Implementasi *Pluggable Authentication Module* dengan Metode *Challenge Response* ECDSA

Naufal Dean Anugrah - 13518123[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13518123@std.stei.itb.ac.id

***Abstract*—Autentikasi adalah suatu proses untuk memastikan identitas pengguna sebelum mengakses suatu sistem. Di era teknologi ini, autentikasi memegang peranan penting untuk mencegah akses data maupun penggunaan fungsionalitas oleh pihak yang tidak memiliki hak. Terdapat berbagai mekanisme dan pendekatan yang dilakukan untuk melakukan autentikasi, salah satunya adalah menggunakan *Pluggable Authentication Module* (PAM). PAM memungkinkan pengembang untuk mendefinisikan suatu sistem autentikasi yang bisa digunakan di berbagai program. Dalam makalah ini akan dibahas mengenai implementasi sebuah modul PAM pada sistem operasi Ubuntu 18.04. Implementasi modul dilakukan menggunakan bahasa *python* dengan metode autentikasi *challenge response* berbasis *Elliptic Curve Digital Signature Algorithm* (ECDSA). Kunci publik pengguna untuk autentikasi disimpan dalam suatu database Postgresql dan kunci privatnya disimpan dalam suatu penyimpanan eksternal.**

***Keywords*—Autentikasi, *Pluggable Authentication Module*, *Challenge Response*, *Elliptic Curve Digital Signature Algorithm*.**

I. Latar Belakang

Saat ini, teknologi merupakan suatu hal yang tak terpisahkan dari kehidupan manusia. Berbagai program baru yang mendukung aktivitas manusia bermunculan setiap harinya. Program tersebut bisa memiliki basis *mobile*, *desktop*, *website*, dsb.

Tidak sedikit program yang menyimpan data sensitif milik pengguna. Oleh karena itu, perlu dilakukan suatu kontrol terhadap akses suatu data atau fungsionalitas dari suatu program. Hal itu bisa dilakukan dengan melakukan suatu autentikasi untuk memastikan bahwa pengguna memiliki hak untuk mengakses data atau fungsionalitas tersebut.

Pada sistem berbasis Linux, sistem autentikasi bisa didefinisikan dalam suatu modul *Pluggable Authentication Module* (PAM). Metode yang digunakan untuk autentikasi pun bisa beragam. Sebagai contoh, dengan mencocokkan username dan password yang diberikan pengguna dengan nilai yang sudah disimpan pada sistem. Akan tetapi, terdapat risiko pada password pengguna apabila terdapat kesalahan dalam implementasi penyimpanan password, misalnya password disimpan dalam bentuk raw atau password disimpan dalam bentuk hash tanpa diberikan salt sebelumnya.

Untuk mengatasi permasalahan tersebut, terdapat pendekatan untuk menggunakan kriptografi kunci publik untuk melakukan autentikasi. Pada pendekatan ini, kunci privat disimpan oleh pengguna dan sistem hanya menyimpan kunci publik milik pengguna tersebut. Penggunaan kriptografi kunci publik untuk autentikasi ini memiliki dua varian utama. Pertama, dengan melakukan enkripsi dengan kunci privat dan dekripsi sekaligus validasi menggunakan kunci publik. Kedua, dengan menggunakan algoritma tanda tangan digital, misalnya *Elliptic Curve Digital Signature Algorithm* (ECDSA). Pada kedua pendekatan, bisa digunakan metode *challenge response* dengan sistem membuat suatu *challenge* berupa string atau angka secara acak. *Challenge* tersebut selanjutnya dioperasikan menggunakan private key yang disimpan oleh pengguna. Kemudian, hasil dari operasi sebelumnya (bisa berupa nilai enkripsi atau tanda tangan digital) akan divalidasi menggunakan nilai *challenge* awal dan kunci publik pengguna yang tersimpan dalam sistem.

II. Dasar Teori

A. Pluggable Authentication Module

Pluggable Authentication Module (PAM) adalah suatu mekanisme autentikasi terpusat yang dapat digunakan di berbagai program. PAM pertama kali diimplementasikan secara utuh di Linux sekitar tahun 1997 [5]. Sebelum mekanisme PAM diterima secara luas, kebanyakan program memiliki sistem autentikasi masing-masing.

Seperti namanya, kelebihan PAM adalah karena sifatnya yang *pluggable* dan *modular*. Modul PAM baru bisa ditambahkan dengan cukup mudah dan relatif independen dengan modul lain yang sudah ada. Kelebihan lainnya adalah, suatu modul dapat digunakan dalam autentikasi berbagai program yang ada. Akibatnya, pengembang hanya perlu mengimplentasikan sistem autentikasi yang bersifat khusus terhadap program dan sistem autentikasi yang bersifat umum dan sudah menjadi standar tidak perlu diulang.

