Implementasi *Digital Signature* untuk Verifikasi Resource Pack pada *Minecraft: Java Edition*

Farhan Nabil Suryono - 13521114
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521114@std.stei.itb.ac.id

***Abstract*—Minecraft merupakan sebuah game yang senantiasa populer bertahun-tahun lamanya. Game ini memudahkan penggunanya untuk mengganti tekstur yang digunakan pada game melalui sebuah fitur bernama *resource pack*. *Resource pack* umumnya didistribusikan secara bebas oleh pembuatnya, namun hal ini menyebabkan beberapa celah untuk melakukan kriminalitas seperti distributor palsu atau pencurian. Makalah ini mempresentasikan pemanfaatan tanda tangan digital sebagai solusi dari masalah ini.**

Keywords—Digital Signature, Minecraft, Resource Pack

I. Pendahuluan

Minecraft adalah sebuah *game* yang sangat terkenal di dunia. *Game* yang dirilis pada tahun 2009 ini masih banyak dimainkan di zaman sekarang. Hal ini dibuktikan hasil analisis David Curry pada artikelnya yang berjudul "Minecraft Revenue and Usage Statistics (2024)" yang menyebutkan bahwa 90 juta orang di dunia memainkan *game* ini setidaknya sebulan sekali pada tahun 2022.

Minecraft merupakan *game* yang sangat terbuka terhadap tambahan-tambahan eksternal. Hal ini ditunjukkan dengan banyaknya *mod* untuk Minecraft. Selain itu, Minecraft juga menyediakan fitur bagi pengguna yang merasa tekstur original Minecraft kurang menarik. Fitur ini bernama *resource pack*. Pengguna dapat menggunakan *resource pack* yang telah dikustomisasi agar pengguna dapat menggunakan tekstur yang menurut mereka lebih menarik.

Banyaknya pengguna yang lebih memilih *custom resource pack* menyebabkan banyak orang berusaha membuat resource pack. Pada umumnya, pembuat *resource pack* ini membebaskan penggunanya untuk men-*download* dan mendistribusikan *resource pack* karya mereka secara bebas karena mereka ingin membagikan karya mereka kepada komunitas Minecraft yang lebih luas dan mendapatkan popularitas serta apresiasi dari sesama pemain. Selain itu, kebebasan ini juga mendorong kreativitas dan inovasi di kalangan pemain.

Namun, kebebasan dalam pendistribusian *resource pack* ini juga menyebabkan beberapa masalah. Pada tahun 2021, salah satu *texture pack* terkenal pada komunitas Minecraft bernama *Faithful* yang dijual dalam toko *official* Minecraft ternyata bukan di-*publish* oleh pembuat aslinya. Selain itu, seorang YouTuber sekaligus pembuat *resource pack* bernama HellCastle & Tylerrr mengatakan bahwa banyak beredar grup dalam aplikasi *discord* yang mengaku mendistribusikan *resource pack* padahal mereka memiliki tujuan mengumpulkan data pribadi orang yang mereka tipu.

Oleh karena itu, penulis dalam makalah ini menawarkan sebuah solusi terhadap masalah-masalah ini yaitu mengaplikasikan tanda tangan digital pada *resource pack* yang didistribusikan untuk memastikan integritas serta autentikasi *resource pack*.

II. Dasar Teori

A. Kriptografi

Berdasarkan definisi dari Schneier (1996), kriptografi adalah sebuah ilmu atau seni untuk menjaga keamanan suatu pesan. Kriptografi menjaga suatu pesan menggunakan teknik-teknik matematika. Pada kriptografi, definisi aman adalah ketika 4 konsep ini terpenuhi, yaitu:

1. Kerahasiaan (*confidentiality*) yang bermakna sebuah pesan tidak dapat dibaca oleh pihak-pihak yang tidak memiliki hak untuk membacanya.
2. Integritas Data (*data integrity*) yang bermakna pesan yang diterima merupakan pesan yang bersifat asli dan utuh tanpa adanya perubahan oleh pihak lain selama proses pengiriman pesan.
3. Autentikasi (*authentication*) yang bermakna pesan dikirim oleh pengirim aslinya dan bukan dari pihak lain yang berusaha menyerupai pengirim asli.
4. Nirsangkal (*non-repudiation*) yang bermakna pengirim pesan tidak dapat menyangkal bahwa dialah pengirim pesan tersebut.

Ada berbagai terminologi yang penting dalam kriptografi antara lain sebagai berikut.

1. Pesan

   Data atau informasi yang dapat dibaca dan dimengerti maknanya oleh pihak-pihak yang dapat memahaminya.

