Penerapan Enkripsi Homomorfik untuk Mengamankan Notifikasi Sensitif guna Mencegah Pembacaan Notifikasi oleh Malware Seluler

1822171 Hans Stephano Edbert Njotohardjo (*Author*)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 18221171@std.stei.itb.ac.id

***Abstract*—Perkembangan zaman semakin meningkatkan kompleksitas serangan peretas untuk mengakses informasi, seperti serangan *malware* List Order.apk.Pdf. Makalah ini bertujuan untuk memberikan ide dalam penggunaan enkripsi homomorfik untuk mengagalkan proses peretasan dengan menlindungi notifikasi yang teradapat pada *notification pool* perangkat *mobile*. Implementasi dari ide tersebut dituangkan dalam bentuk program dengan menggunakan RSA sebagai sistem kriptografi yang paling sesuai. Perkembangan lebih lanjut dapat dilakukan dengan mengintegrasikan program ke dalam *operating system* untuk meningkatkan aspek kerahasiaan data.**

Keywords—malware; enkripsi homomorfik; rsa; list order.apk; notifikasi

I. PENDAHULUAN

Seiring perkembangan zaman, pemahaman manusia terhadap teknologi semakin meningkat. Pemahaman tersebut berkontribusi dalam peningkatan kompleksitas, baik dari sisi keamanan maupun ancaman. Pada 2023, sempat beredar sebuah *malware* berkedok sebagai berkas berekstensi .apk.Pdf yang dikirimkan melalui aplikasi WhatsApp. Sejumlah penerima pesan yang memasang aplikasi tersebut ke dalam perangkat mereka melaporkan kehilangan akses terhadap akun media sosial, surel, hingga *digital banking*.

Alur kerja malware .apk.Pdf dimulai dengan meminta akses terhadap telepon, SMS, dan pembacaan notifikasi perangkat korban. Setelah mendapatkan akses, aplikasi akan mengirimkan SMS dari perangkat korban ke perangkat pelaku untuk mendapatkan nomor telepon korban. Kemudian, pelaku berusaha mengakses akun-akun korban dengan memanfaatkan fitur ‘lupa kata sandi’. Fitur ‘lupa kata sandi’ akan meminta nomor telepon yang terdaftar dalam akun dan mengirimkan *one-time password* (OTP) pada perangkat korban jika nomor tersebut benar. Dengan akses terhadap notifikasi perangkat korban, pelaku dapat membaca OTP yang dikirimkan dan melanjutkan perubahan kata sandi akun korban.

Berdasarkan alur tersebut, akses terhadap notifikasi dan pesan menjadi salah satu kebutuhan yang krusial untuk menentukan kesuksesan proses pencurian data. Dengan demikian, upaya yang berpotensi tinggi untuk menggagalkan intrusi via malware yang sejenis adalah dengan melindungi notifikasi yang masuk. Notifikasi yang akan dilindungi hanya terbatas pada notifikasi dengan konten yang bersifat sensitif sehingga dampak negatif penerapan perlindungan notifikasi terhadap kenyamanan pengguna dapat diminimalisir.

Upaya perlindungan notifikasi dapat direalisasikan dalam bentuk program dalam sistem. Namun, dalam penerapannya, kerahasiaan pesan pengguna terhadap pihak ketiga harus terjamin sehingga program tidak boleh melakukan analisis terhadap pesan pengguna secara langsung. Untuk itu, program dapat memanfaatkan enkripsi homomorfik untuk melakukan analisis terhadap pesan yang masuk dan melakukan pelabelan terhadap notifikasi tanpa mengetahui isi pesan secara langsung.

Terdapat beberapa jenis enkripsi homomorfik yang tersedia. Oleh sebab itu, dilakukan analisis terhadap beberapa algoritma enkripsi homomorfik untuk menentukan algoritma yang paling cocok untuk diaplikasikan ke dalam pembuatan program.

II. DASAR TEORI

A. Malware

Malware, kependekan dari *Malicious Software*, merupakan suatu perangkat lunak berbahaya yang sengaja dirancang dengan tujuan untuk mencuri data atau merusak sistem dari perangkat pengguna. Dengan spektrum perangkat lunak yang luas, malware disebarluaskan ke dalam berbagai format, seperti virus, ransomware, adware, ataupun trojan. Setiap format malware bertindak dengan cara berbeda untuk mendapatkan akses tidak sah dari sistem yang ditargetkan.