B. Tanda Tangan Digital

Tanda tangan digital adalah suatu nilai yang berfungsi untuk menjamin tiga aspek keamanan dalam kriptografi, yaitu autentikasi (*authentication*), keaslian pesan (*data integrity*), dan sebagai mekanisme anti penyangkalan (*non-repudiation*). Tanda tangan digital memiliki nilai yang bergantung pada isi pesan dan kunci yang digunakan untuk membangkitkan tanda tangan.

C. Elliptic Curve dan Elliptic Curve Cryptography

Elliptic Curve atau kurva eliptik adalah suatu kurva yang dapat dinyatakan dalam persamaan umum

$$y^2 = x^3 + ax + b \quad \ldots (1)$$

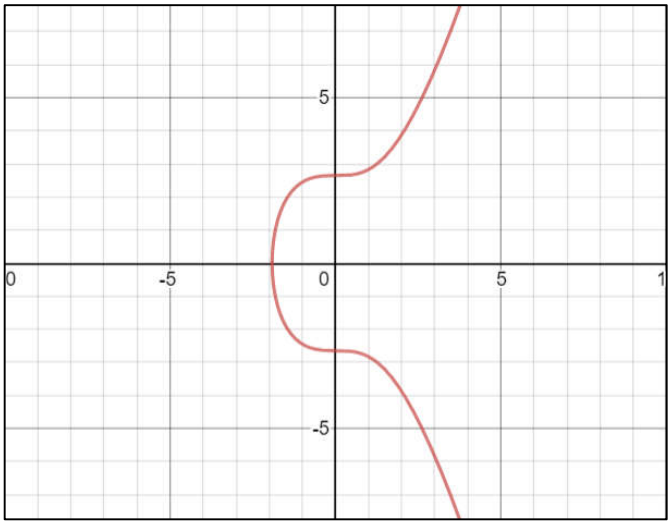
atau dalam suatu medan galois GF(p)

$$y^2 = x^3 + ax + b \bmod p \quad \ldots (2)$$

dengan

$$4a^3 + 27b^2 \neq 0 \quad \ldots (3)$$

Contoh kurva eliptik yang cukup terkenal adalah kurva secp256k1 yang digunakan untuk enkripsi Bitcoin.



Gambar 1 Kurva secp256k1
(dibuat menggunakan https://www.desmos.com/)

Pada dasarnya terdapat dua operasi utama pada kurva eliptik, yaitu penjumlahan dua titik dan perkalian titik dengan suatu nilai skalar. Jika diringkas lagi, perkalian titik juga bisa dipandang sebagai penjumlahan dua titik yang sama sebanyak nilai skalar tertentu. Kemudian, dari operasi dasar tersebut juga didapat operasi pengurangan dan penggandaan titik.

Suatu kurva eliptik pada GF(p) dapat dinyatakan menggunakan beberapa parameter berikut:

1. Nilai a, b, dan p yang merupakan variabel pada persamaan (2).
2. Orde kurva eliptik n.
3. Titik generator G. Nilai n, G, dan titik identitas O memenuhi hubungan $nG = O$.

Pada kurva eliptik didefinisikan dua operasi utama, yaitu:

1. Penjumlahan titik.

Elliptic Curve Cryptography (ECC) adalah suatu pendekatan kriptografi kunci publik yang dilakukan dalam suatu kurva eliptik. ECC memanfaatkan *Elliptic Curve Discrete Logarithm Problem* (ECDLP) pada kurva eliptik. ECDLP menyatakan bahwa proses menghitung $Q = kP$ adalah mudah, tetapi menghitung nilai k dengan diberikan titik Q dan P merupakan proses yang sulit.

D. Elliptic Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm (ECDSA) adalah implementasi algoritma *Digital Signature Algorithm* (DSA) dalam yang memanfaatkan ECC. Secara umum, ECDSA memiliki dua bagian utama, yaitu proses pembangkitan dan verifikasi signature.

Terdapat beberapa parameter yang digunakan dalam algoritma ECDSA, yaitu:

1. Kurva eliptik E dengan parameter (a,b,p,n,G) seperti telah dijelaskan sebelumnya.
2. Kunci privat d berupa integer pada rentang $1 \leq d \leq n - 1$.
3. Kunci public Q berupa titik pada kurva E. Nilai Q memenuhi hubungan $Q = dG$.