2. Pengirim dan Penerima

   Pengirim adalah pihak yang melakukan pengiriman pesan. Sedangkan, penerima adalah pihak yang menerima pesan dari pihak pengirim.

3. Penyusup

Penyusup atau *intruder* adalah pihak ketiga diluar pengirim dan penerima yang melakukan penyadapan, intersepsi, penghapusan, penambahan, atau pengubahan terhadap suatu pesan.

4. Plainteks

   Plainteks atau *plaintext* adalah pesan asli yang dikirim oleh pengirim untuk penerima.

5. Cipherteks

   Cipherteks atau *ciphertext* adalah pesan yang telah dimodifikasi sedemikian rupa sehingga tidak dapat dibaca secara langsung dan dibutuhkan sesuatu informasi khusus yang umumnya berupa kunci untuk membacanya.

6. Enkripsi

   Enkripsi adalah proses mengubah sebuah plainteks menjadi sebuah cipherteks.

7. Dekripsi

   Dekripsi adalah proses pengembalian sebuah cipherteks menjadi plainteks.

Pada sistem kriptografi, kunci merupakan hal terpenting agar memastikan keamanan suatu pesan antara pihak pengirim dan penerima. Berdasarkan mekanismenya, kriptografi terbagi menjadi 3 antara lain sebagai berikut.

1. Algoritma kunci simetri (*symmetric-key cryptography*)

   Algoritma kriptografi yang menggunakan kunci sama untuk proses enkripsi dan dekripsi.

2. Algoritma kunci nir-simetri (*asymmetric-key cryptography*)

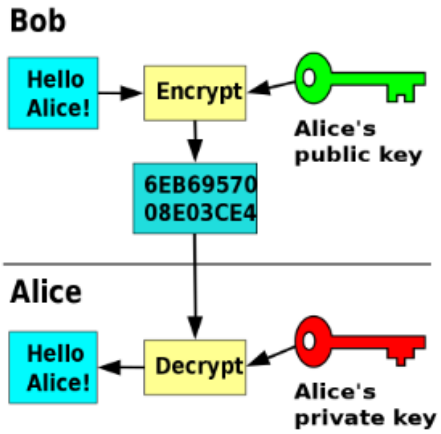
   Algoritma kriptografi yang menggunakan kunci berbeda untuk proses enkripsi dan dekripsi. Algoritma ini juga dikenal sebagai kriptografi kunci publik.

3. Fungsi hash

   Algoritma kriptografi yang menghasilkan suatu pesan yang tidak dapat dikembalikan (*irreversible*) dan umumnya berukuran *fixed*.

B. Kriptografi Kunci Publik

Kriptografi kunci publik merupakan salah satu teknik dalam kriptografi yang menggunakan dua buah kunci yang berbeda untuk melakukan proses enkripsi dan dekripsi. Dua kunci yang digunakan dalam teknik ini sering disebut sebagai kunci publik dan kunci privat. Kunci publik merupakan kunci yang bersifat tidak rahasia dan umumnya dapat diakses publik dalam sebuah *repository*. Sedangkan kunci privat merupakan kunci yang bersifat rahasia dan hanya pemilik kunci yang mengetahui kuncinya sendiri. Teknik kriptografi diperkenalkan oleh Whitfield Diffie dan Martin E. Hellman pada tahun 1976 dan teknik ini telah berkembang pesat sehingga menjadi basis untuk berbagai protokol keamanan di masa modern.



Gambar 2.1 Ilustrasi proses enkripsi dan dekripsi pada kriptografi kunci publik

Pada dasarnya, algoritma ini umumnya memanfaatkan persoalan matematis yang sulit dipecahkan oleh komputer dalam waktu cukup singkat seperti pemfaktoran bilangan besar dan permasalah logaritma diskrit. Oleh karena itu, kunci publik dapat dipublikasikan secara bebas tanpa mengorbankan keamanan. Beberapa contoh algoritma kriptografi yang menggunakan teknik ini adalah Diffie-Hellman, RSA (Rivest-Shamir-Adleman), ElGamal, dan *Elliptic Curve Cryptography* (ECC).

C. Tanda Tangan Digital

Tanda tangan merupakan sebuah bukti yang digunakan untuk mengautentikasi sebuah dokumen cetak sejak dulu. Tanda tangan memiliki karakteristik berupa bukti yang autentik, tidak dapat dilupakan, tidak dapat dipindah dengan tujuan digunakan ulang pada dokumen lain, dokumen yang ditandatangani tidak dapat diubah, dan tidak dapat disangkal oleh penulisnya.