Virus menjadi jenis malware yang awam ditemukan di berbagai perangkat. Virus bekerja dengan menginjeksikan dirinya dan merusak file bersih dalam sistem dan menyebarkan dirinya ketika file tersebut disebarkan. Ransomware merupakan suatu jenis malware yang bekerja dengan melakukan penguncian terhadap file dalam perangkat hingga perangkat itu sendiri. Salah satu wabah ransomware yang terkenal di Indonesia adalah Ransomware WannaCry yang mengunci akses korban terhadap komputer mereka serta mengancam untuk merusak seluruh file komputer jika mereka tidak membayar sejumlah uang.

Selain itu, bentuk lain dari malware modern adalah adware. Sesuai namanya, adware menampilkan berbagai iklan di

perangkat pengguna guna menghasilkan pendapatan melaluinya. Secara praktik, adware biasanya menyamar sebagai aplikasi lain, seperti aplikasi game atau cleaner dalam android. Perilaku penyamaran adware mengindikasikan bahwa adware bisa saja merupakan percabangan dari malware trojan. Malware trojan berfokus pada penyamaran perangkat lunak sebagai aplikasi lain guna tidak memunculkan rasa curiga korban terhadap malware.

B. Penyebaran Malware via WhatsApp

Sejak 2023, terjadi praktik penyebaran massal malware via WhatsApp. Penyebaran tersebut memanfaatkan teknik social engineering berdasarkan event besar yang sedang terjadi atau kegiatan yang awam dilakukan oleh masyarakat Indonesia. Beberapa diantaranya adalah pengiriman paket oleh kurir dengan mengirimkan file 'List Order.apk.Pdf', pembelian tiket Coldplay dengan mengirimkan file 'Coldplay in Jakarta.apk', hingga pengiriman secara acak terkait data TPS pemilu presiden 2024 dengan file 'Data Tps Pemilu 2024.apk'. Setelah pengiriman terjadi, penyebar akan langsung memblokir nomor korban untuk menghindari konfrontasi.

Secara teknik, penyebaran tersebut bersifat repetitif dan mudah diidentifikasi, yakni dengan melihat ekstensi pada file yang dikirimkan. Namun, generasi-generasi yang lebih tua memiliki kesulitan untuk memahami teknologi masa kini. Terlebih, generasi-generasi tersebutlah yang umumnya memiliki jumlah uang dengan nominal yang lebih banyak. Dengan demikian, teknik penyebaran malware .apk melalui WhatsApp masih tergolong efektif.

C. Kriptografi

Kriptografi merupakan ilmu yang mempelajari tentang keamanan informasi atau pesan yang dikirimkan melalui berbagai media perantara. Proses utama yang terjadi di dalam kriptografi tersusun atas enkripsi dan dekripsi. Enkripsi merupakan proses mengubah pesan pengguna atau plainteks menjadi bentuk yang tidak dapat dibaca atau cipherteks, sementara dekripsi sebaliknya. Konsep ini telah digunakan sejak zaman dahulu dengan tujuan untuk memastikan perlindungan terhadap pesan sehingga hanya penerima pesan yang dapat membaca isi pesan. Sementara di era modern, terdapat 4 aspek fundamental yang harus dipenuhi dalam pembentukan algoritma kriptografi:

(1) Autentikasi yang memastikan keaslian pengirim dan penerima pesan,

(2) Kerahasiaan yang memastikan keterjagaan isi pesan dari pembaca pihak ketiga,

(3) Keutuhan yang memastikan tidak ada data yang hilang selama proses enkripsi dan dekripsi, serta

(4) Kenirsangkalan yang mencegah terjadinya penyangkalan terhadap pengiriman pesan oleh pengirim atau penerimaan pesan oleh penerima.

Berdasarkan penggunaan kunci pada proses enkripsi dan dekripsi, algoritma kriptografi dapat dibedakan menjadi 2: simetris dan asimetris. Algoritma kriptografi kunci simetris memanfaatkan kunci yang sama untuk melakukan enkripsi dan dekripsi. Beberapa algoritma kriptografi kunci simetris, antara lain Advanced Encryption Standard (AES) dan Data Encryption Standard (DES). Kriptografi kunci simetris seringkali dipakai untuk proses yang lebih mengutamakan kecepatan, seperti enkripsi terhadap data pribadi atau dokumen. Sementara itu, algoritma kunci asimetris memanfaatkan setidaknya 2 kunci yang berbeda, masing masing untuk melakukan enkripsi dan dekripsi. Beberapa algoritma kunci asimetris yang populer adalah Rivest-Shamir-Adleman (RSA) dan Elliptic-Curve Cryptography (ECC). Umumnya, algoritma kunci asimetris digunakan dalam proses yang membutuhkan keamanan yang lebih tinggi, seperti dalam pengiriman pesan chat atau email melalui internet.