Tahapan untuk melakukan pembangkitan signature adalah sebagai berikut:

1. Hitung nilai hash h dari pesan yang ingin dibangkitkan signaturenya. Fungsi hash yang digunakan bebas asalkan aman secara kriptografi, misal SHA-256.
2. Hitung nilai z yaitu x *most significant bit* dari h dengan x adalah panjang bit dari n.
3. Ambil suatu angka k dari rentang $1 \leq k \leq n - 1$.
4. Hitung $kG = (x_1, y_1)$.
5. Hitung $r = x_1 \bmod n$. Jika $r \neq 0$ lanjut ke langkah berikutnya. Jika tidak, kembali ke langkah nomor 3.
6. Hitung $s = k^{-1}(z + dr) \bmod n$. Jika $s \neq 0$ lanjut ke langkah berikutnya. Jika tidak, kembali ke langkah nomor 3.
7. Didapat signature dari pesan masukan adalah pasangan nilai (r, s).

Tahapan untuk melakukan verifikasi signature adalah sebagai berikut:

1. Cek apakah $1 \leq r, s \leq n - 1$. Jika terpenuhi, lanjut ke langkah berikutnya. Jika tidak, signature tidak valid.
2. Hitung nilai z dengan metode yang sama dengan langkah 1 hingga 2 pada proses pembangkitan signature.
3. Hitung nilai $s_{inv} = s^{-1} \bmod n$.
4. Hitung nilai $u_1 = zs_{inv} \bmod n$ dan $u_2 = rs_{inv} \bmod n$.
5. Hitung nilai $X = u_1 G + u_2 Q$. Jika X adalah titik identitas O, signature tidak valid. Jika bukan, lanjut ke langkah berikutnya.
6. Misal (x_1, x_2) adalah koordinat titik X. Signature valid jika dan hanya jika $r = x_1 \ (mod\ n)$.

E. Challenge Response Authentication

Challenge response authentication adalah suatu metode di mana satu pihak memberikan suatu pertanyaan atau *challenge* dan pihak lain harus menjawab untuk bisa terautentikasi. Contoh implementasi paling sederhana dari metode ini adalah autentikasi berbasis password, di mana satu pihak menanyakan password dan pihak lain memberikan password tersebut.

Metode ini bisa dikombinasikan dengan penggunaan kriptografi kunci publik, misal dengan menggunakan enkripsi-dekripsi biasa atau tanda tangan digital. Pada kasus ini, sistem membuat suatu *challenge* berupa string atau angka secara acak. *Challenge* tersebut selanjutnya dienkripsi atau ditandatangani menggunakan kunci private milik pengguna. Kemudian, hasilnya akan didekripsi atau divalidasi menggunakan kunci publik pengguna yang disimpan di sistem. Jika proses dekripsi menghasilkan nilai *challenge* atau tanda tangan valid, pengguna akan terautentikasi.

III. RANCANGAN SOLUSI

A. Rancangan Umum

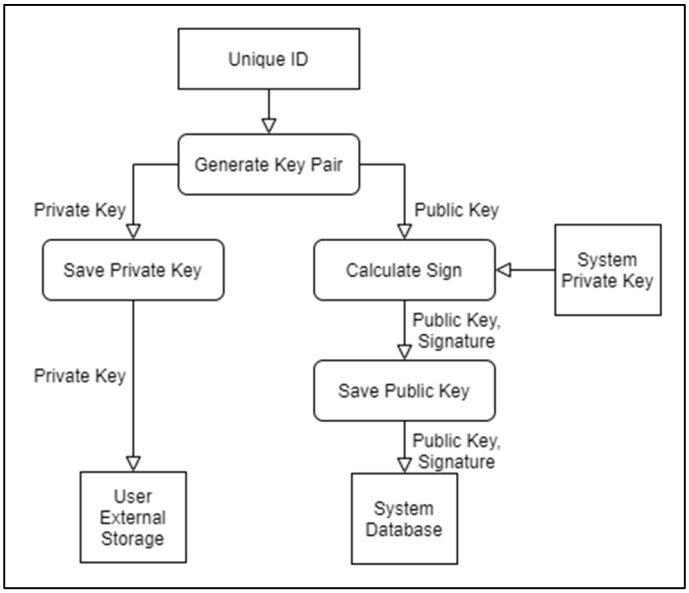
Implementasi tersusun atas proses registrasi dan autentikasi. Detail skema dari kedua proses dapat dilihat pada poin B dan C bab ini. Untuk menjalankan proses tersebut, digunakan satu pasang kunci publik-privat milik sistem dan satu pasang atau lebih kunci publik-privat milik satu pengguna atau lebih.

