan berdasarkan kata sandi sebelumnya secara efektif membentuk suatu rangkaian

dan harus digunakan dalam urutan yang telah ditentukan. *OTP* berdasarkan tantangan diambil dari nomor acak yang dipilih oleh server autentikasi atau detail transaksi atau penghitung yang digunakan. [3]

D. Cara Kerja *Secure Cardless with One Time Password*

Berdasarkan Sruthi. M, Swapna. M.P. [5], cara kerja dari *Secure Cardless With One Time Password* adalah ketika pengguna memerlukan *OTP* dari sistem *ATM*, maka *OTP* harus dibuat pada saat itu hanya dengan waktu saat ini dan data pengguna yang tersedia dalam basis data dan *OTP* akan dikirim ke ponsel pengguna yang terdaftar. Pada saat pembukuan rekening, sistem bank akan menanyakan mengenai registrasi ponsel berupa nomor ponsel pengguna, Informasi ini akan disimpan dalam basis data bank. Setiap kali pengguna memasukkan nomor rekening ke mesin *ATM*, sistem membutuhkan *PIN* untuk mengautentikasi pengguna. Jika nomor *PIN* tersebut telah diverifikasi, maka *OTP* akan dibuat dan dikirim ke nomor ponsel pengguna.

Transaksi akan berhasil jika *OTP* yang dimasukkan valid, jika tidak maka transaksi akan gagal. Terdapat batasan maksimal atau *limit* input *OTP*, jika melebihi *limit* tersebut maka kartu akan diblokir.

IV. Implementasi

A. Lingkungan Implementasi

Implementasi autentikasi *One Time Password* di *mobile banking* dengan algoritma SHA-3 menggunakan bahasa Python. Hal ini disebabkan oleh kemudahan dalam mengembangkan sistem ini. Algoritma SHA-3 yang digunakan adalah SHA-3 256 sehingga *output*-nya sepanjang 256 bit.

Berikut merupakan *library* yang digunakan dalam pengembangan autentikasi *One Time Password* di *mobile banking*.

1. hashlib

   hashlib merupakan *library* yang digunakan untuk melakukan *hash* menggunakan algoritma SHA-3 256 bit.

2. tkinter

   tkinter merupakan *library* yang digunakan untuk membangun antarmuka pengguna (*GUI*) dalam aplikasi Python sehingga pengguna akan merasakan bahwa sedang menggunakan *mobile banking*.

3. psycopg2

   psycopg2 merupakan *library* yang digunakan untuk koneksi dengan *database* PostgreSQL. Pada percobaan ini, menggunakan *database* datapengguna dengan tiga tabel yaitu datapengguna, datalogin, dan datatransaksi.

4. datetime

   datetime merupakan *library* yang digunakan untuk mengakses dan memanipulasi informasi waktu dalam Python. Informasi waktu akan menjadi bagian dari masukan untuk pembangkitan kode *OTP* sehingga hasil *hash* lebih unik.

B. Terhubung dengan *Database*

Berikut adalah kode yang digunakan untuk terhubung dengan *database*.

```
import psycopg2

def connect_to_database():
    try:
        connection = psycopg2.connect(
            host="127.0.0.1",
            database="datapengguna",
            user="postgres",
            password="postgres",
            port=5432
        )

        cursor = connection.cursor()
        query = "SELECT * FROM nama_tabel"
        cursor.execute(query)
        connection.commit()

        cursor.close()
        connection.close()
    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL",
error)
```

Fig. 3. Kode Terhubung dengan *Database* (sumber: dokumentasi pribadi)

Untuk dapat terhubung dan berinteraksi dengan *database* PostgreSQL, langkah pertama yang harus dilakukan adalah meng-*import module* psycopg2. Lalu, lakukan koneksi dengan *database* yang digunakan. Sesuaikan *host*, *database*, *user*, *password*, dan *port* yang ingin dihubungkan.

Setelah koneksi dengan *database* berhasil dibuat, dibuat sebuah objek cursor dengan menggunakan metode connection.cursor(). Cursor ini digunakan untuk mengeksekusi pernyataan SQL dan memanipulasi data di dalam database. cursor.execute() digunakan untuk mengeksekusi query sql. Kemudian di akhir, cursor dan connection ditutup.