D. Enkripsi Homomorfik

Dalam aljabar, homomorfik berarti peta struktur yang dipertahankan dari dua struktur aljabar yang bertipe sama. Jika diartikan dalam kriptografi, maka enkripsi homomorfik menjadi teknik kriptografi yang menjaga struktur cipherteks setelah melalui suatu operasi biner spesifik (umumnya, penjumlahan dan perkalian). Dengan kata lain, enkripsi homomorfik memberikan kemampuan bagi pengguna untuk melakukan operasi terhadap data cipherteks tanpa melakukan proses dekripsi dengan hasil operasi yang bernilai sama dengan operasi yang dilakukan terhadap data plainteks secara langsung. Aplikasi dari enkripsi homomorfik menjamin kerahasiaan data dalam tahap pemrosesan oleh suatu sistem.

Secara umum, fungsi enkripsi homomorfik dapat digolongkan ke dalam 2 tipe: homomorfik aditif dan homomorfik multiplikatif. Penggolongan ini didasarkan pada hasil yang diterima apabila dioperasikan suatu operator terhadap data cipherteks. Berikut notasi untuk setiap golongan.

$$E(p_1) \otimes E(p_2) = E(p_1 + p_2) \tag{1}$$

$$E(p_1) \otimes E(p_2) = E(p_1 \cdot p_2) \tag{2}$$

Berdasarkan jenisnya, enkripsi homomorfik dapat dibedakan menjadi *partially, somewhat,* dan *fully homomorphic encryption*.

1) Partially Homomorphic Encryption (PHE)

PHE merupakan enkripsi homomorfik yang paling sering digunakan dalam berbagai sistem. PHE mengizinkan pengguna untuk hanya melakukan satu jenis operasi terhadap cipherteks tanpa batasan jumlah. Hal ini mengindikasikan sifat homomorfik cipherteks akan hilang ketika dioperasikan perkalian dan penjumlahan secara bersamaan.

TABLE I. DAFTAR ALGORITMA PHE

Algoritma	***Aditif***	***Multiplikatif***
RSA		✅
ElGamal		✅
Exponential ElGamal	✅	
Elliptic Curve ElGamal	✅	
Paillier	✅	

Algoritma	*Aditif*	*Multiplikatif*
Damgard-Jurik	✅	
Benaloh	✅	
Naccache-Stern	✅	
Okamoto-Uchiyama	✅	

2) Somewhat Homomorphic Encryption (SHE)

Dalam perkembangan enkripsi homomorfik, penemuan SHE merupakan suatu batu loncatan. Sebagai perkembangan PHE, SHE mengizinkan penggunanya untuk melakukan berbagai jenis operasi terhadap cipherteks walaupun dalam jumlah yang terbatas. Penemuan SHE mengubah batas *threshold* dan memampukan operasi yang lebih kompleks untuk dilakukan terhadap cipherteks.

3) Fully Homomorphic Encryption (FHE)

Sebagai penyempurnaan dari SHE, FHE menghilangkan limitasi jumlah operasi yang dapat dilakukan terhadap data cipherteks. Namun, untuk mewujudkan hal tersebut, FHE membutuhkan pemakaian tinggi atas sumber daya komputasi serta waktu. Hal didukung dengan klaim IBM atas versi pertama dari HElib milik mereka yang bekerja dengan waktu 100 triliun kali lebih lambat dibandingkan jika dikerjakan secara langsung pada plainteks. Selain IBM, Microsoft dan Google turut membuat *library* versi mereka masing-masing, yakni SEAL & Private Join and Compute.

E. RSA

Rivest-Shamir-Adleman (RSA) merupakan algoritma kriptografi kunci publik yang sangat sering ditemukan dalam berbagai bentuk aplikasi. Jika melihat sisi keamanan, RSA dinilai tetap efektif dalam menghadang usaha-usaha kriptanalisis tradisional walaupun tidak secara kuantum. Hal disebabkan karena pasangan faktor bilangan prima dengan nilai yang sangat besar dapat dianalisis dengan jauh lebih cepat menggunakan *quantum computing*.