Kunci milik sistem digunakan untuk membangkitkan dan memvalidasi signature dari publik key yang disimpan di database. Hal ini bertujuan agar pengguna tanpa akses ke private key sistem tidak bisa membangkitkan sembarang kunci public yang bisa digunakan untuk mengakses ID lain. Kunci ini, baik publik maupun privat disimpan pada sistem dengan akses terhadap kunci privat dibatasi ke *user root*.

Di sisi lain, kunci milik pengguna digunakan untuk membangkitkan dan memvalidasi signature *challenge* yang diberikan oleh sistem. Kunci privat pada pasangan ini disimpan oleh pengguna dalam suatu media penyimpanan eksternal. Sedangkan kunci publik bisa diturunkan dari kunci privat kemudian disimpan ke sistem.

Sebagai catatan, pada makalah ini implementasi modul PAM akan diaplikasikan ke program sudo untuk mempermudah *testing*. Akan tetapi, secara umum hasil implementasi modul PAM tersebut dapat diaplikasikan ke berbagai program lainnya.

B. Skema Register



Gambar 2 Skema Register

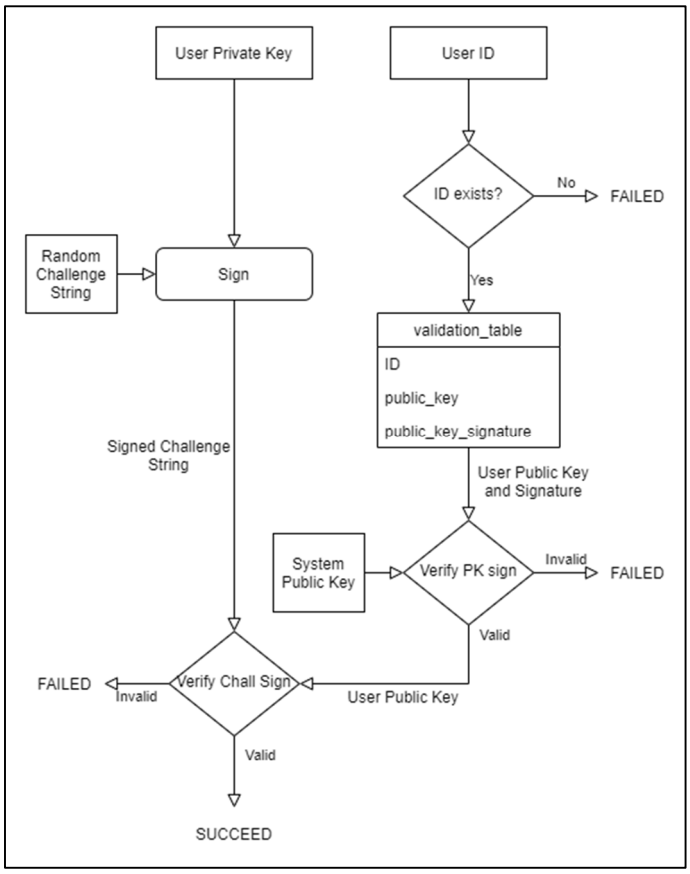
Proses register pengguna bisa dibilang cukup sederhana. Berikut adalah proses yang dilakukan (lihat Gambar 2);

1. Bangkitkan pasangan kunci yang baru dengan ID yang belum dipakai. Misalnya dengan membangkitkan angka acak dan melakukan cek ke database atau dengan melakukan increment sederhana.
2. Simpan kunci privat ke penyimpanan eksternal milik pengguna.
3. Hitung signature dari kunci publik menggunakan kunci privat sistem.
4. Simpan kunci publik dan signature ke database sistem.

C. Skema Autentikasi

Pada saat seorang pengguna melakukan proses autentikasi, akan dicari kunci privat pada penyimpanan eksternal (misal *flash disk* atau *hard disk*) yang sedang terpasang pada sistem. Kemungkinan keluarannya adalah sebagai berikut:

1. Jika terdapat kunci privat yang sesuai dari semua kunci privat yang ditemukan, pengguna akan terautentikasi.
2. Jika tidak ditemukan kunci privat yang sesuai, pengguna akan diarahkan ke mekanisme autentikasi lainnya jika tersedia (misal menggunakan password).
3. Jika tidak ada mekanisme autentikasi lain, proses autentikasi gagal.