Tanda tangan digital adalah tanda tangan untuk sebuah data digital. Tanda tangan digital berbeda dengan tanda tangan biasa yang lalu dipindai atau difoto. Tanda tangan digital dapat digunakan untuk mengamankan informasi, memastikan keaslian data, dan autentikasi dokumen elektronik. Tanda tangan digital memiliki sifat berbeda dengan tanda tangan biasa antara lain tanda tangan digital selalu berbeda untuk dokumen yang berbeda meskipun pembuatnya sama atau pembuatannya menggunakan kunci yang sama.

Ada beberapa syarat dalam sebuah tanda tangan digital yaitu:

1. Tanda tangan digital harus berupa rangkaian bit yang bergantung terhadap data yang ditandatangani.
2. Tanda tangan digital harus menggunakan sebuah informasi unik dari pengirim umumnya berupa kunci untuk mencegah adanya pemalsuan dan penyangkalan.
3. Membangkitkan sebuah tanda tangan digital baru harus relatif mudah dilakukan.
4. Mengenali dan memverifikasi sebuah tanda tangan digital harus relatif mudah dilakukan.

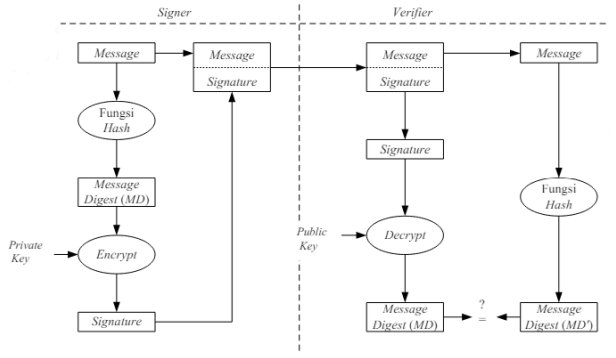
5. Tanda tangan digital harus hampir tidak mungkin dipalsukan menggunakan komputer baik dengan menggunakan tanda tangan yang sama untuk pesan berbeda atau membuat tanda tangan curang untuk sebuah pesan.
6. Salinan tanda tangan digital harus mudah disimpan ke dalam *storage*.

Ada 2 proses dalam melakukan tanda tangan digital antara lain:

1. Penandatanganan (*signing*) yaitu proses memberikan tanda tangan pada suatu data digital.
2. Verifikasi (*verification*) yaitu proses memeriksa keabsahan tanda tangan digital terhadap suatu data digital.

Tanda tangan digital umumnya menggunakan kombinasi dari kriptografi kunci publik dan fungsi *hash*. Contoh algoritma yang umum digunakan untuk tanda tangan digital adalah RSA dan ElGamal. Berikut adalah langkah-langkah umum penandatanganan serta verifikasi sebuah tanda tangan digital yang menggunakan kombinasi dari kriptografi kunci publik dan fungsi *hash*.

1. Pengirim menghitung nilai dari hasil fungsi *hash* terhadap pesan yang dikirim. Hasil *hash* ini dinamakan *message digest* (MD).
2. *Message digest* kemudian dienkripsi menggunakan kunci privat pengirim dan menghasilkan tanda tangan digital dari pesan yang dikirim.
3. Pengirim mengirim pesannya dengan menyertakan tanda tangan digitalnya pada penerima.
4. Penerima menerima pesan serta tanda tangan digital dari pengirim.
5. Penerima melakukan dekripsi terhadap tanda tangan digital menggunakan kunci publik pengirim sehingga menghasilkan sebuah *message digest* 1.
6. Penerima juga melakukan *hash* terhadap pesan yang diterima (tanpa tanda tangan) sehingga menghasilkan *message digest* 2.
7. Penerima melakukan verifikasi dengan membandingkan nilai dari *message digest* 1 dan *message digest* 2. Tanda tangan digital akan berhasil diverifikasi atau valid ketika *message digest* 1 bernilai sama dengan *message digest* 2.



Gambar 2.2 Alur penandatanganan digital menggunakan kombinasi kriptografi kunci publik dan fungsi *hash*

D. Elliptic Curve Digital Signature Algorithm (ECDSA)

Elliptic Curve Digital Signature Algorithm atau ECDSA merupakan implementasi dari DSA yang menggunakan kurva eliptik sebagai kuncinya. DSA atau Digital Signature Algorithm merupakan sebuah algoritma kriptografi kunci publik yang dikhususkan tanda tangan digital dan tidak dapat digunakan untuk enkripsi pesan. DSA dikembangkan dari algoritma ElGamal dan menggunakan SHA-1 untuk menghasilkan *message digest* yang ditetapkan berukuran 160 bit. Kelebihan ECDSA dibandingkan DSA adalah kunci yang digunakan lebih pendek untuk mencapai tingkat *security* yang sama sehingga memiliki performa yang lebih baik. DSA dan ECDSA memiliki 2 fungsi utama yaitu pembangkitan tanda tangan dan pemeriksaan keabsahan tanda tangan.