Dalam penggunaannya, RSA tersusun atas 3 prosedur: pembangkitan kunci, enkripsi, dan dekripsi. Berikut detail dari setiap prosedur.

1) Pembangkitan Kunci

- Tentukan 2 bilangan prima p & q dengan $p \neq q$
- Hitunglah nilai $n = pq$
- Hitunglah nilai $\phi(n) = (p - 1)(q - 1)$
- Tentukan sebuah bilangan bulat e yang relatif prima terhadap $\phi(n)$ sebagai kunci enkripsi
- Tentukan sebuah bilangan bulat d sebagai kunci dekripsi menggunakan persamaan

$$ed = 1 \ (\textbf{mod}\ \phi(n)) \quad (3)$$

- Petakan pasangan (e, n) sebagai kunci publik
- Petakan pasangan (d, n) sebagai kunci privat

2) Enkripsi

Untuk melakukan enkripsi RSA, diperlukan pasangan kunci publik (e, n). Berikut persamaan yang dipakai untuk menghasilkan cipherteks c dari plainteks m dengan modulo n dan kunci publik e.

$$c = m^e \ \textbf{mod}\ n \quad (4)$$

3) Dekripsi

Untuk melakukan dekripsi RSA, diperlukan pasangan kunci privat (d, n). Berikut persamaan yang dipakai untuk menghasilkan kembali plainteks m dari cipherteks m dengan modulo n dan kunci privat d.

$$m = c^d \ \textbf{mod}\ n \quad (5)$$

Berdasarkan TABLE I, RSA tergolong ke dalam algoritma enkripsi PHE multiplikatif. Hal ini dapat dijelaskan menggunakan teknik aljabar terhadap persamaan yang disediakan. Misal terdapat 2 nilai m yang berbeda, dinotasikan dengan m_1 dan m_2. Sesuai dengan persamaan (4), kedua nilai m tersebut akan dipangkatkan terhadap nilai e yang sama. Dengan memakai sifat distributif dari teknik perpangkatan, maka dapatkan:

$$c_1 c_2 = (m_1 m_2)^e \ \textbf{mod}\ n \quad (6)$$

III. DESAIN DAN IMPLEMENTASI

A. Desain Proses

Untuk melindungi pengguna WhatsApp dari jenis malware terkait, diusung solusi mengagalkan proses peretasan dengan melindungi notifikasi agar tidak dapat dibaca oleh pihak ketiga. Secara proses, perlindungan tersebut dapat dilakukan dengan cara menampung notifikasi yang masuk, melakukan pemeriksaan terhadap isi konten, lalu mengirimkan:

(1) notifikasi awal untuk notifikasi yang tidak mengandung informasi sensitif, atau

(2) notifikasi terlindungi untuk notifikasi yang mengandung informasi sensitif.

Dalam proses tersebut, aspek yang perlu diperhatikan adalah kerahasiaan. Dalam kasus ini, pada tahap pemeriksaan, proses yang terjadi juga dianggap tidak boleh mengakses pesan pengguna secara langsung. Untuk mengatasi batasan ini, dibutuhkan enkripsi homomorfik pada pesan sehingga pemrosesan terjadi tidak pada plainteks, tetapi pada cipherteks. Dengan demikian, kepatuhan terhadap aspek kerahasiaan dari pesan pengguna sudah berhasil diraih oleh sistem. Jika digambarkan menggunakan diagram alir, desain proses yang diusulkan adalah sebagai berikut.