C. Pembangkitan Kode *OTP*

Pada percobaan ini, pembangkitan kode *OTP* akan dilakukan pada dua hal, yaitu pada saat registrasi dan penarikan tunai. Hal ini dilakukan karena dua kegiatan tersebut membutuhkan keamanan yang lebih pada saat menggunakan *mobile banking*.

Berikut adalah kode yang digunakan untuk pembangkitan kode *OTP* pada saat registrasi akun.

```
def get_register_time(self):
    return datetime.now()
```

```
def generate_otp(self):
    input_string = self.entry_1.get() + self.entry_2.get() +
```

```
self.entry_3.get() + self.entry_4.get() + self.entry_5.get() +
str(self.get_register_time())
        hashed_string =
hashlib.sha256(input_string.encode()).hexdigest()
        otp = hashed_string[:6]
        numeric_otp = int(otp, 16) % 1000000
        return numeric_otp
```

Fig. 4. Kode Pembangkitan Kode *OTP* Registrasi (sumber: dokumentasi pribadi)

Pada saat melakukan registrasi akun, akan diminta beberapa masukan kepada pengguna, seperti nomor rekening, *email*, *username*, *password*, dan *PIN*. self.entry_1.get() akan mengambil masukan berupa nomor rekening, self.entry_2.get() akan mengambil masukan berupa *email*, self.entry_3.get() akan mengambil masukan berupa *username*, self.entry_4.get() akan mengambil masukan berupa *password*, dan self.entry_5.get() akan mengambil masukan berupa *PIN*. Input tersebut akan menjadi masukan untuk melakukan pembangkitan kode *OTP*. Selain itu, terdapat masukan lainnya untuk pembangkitan kode *OTP* berupa waktu saat pengguna melakukan registrasi. Hal ini dilakukan agar kode *OTP* yang dihasilkan lebih unik.

Berikut adalah kode yang digunakan untuk pembangkitan kode *OTP* saat melakukan penarikan tunai.

```
def get_logged_in_account(self):
        try:
            connection = psycopg2.connect(
                host="127.0.0.1",
                database="datapengguna",
                user="postgres",
                password="postgres",
                port=5432
            )

            cursor = connection.cursor()

            logged_in_account_query = "SELECT username
FROM datalogin ORDER BY login_time DESC LIMIT 1"
            cursor.execute(logged_in_account_query)
            logged_in_account = cursor.fetchone()[0]
            return logged_in_account

        except (Exception, psycopg2.Error) as error:
            print("Error while connecting to PostgreSQL",
error)
            messagebox.showerror("Error", "An error occurred
while retrieving logged-in account")
```

```
def get_nomor_rekening(self):
        try:
            connection = psycopg2.connect(
                host="127.0.0.1",
                database="datapengguna",
                user="postgres",
                password="postgres",
                port=5432
            )

            cursor = connection.cursor()

            logged_in_account_query = "SELECT username
FROM datalogin ORDER BY login_time DESC LIMIT 1"
            cursor.execute(logged_in_account_query)
            logged_in_account = cursor.fetchone()[0]

            query = f"SELECT dp.nomor_rekening FROM
datapengguna dp JOIN datalogin dl ON dp.username =
dl.username WHERE dl.username =
'{logged_in_account}'"
            cursor.execute(query)
            result = cursor.fetchone()[0]

            if result:
                nomor_rekening = result
                self.canvas.create_text(
                    180.0,
                    100.0,
                    anchor="nw",
                    text=nomor_rekening,
                    fill="#000000",
                    font=("MontserratRoman SemiBold", 14 * -1)
                )
                return nomor_rekening
            else:
                messagebox.showerror("Error", "Account not
found.")

        except (Exception, psycopg2.Error) as error:
            print("Error while connecting to PostgreSQL",
error)
            messagebox.showerror("Error", "An error occurred
while saving data to the database")
```

```
def get_cashwithdrawal_time(self):
        return datetime.now()
```

```
def generate_otp(self):
        input_string = self.entry_1.get() +
self.get_logged_in_account() + self.get_nomor_rekening()
+ self.radio1.get() + self.cbb1.get() +
str(self.get_cashwithdrawal_time())
        hashed_string =
hashlib.sha256(input_string.encode()).hexdigest()
        otp = hashed_string[:6]
        numeric_otp = int(otp, 16) % 1000000
        return numeric_otp
```