Gambar 3 Skema Autentikasi

Detail proses validasi kunci privat tersebut adalah sebagai berikut (lihat Gambar 3):

1. Sistem apakah di database terdapat kunci publik dengan ID yang sama dengan kunci privat yang dimasukkan.
   Jika ada, lanjut ke proses selanjutnya. Jika tidak, validasi gagal.
2. Sistem akan mengambil kunci publik dan *signature* (tanda tangan) dari kunci publik tersebut. *Signature* tersebut akan divalidasi dengan menggunakan kunci publik sistem.
   Jika signature valid dan ID yang tersimpan di kunci publik sama dengan ID yang digunakan pada proses sebelumnya, lanjut ke proses selanjutnya. Jika tidak, validasi gagal.
3. Sistem akan membangkitkan suatu *challenge* berupa string sepanjang 64 karakter.

4. Tanda tangan digital dari string *challenge* akan dibangkitkan menggunakan kunci privat tersebut sesuai implementasi ECDSA.
5. Tanda tangan yang dihasilkan akan diverifikasi menggunakan kunci publik user yang telah didapatkan pada proses nomor 2.
   Jika tanda tangan tersebut valid, pengguna berhasil terautentikasi. Jika tidak, validasi gagal, lanjut ke kunci privat selanjutnya jika ada.

IV. Implementasi

A. Lingkungan Implementasi

Lingkungan (*environment*) yang digunakan untuk implementasi modul PAM ini adalah sebagai berikut:

1. Sistem Operasi Ubuntu 18.04 yang dijalankan di Virtualbox 6.1 dengan RAM 2048MB dan CPU 2 core.
2. Bahasa pemrograman yang digunakan adalah Python versi 2.7.17.
3. Database yang digunakan adalah Postgresql versi 13.1. User yang digunakan untuk implementasi adalah **postgres** dengan password **postgres**. Data ID, kunci publik, dan signature kunci publik pengguna akan disimpan dalam tabel **validation_table** dalam database bernama **pam**.

B. Persiapan Lingkungan Implementasi

Proses persiapan ini berasumsi bahwa sistem operasi yang digunakan merupakan hasil *fresh installation*. Berikut adalah tahap-tahap yang perlu dilakukan:

1. Install *dependency* untuk mendukung implementasi modul PAM dalam bahasa Python2. Berikut adalah command yang digunakan:

```
sudo apt-get update
sudo apt-get install python
sudo apt-get install libpam-python
```

2. Install database Postgresql. Script installasi dapat diakses pada laman https://www.postgresql.org/download/linux/ubuntu/ dan tidak akan ditampilkan dalam makalah ini.

3. Izinkan login Postgresql menggunakan password.
   a. Pada file **/etc/postgresql/13/main/pg_hba.conf**:
      - Ubah baris:
        ```
        local all postgres peer
        ```
        menjadi:
        ```
        local all postgres md5
        ```
      - Ubah baris:
        ```
        local all postgres peer
        ```
        menjadi:
        ```
        local all postgres md5
        ```
   b. Login ke user postgres dan lakukan set password dengan command berikut (masukkan password dan verifikasi password setelah muncul prompt input):
      ```
      sudo -u postgres psql postgres
      \password postgres
      ```
      Kemudian keluar dari Postgresql dengan command **\q**.
   c. Restart Postgresql dengan command:
      ```
      sudo service postgresql restart
      ```

4. Login kembali ke Postgresql dan buat database **pam** dan tabel **validation_table**. Untuk mempermudah, penjelasan command akan digabungkan dengan command dengan ditandai simbol # sebelum penjelasan.

```
# command login, input password saat diminta
psql -U postgres -W
# query sql untuk membuat database
CREATE DATABASE pam;
# command untuk connect database, input
# password saat diminta
\c pam
# query sql untuk membuat tabel
CREATE TABLE validation_table (
    id int PRIMARY KEY,
    public_key varchar(500),
    signature varchar(300)
);
```

5. Install dependency Python (psycopg2 untuk membuat koneksi dengan database) secara global. Berikut adalah command yang digunakan:

```
sudo apt install python-pip
sudo pip install psycopg2-binary
```

6. Pada file **/etc/pam.d/sudo**, tambahkan line berikut (lihat Gambar 4 sebagai contoh):

```
auth sufficient pam_python.so [<pam_py_path>]
```

```
nightmare@nightmare-VirtualBox:~$ cat /etc/pam.d/sudo
#%PAM-1.0

auth       sufficient pam_python.so [/media/sf_Kriptografi-Makalah-2/pam.py]
session    required   pam_env.so readenv=1 user_readenv=0
session    required   pam_env.so readenv=1 envfile=/etc/default/locale user_read
env=0
@include common-auth
@include common-account
@include common-session-noninteractive
```