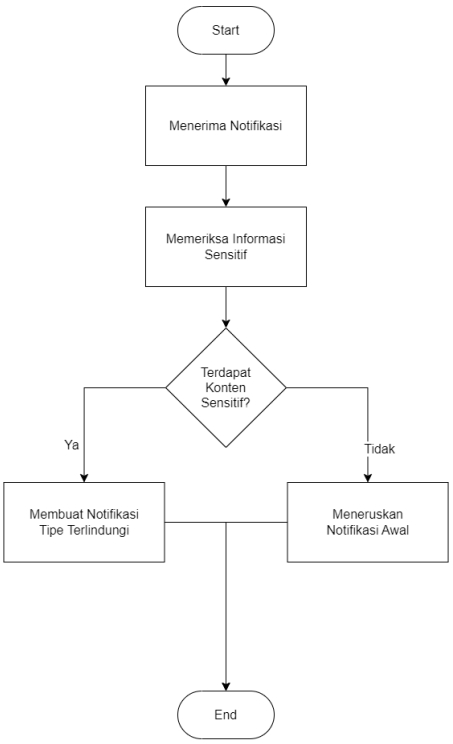

Fig. 1. Diagram Alir Program

Pada tahap awal, notifikasi yang masuk akan ditahan atau diambil datanya dan dienkripsikan menggunakan enkripsi homomorfik. Selain itu, disediakan sebuah array yang berisikan *keywords* yang mengindikasikan keberadaan konten sensitif dalam pesan, seperti variasi kata (*case-sensitive*) 'OTP'. Plainteks *keywords* yang dipakai dapat berisikan 1 hingga 2 kata. Jika plainteks *keywords* terdiri atas 2 kata, setiap kata akan dienkripsi homomorfik secara terpisah dan hasil keduanya akan melalui suatu operasi untuk menghasilkan nilai yang satu.

Pada tahap pemeriksaan, dilangsungkan sebuah fungsi atau proses. Proses tersebut akan menerima masukan cipherteks pesan dan *array* cipherteks dari *keywords*. Cipherteks pesan akan dibandingkan dengan setiap elemen dari *array*. Keluaran dari proses ini akan berbentuk nilai *boolean* yang akan dikembalikan. Nilai boolean *true* akan dikembalikan bila hasil perbandingan bernilai sama, dan *false* jika tidak ada nilai yang sama dengan cipherteks pesan di dalam *array*.

Hasil yang diberikan pada tahap pemeriksaan akan dipakai pada tahap pengiriman notifikasi. Jika dihasilkan nilai *true* dari tahap sebelumnya, maka pesan pada notifikasi akan dihapus dan diganti dengan suatu *placeholder*. Namun, jika nilai yang dihasilkan adalah *false*, maka pesan akan ditampilkan secara normal. Setelah notifikasi dikembalikan, maka program telah selesai dieksekusi. Untuk memudahkan pemahaman terkait proses yang dilalui, silakan merujuk kepada *Fig 2.*

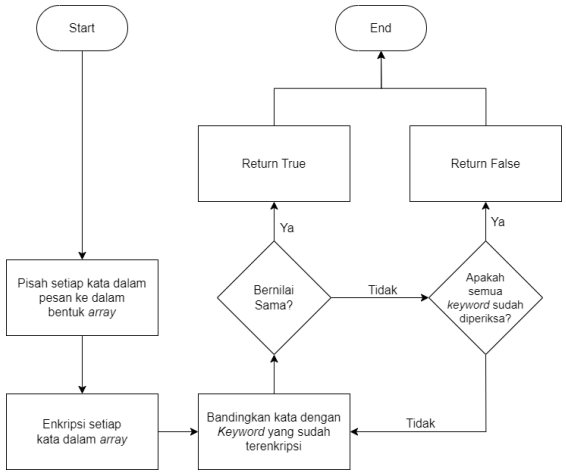

Fig. 2. Diagram Alir Tahap Pemeriksaan

B. *Desain Arsitektur*

Berdasarkan kebutuhan tersebut, algoritma yang paling cocok untuk digunakan adalah algoritma enkripsi RSA. Alasan utama penggunaan RSA dalam sistem tersebut adalah karena RSA mempertahankan bentuk hasil enkripsi untuk plainteks yang sama. Hal ini dibutuhkan karena selama tahap pemeriksaan, proses dalam sistem hanya menerima cipherteks tanpa mengetahui kunci publik atau kunci privat yang digunakan. Dengan persistensi hasil enkripsi, proses perbandingan antara *keywords* dengan cipherteks yang terjadi di dalam tahap pemeriksaan dapat dilansungkan. Selain itu, pemilihan algoritma enkripsi RSA juga dipengaruhi oleh faktor kesederhanaan cara perhitungan dibandingkan algoritma PHE lainnya. Jika dibandingkan terhadap SHE dan FHE, maka pemilihan RSA sudah dinilai tepat dalam aspek ukuran dan kecepatan proses.

Pengembangan program yang dilakukan adalah dengan menggunakan bahasa pemrograman Python dengan penggunaan *library* LightPHE untuk menjamin proses enkripsi dan dekripsi yang dilakukan. Proses yang akan direkayasa pada program ini hanya berfokus pada tahap pemeriksaan, yakni tahapan yang mengimplementasikan proses enkripsi dan dekripsi homomorfik dalam program.

