Peningkatan Keamanan *On-Device Model* pada Aplikasi *Offline* Android dengan AES dan Obfuskasi

Munazir Dzuana Setiawan, Cecep Nurul Alam, dan Jumadi

Universitas Islam Negeri Sunan Gunung Djati Bandung
Jl. A.H. Nasution No. 105A, Cibiru, Kota Bandung, Jawa Barat
E-mail : munazir805@gmail.com, cecep@uinsgd.ac.id, jumadi@uinsgd.ac.id.

Abstrak

Model AI merupakan sebuah hasil pelatihan data dengan arsitektur *machine learning*. Salah satu implementasinya dengan metode *on-device model,* yang memiliki keunggulan seperti waktu respons yang cepat, hemat bandwidth, pengurangan biaya komputasi cloud. Metode *on-device mode*l pada aplikasi Android memiliki kerentanan terhadap serangan *reverse engineering* atau *decompiling*, yang menyebabkan pencurian model sehingga dapat diakses tanpa izin. Kombinasi antara kriptografi dan obfuskasi pada aplikasi Android menjadi salah satu solusi dalam melindungi model kecerdasan buatan (AI) dari potensi eksploitasi. Kriptografi berfungsi untuk melindungi model AI, dan obfuskasi berfungsi untuk mengaburkan sumber kode aplikasi Android. Penelitian ini menggunakan model *MobileNet* V1 TensorFlow Lite yang dienkripsi menggunakan algoritma AES (Advanced Encryption Standard), dan proses dekripsi dilakukan dengan multi-threading, serta teknik obfuskasi menggunakan R8. Kombinasi AES, obfuskasi, dan implementasi multi-threading memberikan perlindungan yang baik dan efektif untuk model AI dalam aplikasi Android offline.

Kata kunci : Android, Kriptografi, Obfuskasi, AES, R8.

Pendahuluan

Integrasi *Machine Learning* (ML) atau *Deep Learning* (DL) ke dalam layanan digital atau aplikasi seringkali melibatkan interaksi berulang dengan *cloud*, yang dapat meningkatkan risiko privasi bagi miliaran pengguna [1]. Data yang diproses di *cloud*, seperti rekaman suara yang mengandung informasi biometrik unik, berpotensi disalahgunakan misalnya, dalam serangan peniruan identitas atau penyebaran rekaman palsu. Mengatasi masalah ini perlunya penerapan *On-Device Model*. Model AI merupakan hasil pelatihan data dengan arsitektur machine learning dan dapat diimplementasikan dengan pendekatan *On-Device Model* menawarkan solusi dengan beberapa keunggulan seperti penghematan *bandwidth* komunikasi, pengurangan biaya komputasi cloud, waktu respons yang lebih cepat, dan yang paling penting, menjaga data tetap berada di perangkat lokal [2]. Hal ini sangat penting terutama pada bagi aplikasi yang membutuhkan respons cepat dan tingkat privasi tinggi, sehingga pengguna dapat lebih terjaga, mengurangi risiko penyalahgunaan data sensitif.

Penggunaan *Machine Learning* (ML) dan *Deep Learning* (DL) pada aplikasi meningkat pesat berkat metode penerapan ML/DL pada perangkat seluler [3]. Pada tahun 2017, beberapa perusahaan meluncurkan *framework* DL untuk perangkat seluler dan *edge computing*, seperti TensorFlow Lite dari Google, Caffe2 dari Facebook, CoreML dari Apple, NCNN dari Tencent, dan MDL dari Baidu [4]. Salah satu metode yang paling populer untuk menjalankan model AI pada perangkat seluler menggunakan TensorFlow Lite. TensorFlow Lite merupakan versi ringan dari framework Tensorflow. *Framework* ini memungkinkan konversi model yang ada ke dalam format (*.tflite*) yang telah dioptimalkan untuk penggunaan pada perangkat mobile, seperti pada aplikasi Android.

Namun, ditemukan bahwa dari 337 aplikasi di Play Store menggunakan TensorFlow Lite, sebanyak 271 aplikasi tidak menerapkan langkah-langkah keamanan pada model mereka, yang membuat model tersebut rentan diekstraksi dengan mudah dengan teknik *reverse engineering* [5]. Hal ini menjadi sebuah masalah, jika model disebarluaskan tanpa adanya perlindungan yang memadai, pihak luar organisasi dapat menggunakan model AI tersebut tanpa izin. Model AI dalam tingkat produksi dianggap sebagai kekayaan intelektual karena data pelatihannya sering kali sulit diperoleh, dan membuat model yang akurat serta efisien membutuhkan keahlian yang mendalam [1]. Perlindungan terhadap model yang digunakan pada perangkat