Fig. 5. Kode Pembangkitan Kode *OTP* Penarikan Tunai (sumber: dokumentasi pribadi)

Pada saat melakukan penarikan tunai, akan diminta beberapa masukan kepada pengguna, seperti jalur tarik tunai,

nominal penarikan, dan *PIN*. self.entry_1.get() akan mengambil masukan berupa *PIN*, self.radio1.get() akan mengambil masukan berupa jumlah nominal penarikan, dan self.cbb1.get() akan mengambil masukan berupa jalur tarik tunai. *Input* tersebut akan menjadi masukan untuk melakukan pembangkitan kode *OTP*. Selain itu, terdapat masukan lainnya untuk melakukan pembangkitan kode *OTP* berupa *username*, nomor rekening, dan waktu saat pengguna melakukan penarikan tunai pada *mobile banking*. Hal ini dilakukan agar kode *OTP* yang dihasilkan lebih unik.

D. Verifikasi Kode *OTP*

Berikut adalah kode yang digunakan untuk verifikasi kode *OTP* pada saat registrasi akun.

```
def verify_otp(self):
    try:
        connection = psycopg2.connect(
            host="127.0.0.1",
            database="datapengguna",
            user="postgres",
            password="postgres",
            port=5432
        )

        cursor = connection.cursor()

        logged_in_account_query = "SELECT username
FROM datapengguna ORDER BY register_time DESC
LIMIT 1"
        cursor.execute(logged_in_account_query)
        logged_in_account = cursor.fetchone()[0]

        query = f"SELECT otp, register_time FROM
datapengguna where username = '{logged_in_account}'"
        cursor.execute(query)
        result = cursor.fetchone()

        otp_time = result[1]
        current_time = datetime.now()

        if result[0] == str(self.entry_1.get()) and
current_time < otp_time + timedelta(minutes=5) :
            self.clear()
            messagebox.showinfo("Success", "Registration
successfully")
            self.origin.Login()
        elif current_time > otp_time + timedelta(minutes=5)
:
            self.clear()
            messagebox.showwarning("Times Up",
"Registration failed")
            delete_query = f"DELETE FROM datapengguna
where username = '{logged_in_account}'"
            cursor.execute(delete_query)
            connection.commit()
            self.origin.Register()
        elif result[0] != str(self.entry_1.get()) :
            self.clear()
            messagebox.showerror("Error", "Invalid otp")

    except (Exception, psycopg2.Error) as error:
        print("Error while connecting to PostgreSQL",
error)
        messagebox.showerror("Error", "An error occurred
while saving data to the database")
```

Fig. 6. Kode Verifikasi *OTP* (sumber: dokumentasi pribadi)

Verifikasi kode *OTP* akan dilakukan untuk kode *OTP* pada saat registrasi. Hal ini dikarenakan untuk melakukan verifikasi kode *OTP* saat registrasi dilakukan di mobile banking, sedangkan verifikasi kode *OTP* saat penarikan tunai dilakukan pada jalur penarikan tunai yang dipilih, apakah itu melalui *ATM*, Indomaret, atau agen Bank.

Kode *OTP* yang dihasilkan hanya dapat digunakan selama 5 menit. Jika kode *OTP* yang dihasilkan dimasukkan lebih dari 5 menit, walaupun input yang dimasukkan benar, maka sistem tidak akan menerima kode tersebut dan registrasi dianggap tidak berhasil sehingga proses registrasi harus diulang.

V. Pembahasan

Pengujian akan dilakukan berdasarkan kasus uji untuk beberapa *use case* dari sistem. Berikut adalah rencana pengujian.