C. *Implementasi*

Dengan rancangan implementasi yang telah dibuat, berikut tampilan implementasi program menggunakan *command line interface*.

```
[ ... PROGRAM STARTED ... ]
Application Name:- Messages (SMS)
Notification Title:- OVO
Notification Description:- HINDARI PENIPUAN, JANGAN BERIKAN KODE INI
 KE SIAPAPUN TERMASUK PIHAK YANG MENGAKU DARI OVO Kode Verifikasi: 9
56491 Hubungi 1500696 untuk bantuan

[... RESULT: Sensitive Keywords Detected ...]
Messages (SMS)
OVO - Tap to View Message

Continue..? (y/n)
```

Fig. 3. Tampilan Program

Program tersebut dimulai dengan keadaan sudah memiliki *array of keywords* yang akan dipakai. Ketika dijalankan, program menginisiasi kunci enkripsi dan dekripsi yang akan digunakan dan menggunakannya untuk melakukan enkripsi terhadap isi *array of keywords*. Hasil enkripsi tersebut akan disimpan dalam suatu *array* dan dipakai pada tahap pemeriksaan.

Berdasarkan desain, program akan membaca notifikasi yang masuk ke dalam notification pool pada *smartphone* dan memeriksa konten atau data pada notifikasi tersebut. Dalam program ini, proses tersebut digantikan dengan pengguna memasukkan secara mandiri terkait notifikasi yang sekiranya masuk. Data notifikasi yang dimasukkan untuk tahap ini adalah sebagai berikut.

- *Application Name:* Messages (SMS)
- *Notification Title:* OVO
- *Notifiaction Description:*

  HINDARI PENIPUAN, JANGAN BERIKAN KODE INI KE SIAPAPUN TERMASUK PIHAK YANG MENGAKU DARI OVO Kode Verifikasi: 956491 Hubungi 1500696 untuk bantuan

Dengan diterimanya data notifikasi, dilakukan enkripsi terhadap setiap kata di dalamnya untuk–secara konsepsi–dikirimkan sebagai masukan tahap pemeriksaan. Enkripsi dilakukan sebanyak 2 iterasi sehingga dihasilkan 2 array yang berisikan data terenkripsi tersebut. Enkripsi iterasi pertama dilakukan untuk mengenkripsi setiap kata, sementara enkripsi iterasi kedua dilakukan untuk 2 kata sekaligus. Hal ini dilakukan untuk mendeteksi *keywords* yang mengandung 2 kata, seperti ‘Kode Verifikasi:’ pada contoh.

Dengan kedua *array* yang sudah terisi, maka dilakukan pencocokan atau komparasi terhadap data notifikasi dengan data *keywords*. Jika berhasil ditemukan kecocokan, maka *notification description* akan digantikan dengan suatu *placeholder*. Jika tidak ditemukan kecocokan, maka tidak dilakuakan perubahaan apapun terhadap masukan data notifikasi.