Gambar 4 Konten File Konfigurasi PAM untuk sudo

C. Kode Program

Berikut adalah kode program yang digunakan untuk mengimplementasikan PAM sesuai rancangan yang telah dituliskan sebelumnya. Sebagai catatan, pada makalah ini hanya ditampilkan bagian yang penting, yaitu fungsi get_matched_public_key dan pam_sm_authenticate. Kemudian, fungsi get_private_keys_from_usb (tidak ditampilkan implementasinya di makalah) berfungsi untuk mengembalikan list path kunci privat dari semua USB yang terpasang. Kurva yang digunakan adalah secp256k1 [6]. Kemudian, kode template pam.py didapat dari dokumentasi pam python [7]. Karena fokus pada makalah ini adalah autentikasi, fungsi yang perlu diimplementasikan adalah pam_sm_authenticate.

Implementasi secara lengkap dapat dilihat pada repositori Github yang tautannya dituliskan pada Bab VIII.

pam.py

```
...
def get_private_keys_from_usb():
    ...


def get_matched_public_key(id):
```

```
    cursor = db.cursor()
    cursor.execute('SELECT public_key, signature FROM
                validation_table WHERE id=%s', (id,))
    res = cursor.fetchone()
    if res is not None:
        try:
            user_public_key, signature = res
            r, s = list(map(int,
                                signature.split(',')))
            # load system private key
            curve = SECP256K1
                    .load_key(SYSTEM_PUBLIC_KEY_PATH)
            ecdsa = ECDSA(curve)
            # validate signature
            hs = hashlib.sha256(user_public_key)
            pubkey_hash = int(hs.hexdigest(), 16)
            ecdsa.verify(pubkey_hash, r, s)
        except Exception as e:
            raise NoMatchedPubKeyError()
        else:
            return user_public_key
    else:
        raise NoMatchedPubKeyError()

def pam_sm_authenticate(pamh, flags, argv):
    # Setup user
    try:
        print('[+] Using PAM to authenticate...')
        user = pamh.get_user(None)
    except pamh.exception as e:
        print('[+] Error happened. Exiting...')
        return e.pam_result
    if user == None:
        pamh.user = DEFAULT_USER

    try:
        # Sign using all available private key in usb
        for key_path in get_private_keys_from_usb():
            try:
                print('[+] Trying key {0}:'
                                .format(key_path)),

                # load curve from key
                curve = SECP256K1.load_key(key_path)
                ecdsa = ECDSA(curve)

                # Get matched public key
                pubkey = get_matched_public_key(
                                            curve.id)

                # Sign some random string
                chall = os.urandom(64)
                hs = hashlib.sha256(chall)
                chall_hash = int(hs.hexdigest(), 16)
                r, s = ecdsa.sign(chall_hash)

                # Verify signature
                ecdsa.curve = SECP256K1
                                .parse_repr(pubkey)
                ecdsa.verify(chall_hash, r, s)
            except LoadKeyError as e:
                print('invalid key file')
            except NoMatchedPubKeyError as e:
                print('no matched pubkey')
            except ValidationError as e:
                print('validation error')
            except Exception as e:
                print(e)
            else:
                print('succeed')
                print('[+] Authenticated using
                                                PAM...')
                return pamh.PAM_SUCCESS
        except Exception as e:
            print(e)
            print('[+] PAM authentication error...')
            return pamh.PAM_AUTH_ERR

        # Matching private key not found in any usb
        print('[+] Matching private key not found...')
        return pamh.PAM_AUTH_ERR
...
```

Selain itu, juga terdapat beberapa kode lain yang tidak ditampilkan pada makalah ini, seperti kode modul ecdsa, kode utilitas untuk setup pasangan kunci sistem (file gen_system_key.py), dan kode utilitas untuk register USB (file register.py). Semua kode tersebut dapat dilihat pada repositori Github yang tautannya dituliskan pada Bab VIII.

V. PENGUJIAN

Pertama, akan dilakukan register terhadap tiga kunci pada USB, kunci tersebut bernama key1.pri, key2.pri, dan key3.pri. Pada file key1.pri, nilai ID akan diubah 1 digit. Pada file key2.pri, nilai private key *d* akan diubah 1 digit. File key3.pri dibiarkan seperti sedia kala. Terakhir, ditambahkan 1 file kosong bernama key0.pri.