TABLE I. Rencana Pengujian

ID	*Use Case*	Jenis Pengujian	*ID* Uji
1	*Use Case* menjalankan sistem	BlackBox Testing	U-1 01
2	*Use Case* melakukan registrasi	BlackBox Testing	U-1 02
3	*Use Case* melakukan *login*	BlackBox Testing	U-1 03
4	*Use Case* menyimpan input pengguna ke *database*	BlackBox Testing	U-1 04
5	*Use Case* melihat data di *database*	BlackBox Testing	U-1 05
6	*Use Case* mendapatkan kode *OTP* saat registrasi dan penarikan tunai	BlackBox Testing	U-1 06
7	*Use Case* memverifikasi kode *OTP* registrasi	BlackBox Testing	U-1-07
8	*Use Case* melakukan *logout*	BlackBox Testing	U-1 08

A. *Use Case* menjalankan sistem

Untuk dapat menjalankan sistem, terlebih dahulu membuat *database* PostgreSQL dengan nama datapengguna dan sesuaikan *host*, *user*, *password*, dan *port*. Lalu *connect database* tersebut pada pgAdmin 4. Pada *database* tersebut, buat tiga tabel yaitu, datapengguna, datalogin, dan datatransaksi. Untuk kode, jalankan Main.py. Maka, akan muncul tampilan halaman *Splash* seperti berikut.

Fig. 7. Tampilan Halaman *Splash* (sumber: dokumentasi pribadi)

Berikut adalah *query* dalam pembuatan ketiga tabel.

```
CREATE TABLE datapengguna (
  nomor_rekening VARCHAR(355),
  email VARCHAR(355),
  username VARCHAR(355),
  password VARCHAR(355),
  pin VARCHAR(50),
  otp VARCHAR(50),
  register_time TIMESTAMP
);
```

```
CREATE TABLE datalogin (
  username VARCHAR(355),
  password VARCHAR(355),
  login_time TIMESTAMP
);
```

```
CREATE TABLE datatransaksi (
  username VARCHAR(355),
  nomor_rekening VARCHAR(355),
  jalur_tarik_tunai VARCHAR(355),
  nominal VARCHAR(355),
  pin VARCHAR(355),
  otp VARCHAR(355),
  cashwithdrawal_time TIMESTAMP
);
```

Fig. 8. *Query* pembuatan tabel (sumber: dokumentasi pribadi)

B. *Use Case* melakukan registrasi

Untuk dapat melakukan registrasi, tekan tombol "Belum Punya Akun" pada halaman Splash. Lalu, akan ditampilkan halaman Registrasi. Isi semua *entry* yang diminta dimana terdapat beberapa persyaratan yaitu nomor rekening harus berisi angka, *email* harus diisi sesuai dengan format *email*, *username* dan *password* minimal harus terdiri dari satu huruf besar, huruf kecil, dan angka, dan *PIN* harus terdiri dari 6 angka. Lalu, tekan tombol "Register".

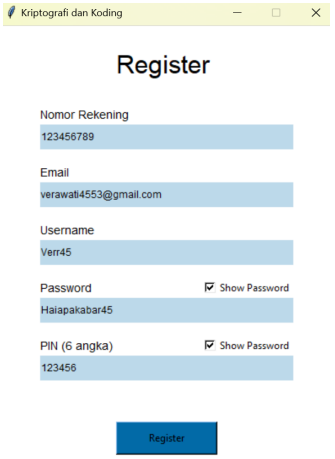


Fig. 9. Tampilan Halaman Registrasi (sumber: dokumentasi pribadi)

C. *Use Case* melakukan *login*

Pada halaman *login*, masukkan *username* dan *password* yang telah terdaftar sebelumnya untuk dapat memasuki halaman *Home*.

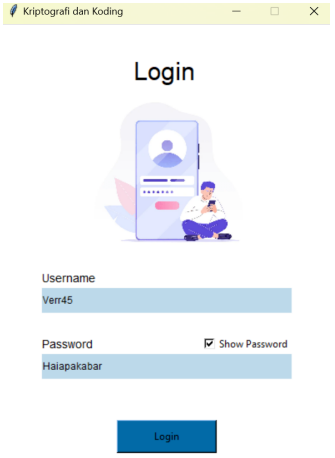


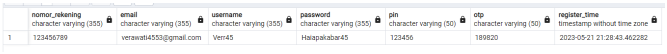
Fig. 10. Tampilan Halaman *Login* (sumber: dokumentasi pribadi)

D. *Use Case* menyimpan *input* pengguna ke *database*

Input-an pengguna pada halaman register, *login*, dan penarikan tunai akan disimpan pada *database* setelah menekan tombol yang terdapat pada halaman tersebut.