Setelah melewati keseluruhan tahapan tersebut, program menanyakan kembali untuk mengulang kembali masukan atau tidak. Aktivitas ini dilakukan sebagai representasi program yang seharusnya senantiasa berjalan selama *smartphone* hidup. Berikut kode program yang telah dibuat.

```
from lightphe import LightPHE

def encode(a):
    return int(''.join([str(ord(c)) for c in a]))


def main():
    p = LightPHE(algorithm_name="RSA")
    keywords = ["otp", "OTP", "OTP:", "(OTP)", "Kode Verifikasi", "kode
verifikasi", "Kode verifikasi", "KODE VERIFIKASI", "Kode Verifikasi",
"Kode Verifikasi:"]

    print("[ ... PROGRAM STARTED ... ]")

    encKeywords = []

    for i in range (len(keywords)):
        splitted = keywords[i].split(" ")
        if (len(splitted) > 1):
            encKeywords.append(p.encrypt(plaintext=(encode(splitted[0]) *
encode(splitted[1]))))
        else:
            encKeywords.append(p.encrypt(plaintext=(encode(splitted[0]))))


    loop = True
    while (loop):
        appName = input("Application Name:- ")
        appNTitle = input("Notification Title:- ")

        # Data dalam Notifikasi, dipisahkan per kata
        appNDesc = input("Notification Description:- ").split(" ")

        # Enkripsi terhadap data notifikasi
        encNDesc = []
        encNDesc2 = []
        for i in range (len(appNDesc)):
            cipher = p.encrypt(plaintext=encode(appNDesc[i]))
            encNDesc.append(cipher)
            if (i > 0):
                cipher2 = p.encrypt(plaintext=(encode(appNDesc[i-1]) *
encode(appNDesc[i])))
                encNDesc2.append(cipher2)

        # Membandingkan terhadap Keywords 1 Kata
        state = False
        for i in range (len(encNDesc)):
            for j in range (len(encKeywords)):
                if encNDesc[i].value == encKeywords[j].value:
                    state = True
                    break
            if (state):
                break

        # Membandingkan terhadap Keywords 2 Kata
        if not (state):
            for i in range (len(encNDesc2)):
                for j in range (len(encKeywords)):
                    if encNDesc2[i].value == encKeywords[j].value:
                        state = True
                        break
                if (state):
                    break


        print()
        print("[... RESULT: ", end="")

        if (state):
            # Placeholder
            print("Sensitive Keywords Detected ...]")
            print(appName)
            print(appNTitle, "- Tap to View Message")
        else:
            print("No Sensitive Keywords Detected ...]")
            print(appName)
            print(appNTitle, "-", " ".join(appNDesc))

        print()
        cont = input("Continue..? (y/n) ")
        if (cont.lower() != "y"):
            loop = False
            print("[... PROGRAM END ...]")


    return 0

main()
```

IV. PENUTUP

A. *Kesimpulan*

Program *notification masking*, *covering*, ataupun *securing* menjadi langkah preventif lapisan akhir dengan tujuan mengagalkan peretas untuk membaca data pengguna melalui notifikasi. Program ini menekankan nilai kerahasiaan data, bahkan terhadap pemberi layanan, sehingga dibutuhkan aplikasi enkripsi homomorfik yang cepat, ringan, dan mudah diimplementasikan. Untuk itu, digunakan sistem kriptografi RSA sebagai *Partially Homomorphic Encryption Algorithm*

untuk mengubah data ke dalam bentuk cipherteks yang dapat dimanipulasikan untuk kebutuhan *secured notification*.

B. Saran

Pada mulanya, rancangan arsitektur program meruapakan sebuah program aplikasi *mobile standalone*. Namun, melihat isu yang diangkat dari topik ini adalah terkait kerahasiaan, maka akan lebih baik jika program ini diaplikasikan sebagai bagian dari bagian dari *operating system* atau *security application* yang sudah tersedia sejak pabrik. Sebagai bagian dari *operating system*, maka risiko serangan melalui teknik atau *malware* sejenis akan berkurang dengan signifikan. Selain itu, pengembangan program ini juga ternilai mudah dan tidak memberatkan proses pada perangkat.

REFERENCES

[1] Security Awareness Indonesia, "Analisa Android Malware List Order.PDF.APK.PDF | BSSN | Security Awareness," YouTube, 03-Jun-2023. [Video file]. Available: https://www.youtube.com/watch?v=d_cwGZhzty0. [Accessed: Jun. 12, 2024].

[2] AVG Technologies, "What Is Malware? | How It Works & What It Does," avg.com. [Online video]. Available: https://youtu.be/6jYiMde0JCI?feature=shared. [Accessed: Jun. 12, 2024].

[3] Keyfactor, "What is homomorphic encryption, and why isn't it mainstream?" keyfactor.com. [Online]. Available: https://www.keyfactor.com/blog/what-is-homomorphic-encryption/. [Accessed: Jun. 12, 2024].

[4] S. Halevi and V. Shoup, "BGV_country_db_lookup," in HElib. [Online]. Available: https://github.com/homenc/HElib/tree/master/examples/BGV_country_db_lookup. [Accessed: Jun. 12, 2024].

[5] R. Munir, "Algoritma RSA," Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2023-2024/11-Algoritma-RSA-2024.pdf . [Accessed: Jun. 12, 2024].

[6] R. Munir, "Enkripsi Homomorfik" itb.ac.id. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/38-Enkripsi-homomorfik-2024.pdf. [Accessed: Jun. 12, 2024].

[7] S. I. Serengil, "LightPHE: A Lightweight Partially Homomorphic Encryption Library for Python," GitHub. [Online]. Available: https://github.com/serengil/LightPHE. [Accessed: Jun. 12, 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

Hans Stephano Edbert Njotohardjo 18221171