Berikut adalah isi dari keempat file kunci privat tersebut:

key0.pri (kosong)
key1.pri (ID diubah dari 396761430 menjadi 396761439)
-----BEGIN SECP256K1 PRIVATE KEY----- 396761430 63836371499806850520937658570984687413386265269004864122274608567628922611171 -----END SECP256K1 PRIVATE KEY-----
key2.pri (bagian akhir private key d diubah dari ...29940 menjadi 29941)
-----BEGIN SECP256K1 PRIVATE KEY----- 427339556 53853157560882139337824630867227810404769283737140156345692544461830054929941 -----END SECP256K1 PRIVATE KEY-----

```
key3.pri
(tidak diubah)
-----BEGIN SECP256K1 PRIVATE KEY-----
934531436
480316467932130318054863607084406083980595551195
864647222861644726203658403 83
-----END SECP256K1 PRIVATE KEY-----
```

Berikut adalah kunci publik yang tersimpan pada tabel validation_table (atribut signature tidak ditampilkan):

```
pam=# select id, public_key from validation_table;
    id     |                                                                  public_key
-----------+----------------------------------------------------------------------------------------------
 396761439 | -----BEGIN SECP256K1 PUBLIC KEY-----                                                        +
           | 396761439                                                                                    +
           | 13052951226142071483538676332853748614729957104033665542985892588923083333687 +
           | 49593519967599369838699533637351662303155644818331987261717789106318060400515 +
           | -----END SECP256K1 PUBLIC KEY-----                                                          +
           |
 427339556 | -----BEGIN SECP256K1 PUBLIC KEY-----                                                        +
           | 427339556                                                                                    +
           | 10304787173955401973060254102965370544166884254686819589620957065324260192332 2+
           | 96811812272068993639714455704300549703177048562001597270913386828178209957850 +
           | -----END SECP256K1 PUBLIC KEY-----                                                          +
           |
 934531436 | -----BEGIN SECP256K1 PUBLIC KEY-----                                                        +
           | 934531436                                                                                    +
           | 44570915476442095972138359788448699876645676130321160223499701205377832286563 +
           | 22589786638684220719747594095161408872530317113884042136776539377907123425917 +
           | -----END SECP256K1 PUBLIC KEY-----                                                          +
           |
(3 rows)
```

Gambar 5 Isi Tabel validation_table

Kemudian, saat dicoba menjalankan command "**sudo -i**" didapat keluaran berikut:

```
nightmare@nightmare-VirtualBox:~$ sudo -i
[+] Using PAM to authenticate...
[+] Trying key /media/nightmare/NIGHTMARE/keys/key0.pri: invalid key file
[+] Trying key /media/nightmare/NIGHTMARE/keys/key1.pri: no matched pubkey
[+] Trying key /media/nightmare/NIGHTMARE/keys/key2.pri: validation error
[+] Trying key /media/nightmare/NIGHTMARE/keys/key3.pri: succeed
[+] Authenticated using PAM...
root@nightmare-VirtualBox:~#
```

Gambar 6 Keluaran Command "sudo -i" dengan USB

Dapat dilihat pada Gambar 6 bahwa key0 gagal karena file kunci privat tidak valid, key1 gagal karena ID sudah diubah, key2 gagal karena private key d diubah, dan key3 valid. Setelah mencoba key3, pengguna akan terautentikasi sebagai user root.

Kemudian, sebagai tambahan dicoba untuk menjalankan command yang sama pada sesi terminal yang baru. Akan tetapi, pada kasus ini tanpa USB. Command tersebut menghasilkan keluaran berikut:

```
nightmare@nightmare-VirtualBox:~$ sudo -i
[+] Using PAM to authenticate...
[+] Matching private key not found...
[sudo] password for nightmare:
```

Gambar 7 Keluaran Command "sudo -i" tanpa USB

Dapat dilihat pada Gambar 7, tidak ditemukan file kunci privat sehingga autentikasi langsung gagal. Kemudian, autentikasi dilanjutkan ke mekanisme default dari sudo, yaitu menggunakan password user root.

Dari pengujian yang telah dilakukan, dapat disimpulkan bahwa modul PAM sudah berjalan dengan baik. Kunci privat pada penyimpanan eksternal berhasil dideteksi. Kemudian, proses validasi kunci privat juga sudah menghasilkan keluaran sebagaimana mestinya.