E. Algoritma SHA-256

Algoritma SHA-256 merupakan bagian dari keluarga algoritma SHA-2. Algoritma ini dipublikasikan oleh NSA dan NIST pada tahun 2001 seiring melemahnya algoritma SHA-1 terhadap *brute force attack*. Karakteristik utama dari algoritma ini adalah panjang *hash digest* yang berukuran 256-bit.

Beberapa langkah dalam pembuatan *hash* menggunakan algoritma SHA-256 adalah sebagai berikut.

1. Padding bits yang bertujuan untuk membuat plainteks berukuran 64 bits lebih sedikit dari suatu kelipatan 512. 64 bits terakhir diisi dengan hasil pemrosesan dari plainteks awal.
2. Buffer initialization yaitu melakukan inisialisasi 8 data 32-bit sesuai spesifikasi SHA-256.
3. Compression yaitu melakukan operasi kompleks serta memecah pesan sebanyak beberapa ronde sehingga menghasilkan sebuah *output* berupa *hash digest* berukuran 256 bit.

SHA-256 sangatlah populer dan berbagai aplikasinya meliputi tanda tangan digital, *password hashing*, SSL Handshake, dan *integrity checks*.

F. Minecraft: Java Edition

Minecraft adalah sebuah 3D *sandbox game* yang dibuat oleh Mojang Studios dan sekarang dimiliki oleh Microsoft. *Game* ini diciptakan pada tahun 2009. *Gameplay* dari Minecraft berfokus pada eksplorasi dan berinteraksi dengan dunia yang dibangkitkan secara acak. Minecraft memiliki banyak jenis blok, tanaman, makhluk hidup atau *mobs*, dan barang-barang. Aktivitas yang dapat dilakukan pemain dalam game ini meliputi membuat bangunan, menambang bebatuan mulia, melawan *mobs* yang jahat, dan membuat barang baru dengan *crafting*.

Minecraft terbagi menjadi 2 edisi antara lain *Java Edition* dan *Bedrock Edition*. Minecraft Java Edition merupakan jenis Minecraft yang dibuat menggunakan bahasa Java dan hanya dapat dimainkan di PC. Sedangkan Minecraft Bedrock Edition merupakan jenis Minecraft yang dapat dimainkan dalam berbagai *platform* dan dibuat menggunakan bahasa C++. Dalam makalah ini, jenis Minecraft yang digunakan hanyalah Minecraft Java Edition karena lebih terbuka dalam penggunaan *resource pack*.

G. *Minecraft Resource Pack*

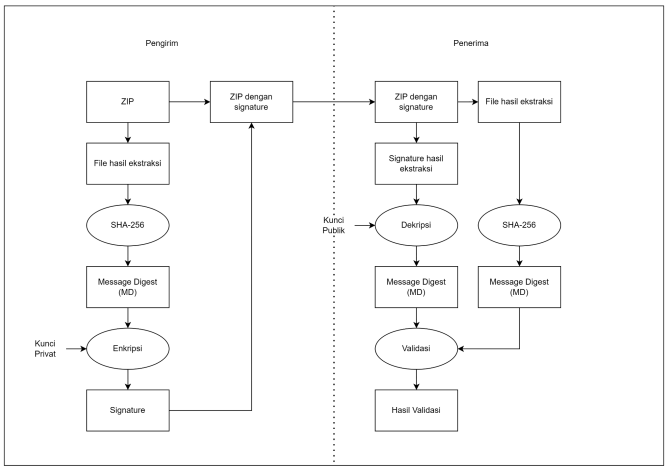
Resource Pack pada Minecraft adalah sebuah fitur pada game Minecraft yang memungkinkan pemain memodifikasi tekstur, model, musik, suara, bahasa, dan teks secara bebas. Fitur ini dapat digunakan tanpa memodifikasi *source code* dari Minecraft sehingga *accessible* bagi kebanyakan pemain untuk menggunakan atau bahkan membuatnya.

Gambar 2.3 Perbandingan tekstur Minecraft asli (kanan) dengan salah satu tekstur *custom* bernama Faithful (kiri)

Karena kemudahan dari penggunaan serta pembuatan *resource pack* dalam *game* Minecraft, banyak pemain yang membuat dan mendistribusikan *resource pack* buatannya ke dalam internet. *Resource pack* yang telah di-*download* lalu dimasukkan ke dalam folder resourcepacks dalam direktori *game* dan di-*load* di dalam *game*.