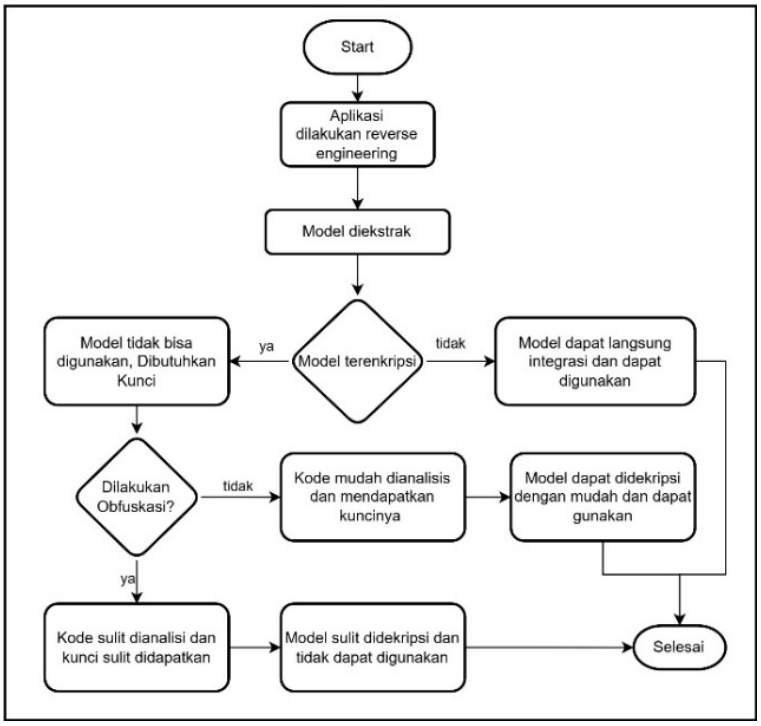
DOI : http://dx.doi.org/10.32409/jikstik.24.2.3781

seluler menjadi sangat penting mengingat pelatihan model mahal dalam data dan daya komputasi, model tingkat produksi selalu menjadi target penyerang [5].

Terdapat penelitian terdahulu yang berkaitan dengan enkripsi model di perangkat seluler, tentang enkripsi dan dekripsi real-time untuk melindungi model *machine learning* pada Android. Penelitian tersebut mengusulkan algoritma AES yang terbukti cepat dan efisien untuk untuk enkripsi dan dekripsi model pembelajaran mesin di berbagai perangkat, termasuk ponsel dengan daya komputasi rendah. Hasil penelitian untuk model Inception V3 membutuhkan waktu 2079 ms dan 2097 ms untuk dekripsi pada CPU 8 core dan 4 core. Pada penelitian tersebut menyarankan untuk melakukan optimalisasi pada model yang sangat besar seperti membuat beberapa fragmen kecil saat dienkripsi agar fragmen-fragmen ini dapat didekripsi secara terpisah pada thread yang berjalan paralel, dan juga merekomendasikan menyimpan kunci model secara lokal di APK Android yang aman dari *decompiling* [3].

Adapun penelitian sebelumnya yang menggunakan metode enkripsi simetris dan asimetris dengan memanfaatkan *Trusted Execution Environment* (TEE) pada platform ARM *TrustZone* serta *Sanctuary Enclave*. Implementasi OMG *(Offline Model Guard)* pada ARM HiKey 960 menunjukkan kemampuan menjalankan pengenalan kata kunci secara offline dengan akurasi 75%. Penggunaan Sanctuary untuk isolasi dua arah tidak memberikan dampak signifikan terhadap waktu eksekusi (379 ms tanpa OMG dan 387 ms dengan OMG), sehingga menjamin kinerja yang efisien [1].

Pada gambar 1 menjelaskan bahwa ketika penyerang melakukan *reverse engineering* pada aplikasi offline, maka model yang tidak dilakukan enkripsi rentan untuk dicuri dan digunakan oleh pihak yang tidak memiliki akses yang sah. Jika model yang terenkripsi namun tidak ada perlindungan pada aplikasinya seperti obfuskasi maka sumber kode dapat dianalisis dan kunci untuk dekripsi model AI didapatkan dengan mudah sehingga model digunakan oleh pihak yang tidak mempunyai akses sah. Perlunya kriptografi untuk mengenkripsi model tersebut dan melakukan obfuskasi pada aplikasi Android untuk mengatasi masalah keamanan pada model AI di aplikasi Android. Kriptografi akan melindungi model AI dari akses yang tidak sah, sedangkan obfuskasi akan membuat sumber kode lebih sulit dipahami dan dimodifikasi [4], [5].