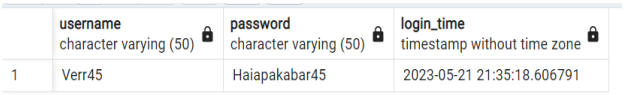
E. *Use Case* melihat data di *database*

Setelah tombol pada halaman register, *login*, dan penarikan tunai ditekan, maka *input*-an data pengguna akan disimpan pada *database*. Berikut adalah data yang ada di *database*.



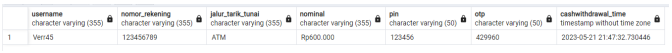
	nomor_rekening character varying (355)	email character varying (355)	username character varying (355)	password character varying (355)	pin character varying (50)	otp character varying (50)	register_time timestamp without time zone
1	123456789	verawati4553@gmail.com	Verr45	Haiapakabar45	123456	189820	2023-05-21 21:28:43.462282

Fig. 11. Tabel datapengguna (sumber: dokumentasi pribadi)



	username character varying (50)	password character varying (50)	login_time timestamp without time zone
1	Verr45	Haiapakabar45	2023-05-21 21:35:18.606791

Fig. 12. Tabel datalogin (sumber: dokumentasi pribadi)



	username character varying (355)	nomor_rekening character varying (355)	jalur_tarik_tunai character varying (355)	nominal character varying (355)	pin character varying (50)	otp character varying (50)	cashwithdrawal_time timestamp without time zone
1	Verr45	123456789	ATM	Rp600.000	123456	429960	2023-05-21 21:47:32.730446

Fig. 13. Tabel datatransaksi (sumber: dokumentasi pribadi)

F. Use Case mendapatkan kode *OTP* saat registrasi dan penarikan tunai

Kode *OTP* pada saat registrasi akun dan penarikan tunai akan muncul setelah mengisi seluruh *entry* yang dibutuhkan dan menekan tombol "Register" dan "Konfirmasi". Kode *OTP* akan tersedia hanya selama 5 menit.

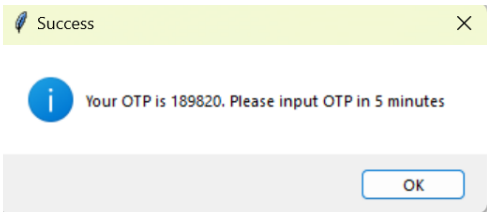


Fig. 14. Tampilan Kode *OTP* Registrasi Akun (sumber: dokumentasi pribadi)

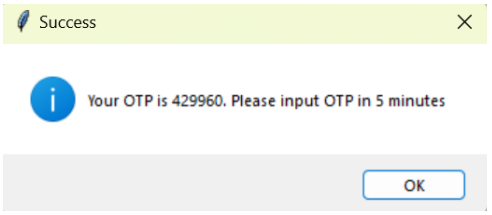


Fig. 15. Tampilan Kode *OTP* Penarikan Tunai (sumber: dokumentasi pribadi)

G. Use Case memverifikasi kode *OTP* registrasi

Setelah mendapatkan kode *OTP*, maka akan diteruskan ke halaman verifikasi kode *OTP*. Masukkan kode OTP tersebut dengan maksimal waktu 5 menit. Jika memasukkan kode *OTP* di luar 5 menit, maka dianggap tidak valid dan harus mengulangi proses registrasi.

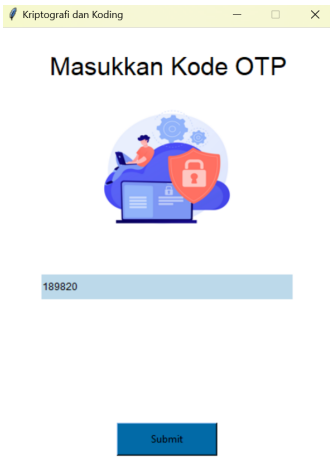


Fig. 16. Tampilan Verifikasi Kode *OTP* (sumber: dokumentasi pribadi)

H. Use Case melakukan *logout*

Untuk keluar dari sistem, maka pada tampilan *Home*, tekan tombol "Keluar", maka akan diarahkan ke halaman *Splash*.