VI. Analisis Keamanan

Metode autentikasi menggunakan *Elliptic Curve Digital Signature Algorithm* yang telah diimplementasikan memiliki tingkat keamanan yang lebih tinggi dibanding metode lainnya, misalnya autentikasi menggunakan username dan password. Alasannya adalah sebagai berikut:

1. Sistem hanya menyimpan kunci public sehingga dapat terhindar dari risiko pembobolan database atau akses *illegal* lain. Hal ini akan tetap berlaku bahkan jika akses database dibiarkan terbuka.
2. *Challenge* string memiliki panjang 64 byte sehingga untuk saat ini tidak mungkin diserang secara *bruteforce*.
3. Bahkan jika semisal serangan *bruteforce* berhasil, penyerang tersebut masih tidak memiliki kunci privat yang digunakan untuk membangkitkan tanda tangan digital untuk *challenge* tersebut.
4. Kunci publik yang disimpan pada database Postgresql juga dilengkapi dengan ID serta signature. Signature tersebut dibangkitkan menggunakan kunci privat milik sistem dan hanya bisa diakses oleh *root user*. Apabila program yang diautentikasi bukanlah sudo, pengguna tidak akan bisa membangkitkan kunci publik baru dengan signature yang valid untuk bisa digunakan oleh ID lain. Sebagai tambahan, ID pengguna yang terautentikasi bisa dicatat dalam suatu audit log sehingga aktivitas pengguna tersebut bisa direkam. Apabila terdapat aktivitas yang mencurigakan, pengguna tersebut dapat langsung ditindaklanjuti.

VII. Kesimpulan dan Saran Pengembangan

Pluggable Authentication Module (PAM) adalah suatu mekanisme yang dapat digunakan untuk mendefinisikan suatu metode autentikasi yang bersifat terpusat. Metode autentikasi yang digunakan dapat beragam, salah satunya dengan memanfaatkan algoritma tanda tangan digital seperti *Elliptic Curve Digital Signature Algorithm* (ECDSA). Dengan implementasi ini, sistem hanya menyimpan kunci publik milik pengguna. Akibatnya, risiko yang mungkin terjadi akibat pembobolan database sistem atau penyalahgunaan akses oleh pihak internal dari sistem dapat dikurangi. Mengingat autentikasi hanya bisa didapatkan menggunakan kunci privat yang disimpan oleh pengguna.

Modul PAM dapat dimanfaatkan dalam berbagai program selain sudo yang sudah diimplementasikan pada makalah ini. Selain itu, media autentikasi yang dipakai juga bisa dikembangkan lebih jauh, sebagai contoh dengan menggunakan *smartcard*, *smartphone*, atau bahkan sistem lain misalnya untuk proses autentikasi pemakaian *Application Programming Interface*.

VIII. Tautan Github

Berikut adalah tautan repositori Github yang berisi kode yang digunakan dalam penyusunan makalah ini:

https://github.com/naufal-dean/Kriptografi-Makalah-2

IX. Ucapan Terima Kasih

Penulis ingin menyampaikan ucapan terima kasih kepada berbagai pihak yang telah membantu dalam proses penyelesaian makalah ini. Pertama, kepada Tuhan Yang Maha Esa yang senantiasa melimpahkan rahmat dan berkah-Nya sehingga penulis dapat menyelesaikan makalah ini dengan baik. Kedua, kepada Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen mata kuliah IF4020 Kriptografi yang sudah membimbing penulis untuk dapat memahami materi yang digunakan pada makalah ini. Ketiga, kepada penulis sumber referensi yang digunakan pada makalah ini. Keempat, kepada orang tua penulis yang selalu mendukung dan memberikan semangat untuk selalu belajar. Kelima, teman-teman anggota kelas Kriptografi yang selalu memberikan semangat untuk dapat bersaing secara sehat selama menjalani kuliah.

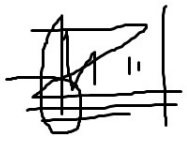
Referensi

[1] Johnson, Don et all. “The Elliptic Curve Digital Signature Algorithm (ECDSA)”. Waterloo: Certicom. 2001.
[2] Harn, L. et all. “Password Authentication Using Public-Key Cryptography”. 1988.
[3] Munir, Rinaldi. “Elliptic Curve Cryptography (ECC)”. Bandung: Institut Teknologi Bandung. 2020.
[4] Munir, Rinaldi. “Tanda-tangan Digital”. Bandung: Institut Teknologi Bandung. 2020.
[5] Lauber, Susan. “An introduction to Pluggable Authentication Modules (PAM) in Linux”. https://www.redhat.com/sysadmin/pluggable-authentication-modules-pam. 2020.
[6] Anonim. “Secp256k1”. https://en.bitcoin.it/wiki/Secp256k1. 2019.
[7] Stuart, Russel. “pam_python documentation”. http://pam-python.sourceforge.net/doc/html/. 2020.

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.

Sleman, 21 Desember 2020



Naufal Dean Anugrah
13518123