III. Rancangan Skema Penandatanganan

Resource pack pada *game* Minecraft umumnya didistribusikan dalam bentuk arsip dengan format zip. Untuk memudahkan konsistensi antara proses *hash* dalam sisi penerima dan pengirim, *hash* akan digunakan terhadap data hasil ekstraksi. Algoritma *hash* yang digunakan adalah SHA-256. Kurva yang digunakan dalam proses enkripsi dan dekripsi adalah kurva P-256 dari NIST. Tanda tangan yang dihasilkan nantinya akan disimpan ke dalam *zip* dan akan diabaikan ketika diproses di sisi penerima.



Gambar 3.1 Skema penandatanganan

Langkah-langkah penandatanganan oleh pihak pengirim yang digunakan adalah sebagai berikut.

1. Pengirim membuat ZIP berisi *resource pack.*
2. ZIP akan diekstrak oleh program dan dibaca untuk menghasilkan *message digest*.
3. *Message digest* dienkripsi menggunakan kunci privat pengirim dengan algoritma ECDSA sehingga menghasilkan tanda tangan digital.
4. Tanda tangan digital dimasukkan ke dalam suatu ZIP baru beserta data dari ZIP asli.

Langkah-langkah verifikasi oleh pihak penerima yang digunakan adalah sebagai berikut.

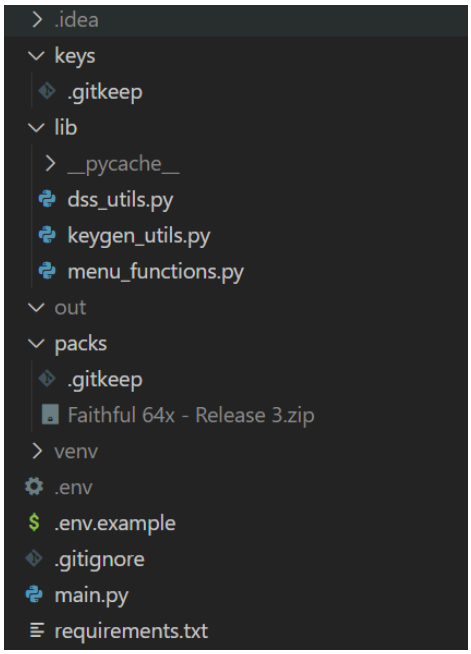
1. Penerima menerima ZIP berisi *resource pack* dan tanda tangan digital.
2. Program akan mengekstrak tanda tangan digital dari ZIP dan melakukan dekripsi dengan kunci publik pengirim sehingga menghasilkan *message digest*.
3. ZIP akan diekstrak oleh program dan dibaca untuk menghasilkan *message digest* dengan mengabaikan *file* tanda tangan digital.
4. Kedua *message digest* dibandingkan. Apabila keduanya bernilai sama maka verifikasi berhasil dan sebaliknya.

IV. Analisis dan Pembahasan

A. *Implementasi*

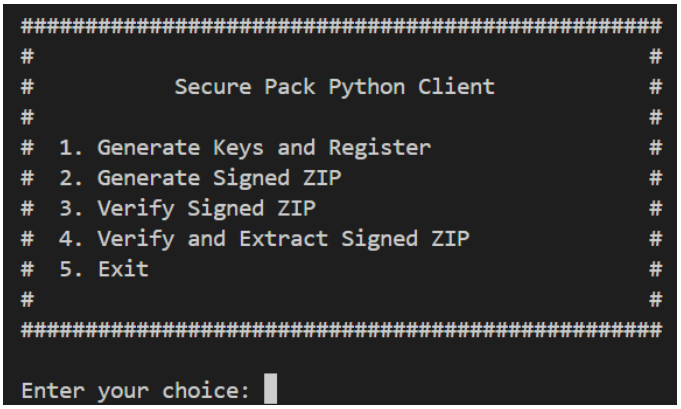
Pada implementasi penulis, penulis menggunakan beberapa bahasa Python dan dibantu sebuah pustaka kriptografi yang bernama PyCryptodome serta pustaka pengolahan file ZIP bernama zipfile dalam pembuatan *client*. *Client* ini memiliki 4 fitur utama yaitu:

1. Pembangkitan pasangan kunci privat dan kunci publik. Kedua kunci akan dibangkitkan lalu disimpan dalam folder *keys*.
2. Pembuatan ZIP dengan *signature*. Hasil ZIP dengan *signature* akan disimpan program dalam folder *out*.
3. Verifikasi ZIP yang mengandung *signature*. Program hanya akan menampilkan hasil verifikasi *signature*.
4. Verifikasi dan ekstraksi ZIP yang mengandung *signature* ke dalam folder Minecraft. Program akan menampilkan hasil verifikasi *signature* dan mengekstrak isi ZIP apabila verifikasi berhasil.