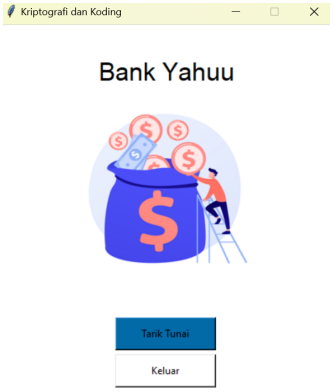


Fig. 17. Tampilan Halaman *Home* (sumber: dokumentasi pribadi)

Berdasarkan pengujian di atas, maka dapat disimpulkan hasil pengujian terhadap sistem.

TABLE II. HASIL PENGUJIAN

ID Uji	Kriteria Evaluasi Hasil	Hasil yang Didapat	Kesimpulan
U-1 01	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima
U-1 02	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima
U-1 03	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima
U-1 04	Keluaran yang	Keluaran yang	Diterima

	muncul sesuai	muncul sesuai	
U-1 05	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima
U-1 06	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima
U-1-07	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima
U-1 08	Keluaran yang muncul sesuai	Keluaran yang muncul sesuai	Diterima

VI. Kesimpulan dan Saran

Berdasarkan hasil pembahasan di atas, dapat disimpulkan bahwa SHA-3 memberikan solusi yang kuat dan andal dalam mengamankan autentikasi *One Time Password* di *mobile banking* dan memastikan kerahasiaan informasi pengguna. Hal ini bermanfaat untuk meningkatkan keamanan saat melakukan transaksi melalui bank. Namun untuk lebih menjaga keamanan saat bertransaksi, disarankan untuk menambahkan metode autentikasi berdasarkan faktor unik yang hanya dimiliki oleh 1 pengguna seperti *biometric*. Langkah ini dilakukan untuk membantu adanya ancaman pada keamanan *cyber*.

Video Link at Youtube

Berikut adalah *link* Youtube penjelasan terkait sistem implementasi SHA-3 untuk autentikasi *One Time Password* pada *mobile banking*. https://youtu.be/9rRBuV4Z5_k

Source Code

Berikut adalah *source code* pembuatan sistem implementasi SHA-3 untuk autentikasi *One Time Password* pada *mobile banking*. https://github.com/verawatisimatupang/Tugas-Akhir-Kriptografi-dan-Koding.git

Ucapan Terima Kasih

Puji dan syukur saya ucapkan kepada Tuhan yang Maha Esa yang telah membantu saya dalam menyelesaikan pembuatan makalah ini. Saya juga mengucapkan terima kasih kepada kedua orangtua saya yang selalu memberikan dukungan dan semangat sehingga makalah ini dapat terselesaikan. Selain itu, saya ingin menyampaikan rasa terima kasih kepada Bapak Rinaldi Munir, Dosen II4031 Kriptografi dan Koding, yang telah mengajarkan mata kuliah ini dengan baik dan menarik sehingga mudah dipahami. Terakhir, saya ingin mengucapkan terima kasih kepada para penulis yang tercantum dalam daftar referensi pembuatan makalah ini karena telah memberikan kontribusi penting dalam penyusunan makalah ini.

Referensi

[1] Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2013). "The Keccak reference"

[2] Munir, Rinaldi (2023). SHA-3 (Keccak). Diakses pada 20 Mei 2023, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2022-2023/15-SHA-3-2023

[3] Khankari, N. (2021). Survey on One-Time Password. Diakses pada 20 Mei 2023, dari https://www.researchgate.net/profile/Nilesh-Khankari/publication/349454611_SURVEY_ON_ONE_TIME_PASSWORD/links/603094e4a6fdcc37a83ac4d6/SURVEY-ON-ONE-TIME-PASSWORD.pdf

[4] Phothikitti, Kitti. (2020). Factors Influencing Intentions to Use Cardless Automatic Teller Machine (ATM). *International Journal of Economics and Business Administration*, 8(3), 40 - 56

[5] Sruthi. M, Swapna. M.P. (2019). Secure and Smart Future ATM with One Time Password. International Journal of Engineering Science and Computing, 9(4)

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, 22 Mei 2023

Verawati Esteria S. Simatupang (18220002)