Gambar 4.1 Struktur awal program *client*

Keempat fungsi di atas diimplementasikan dan dapat digunakan melalui sebuah menu di dalam program *client*. Fungsi 1 hingga 4 masing-masing dapat diakses melalui opsi 1 hingga 4 dalam program.

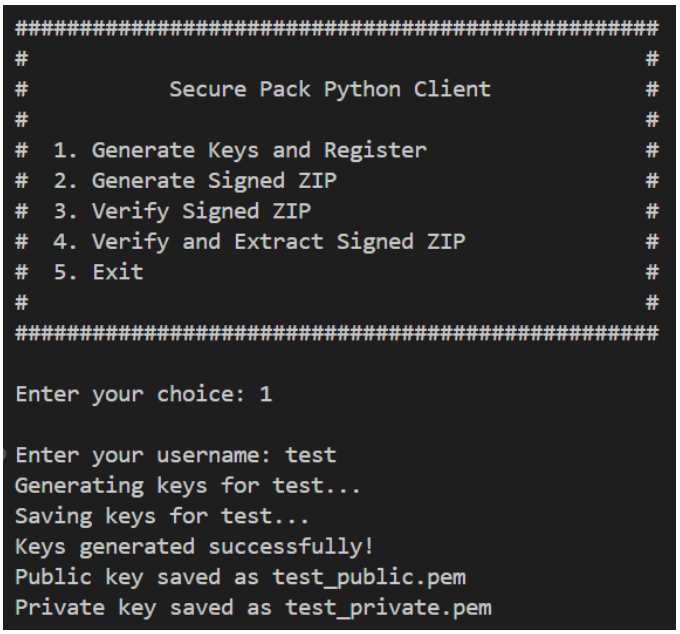


Gambar 4.2 Tampilan menu program *client*

B. Pengujian dan Analisis

Pengujian yang akan dilakukan penulis menggunakan sebuah *resource pack* yang terkenal bernama Faithful yang tersimpan dalam sebuah ZIP.

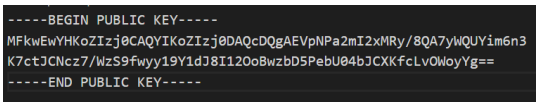
Sebelum melakukan penandatanganan digital, pengguna harus membangkitkan kunci terlebih dahulu. Kunci dibangkitkan melalui opsi 1 dengan memasukkan *username* pengguna yang harus unik dan akan menghasilkan 2 file dengan format PEM berisikan kunci untuk *username* tersebut.



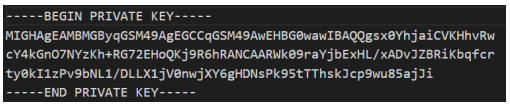
Gambar 4.3 Tangkapan layar proses registrasi untuk pembangkitan kunci

Penulis disini membuat sebuah *user* baru dengan nama test. Secara instan, program akan membuatkan dua pasang kunci publik dan privat untuk user tersebut yang disimpan sebagai test_public.pem dan test_private.pem. *Username* ini harus unik dan pengguna lain yang ingin mendaftarkan *username* ini lagi akan digagalkan agar tidak menghilangkan kunci yang sudah ada sebelumnya. Kunci yang dihasilkan adalah sebagai berikut:

1. Kunci Publik



2. Kunci Privat



Setelah penulis sudah membuat *username* dan membangkitkan dua pasang kunci, penulis melakukan penandatanganan terhadap *resource pack* yang akan menggunakan kunci privat penulis.

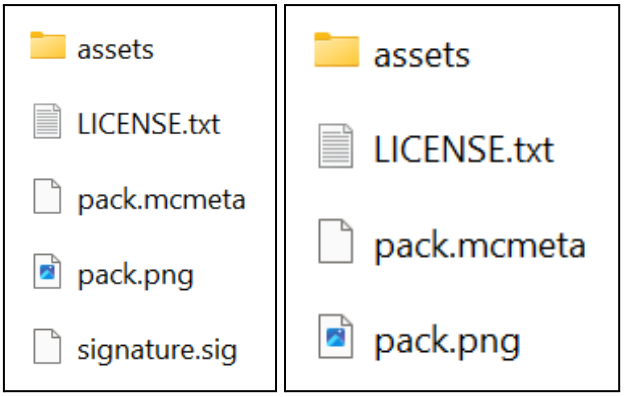
```
################################################
#                                              #
#           Secure Pack Python Client          #
#                                              #
#  1. Generate Keys and Register               #
#  2. Generate Signed ZIP                      #
#  3. Verify Signed ZIP                        #
#  4. Verify and Extract Signed ZIP            #
#  5. Exit                                     #
#                                              #
################################################

Enter your choice: 2

Enter your username: test
Enter the file path: packs/Faithful 64x - Release 3.zip
Signed ZIP file generated successfully!
```

Gambar 4.4 Tangkapan layar proses pembuatan *resource pack* dengan tandatangan

ZIP baru yang dihasilkan akan berisi file baru berupa signature.sig yang berisi tanda tangan digital untuk *resource pack* terkait.



Gambar 4.5 Perbandingan isi ZIP yang telah ditandatangan (kiri) dan sebelum ditandatangan (kanan)

Setelah membuat ZIP yang telah ditandatangani, penulis mencoba empat kasus pengujian. Pertama, penulis mencoba memverifikasi ZIP tanpa ada perubahan terhadap isi ZIP dan dengan *public key* yang benar.

```
################################################
#                                              #
#           Secure Pack Python Client          #
#                                              #
#  1. Generate Keys and Register               #
#  2. Generate Signed ZIP                      #
#  3. Verify Signed ZIP                        #
#  4. Verify and Extract Signed ZIP            #
#  5. Exit                                     #
#                                              #
################################################

Enter your choice: 3

Enter the creator's username: test
Enter the file path: out/Faithful 64x - Release 3_signed.zip
Signature verified successfully!
```

Gambar 4.6 Tangkapan layar hasil verifikasi tanpa perubahan apapun

Dapat dilihat pada gambar 4.6, *signature* berhasil divalidasi dengan benar. Lalu, untuk pengujian kedua, penulis mencoba memverifikasi ZIP namun dengan *public key* yang salah yaitu dengan mengganti isi test_public.pem dengan *public key username* lain. Penulis tidak mengubah hanya sedikit karena akan menggagalkan proses verifikasi dari format PEM.

```
################################################
#                                              #
#           Secure Pack Python Client          #
#                                              #
#  1. Generate Keys and Register               #
#  2. Generate Signed ZIP                      #
#  3. Verify Signed ZIP                        #
#  4. Verify and Extract Signed ZIP            #
#  5. Exit                                     #
#                                              #
################################################

Enter your choice: 3

Enter the creator's username: test
Enter the file path: out/Faithful 64x - Release 3_signed.zip
Signature verification failed!
```

Gambar 4.7 Tangkapan layar hasil verifikasi dengan *public key* salah

Dapat dilihat pada gambar 4.7, *signature* gagal divalidasi karena *public key* yang digunakan bukan pasangan yang benar. Lalu, untuk pengujian ketiga, penulis mencoba memverifikasi ZIP namun dengan mengubah isi dari ZIP yang telah ditandatangani. Penulis melakukannya dengan menghapus file pack.png di dalam ZIP.

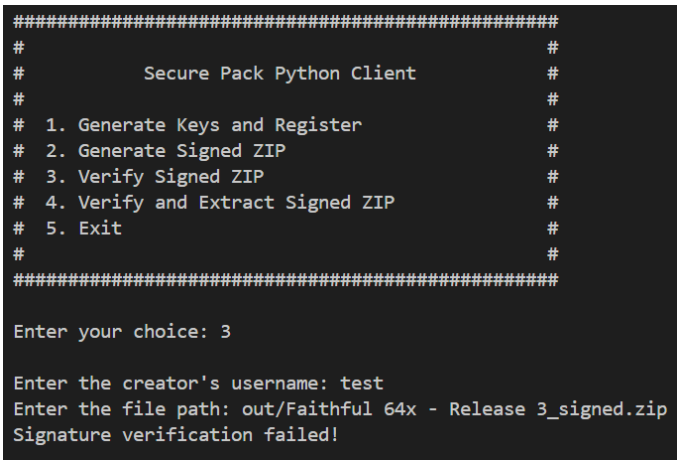
```
################################################
#                                              #
#           Secure Pack Python Client          #
#                                              #
#  1. Generate Keys and Register               #
#  2. Generate Signed ZIP                      #
#  3. Verify Signed ZIP                        #
#  4. Verify and Extract Signed ZIP            #
#  5. Exit                                     #
#                                              #
################################################

Enter your choice: 3

Enter the creator's username: test
Enter the file path: out/Faithful 64x - Release 3_signed.zip
Signature verification failed!
```

Gambar 4.8 Tangkapan layar hasil verifikasi dengan perubahan terhadap isi ZIP

Dapat dilihat pada gambar 4.8, *signature* gagal divalidasi karena adanya perubahan terhadap isi ZIP yang berpengaruh sehingga menghasilkan *hash digest* yang salah. Lalu, untuk pengujian terakhir, penulis mencoba memverifikasi ZIP namun dengan mengubah isi dari signature.sig. Penulis melakukannya dengan menambah huruf A di belakangnya.



Gambar 4.9 Tangkapan layar hasil verifikasi dengan perubahan terhadap *signature*

Dapat dilihat pada gambar 4.9, *signature* gagal divalidasi karena adanya perubahan terhadap isi file *signature* sehingga *hash digest* yang dihasilkan dari proses dekripsi akan berubah.

Kesimpulan

Hasil pengujian menyatakan bahwa program sukses melakukan verifikasi keaslian *resource pack* dan menggagalkan ZIP *resource pack* yang sudah tidak aman. Oleh karena itu, tanda tangan digital dapat dimanfaatkan untuk memastikan keaslian serta verifikasi *resource pack* pada *game* Minecraft. Tanda tangan digital ini juga dapat mencegah berbagai kasus seperti ZIP *resource pack* palsu yang telah dimodifikasi.

Untuk pengembangan lebih lanjut, penulis menyarankan adanya *Public Key Infrastructure* sehingga pendaftaran dan verifikasi dapat digunakan secara *online*. Selain itu, penulis juga menyarankan pembuatan sebuah *mod* yang mengintegrasi proses verifikasi ke dalam *game* sehingga memudahkan pemain yang ingin melakukan verifikasi.

Link Program

Berikut adalah tautan menuju implementasi program yang digunakan pada makalah ini:

https://github.com/Altair1618/SecurePack-Client

Ucapan Terima Kasih

Penulis disini ingin menyampaikan ucapan terima kasih kepada semua pihak yang telah membantu penulis dalam penyusunan makalah yang berjudul “Implementasi Digital Signature untuk Verifikasi Resource Pack pada Minecraft: Java Edition” sebagai tugas akhir mata kuliah IF4020 Kriptografi, khususnya kepada:

1. Tuhan yang Maha Esa karena atas nikmat, rahmat, dan karunia-Nya, penulis dapat menyelesaikan makalah ini.
2. Keluarga penulis, khususnya kepada kedua orang tua yang senantiasa memberikan dukungan kepada penulis.
3. Dr. Ir. Rinaldi Munir, M.T., selaku dosen pengajar mata kuliah IF4020 Kriptografi yang telah mengajar, memberi banyak ilmu, dan menyediakan arsip materi yang sangat membantu dalam pengerjaan makalah ini.
4. Teman-teman yang telah banyak membantu penulis menjalani perkuliahan.

Referensi

[1] https://goteleport.com/blog/comparing-ssh-keys/. Diakses pada tanggal 12 Juni 2024.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/01-Pengantar-Kriptografi-(2024).pdf. Diakses pada tanggal 12 Juni 2024.

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/17-Kriptografi-Kunci-Publik-2024.pdf. Diakses pada tanggal 12 Juni 2024.

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/29-Tanda-tangan-digital-2024.pdf. Diakses pada tanggal 12 Juni 2024.

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/31-DSS-2024.pdf. Diakses pada tanggal 12 Juni 2024.

[6] https://minecraft.wiki/w/Minecraft. Diakses pada tanggal 12 Juni 2024.

[7] https://minecraft.wiki/w/Resource_pack. Diakses pada tanggal 11 Juni 2024.

[8] https://www.businessofapps.com/data/minecraft-statistics/. Diakses pada tanggal 12 Juni 2024.

[9] https://www.minecraft.net/en-us/article/what-minecraft. Diakses pada tanggal 12 Juni 2024.

[10] https://www.minecraftforum.net/forums/mapping-and-modding-java-edition/resource-packs/resource-pack-discussion/3132166-the-theft-of-faithful-32x32. Diakses pada tanggal 11 Juni 2024.

[11] https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm. Diakses pada tanggal 12 Juni 2024.

[12] https://www.youtube.com/watch?v=lMLa_2hdUV8. Diakses pada tanggal 11 Juni 2024.

[13] https://www.youtube.com/watch?v=_-huRUv3S7Y. Diakses pada tanggal 12 Juni 2024.

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, 12 Juni 2024

Farhan Nabil Suryono
13521114