Gambar 1: *Flowchart b*agaimana pihak tidak berwenang mendapatkan model

Kriptografi adalah ilmu atau seni untuk menjaga kerahasiaan pesan dengan cara menjadikannya ke dalam bentuk yang tidak dapat dimengerti lagi maknanya. Terdapat 2 konsep utama dalam proses kriptografi yaitu enkripsi dan dekripsi. enkripsi data menjadi bentuk yang tidak dapat dikenali atau biasa disebut ciphertext bertujuan untuk melindungi informasi dari akses yang tidak sah, sedangkan untuk dekripsi mengembalikan data ciphertext menjadi bentuk aslinya biasa disebut plaintext [6]. Kombinasi dengan kedua konsep ini menjadikan kriptografi pilihan utama untuk keamanan digital. Proses kriptografi yakni enkripsi dan dekripsi bekerja memerlukan kunci. Saat ini algoritma krip-

tografi dibedakan menjadi 2 jenis yaitu algoritma simetri dan asimetris. Algoritma simetris menggunakan satu kunci tunggal yang sama untuk proses enkripsi dan dekripsi. Algoritma asimetris menggunakan 2 kunci yaitu kunci publik dan kunci privat. Kunci publik digunakan untuk melakukan proses enkripsi yang bersifat umum dan dapat dibagikan, sedangkan kunci privat untuk melakukan proses dekripsi yang bersifat rahasia dan tidak boleh dibagikan hanya pihak yang berhak melakukan dekripsi [7]. Masing-masing algoritma kriptografi asimetris dan simetris ini digunakan untuk metode yang berbeda tergantung pada kebutuhan dan konteks. Algoritma yang digunakan pada penelitian ini menggunakan algoritma simetris dikarenakan kebutuhan untuk implementasi pada sistem yang dirancang.

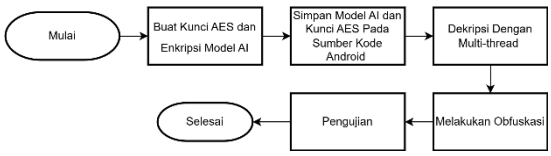
Algoritma kriptografi simetris dapat dikelompokkan menjadi dua kategori, yaitu stream cipher (cipher aliran) dan block cipher (cipher blok). Stream cipher merupakan algoritma kriptografi yang beroperasi dalam bentuk bit tunggal. Algoritma kriptografi simetris kategori block cipher beroperasi dalam bentuk blok bit [8], contoh algoritma simetris seperti AES, DES, Blowfish, Saphent, Skipjack, IDEA, Twofish, RC4 dan lain-lain.

Advanced Encryption Standard (AES) menggantikan *Data Encryption Standard* (DES) karena kecepatan dan kemampuannya yang unggul dalam mengamankan data [9]. AES termasuk dalam kelompok algoritma simetris dengan kategori *block cipher* [8], oleh karena itu AES menggunakan tiga ukuran kunci yaitu 128-bit, 192-bit, dan 256-bit untuk mengenkripsi data [10], beroperasi dengan kunci rahasia yang sama digunakan untuk enkripsi dan dekripsi serta menggunakan transformasi matematis, seperti substitution-permutation network, untuk mengaburkan data [9]. Hal ini menjadikannya metode enkripsi yang banyak digunakan untuk melindungi informasi sensitif dalam komunikasi digital dan data yang tersimpan.

Obfuskasi merupakan sebuah teknik yang digunakan untuk melindungi kode program [11]. Teknik obfuskasi dapat mengubah sumber kode menjadi bentuk yang lebih rumit untuk dipahami dan dianalisis serta dapat melindungi aplikasi dari upaya *disassembling, decompiling,* dan *debugging* [12]. Penerapan obfuskasi menjadi langkah penting untuk menjaga kerahasiaan komponen krusial seperti algoritma dan logika aplikasi agar tidak terekspose.

Metode Penelitian

Metode penelitian ini mencakup beberapa tahap implementasi dan pengujian. Seperti yang dijelaskan pada Gambar 2, terdapat 4 tahap implementasi dan 1 tahap pengujian. Tahap pertama adalah pembuatan kunci AES dan enkripsi model AI. Selanjutnya, model AI dan kunci AES disimpan. Kemudian, proses dekripsi dilakukan menggunakan multi-threading. Setelah semua tahap implementasi selesai, obfuskasi diaktifkan. Adapun pengujian dilakukan dalam tiga tahap, yaitu pengujian model AI, pengujian performa dekripsi algoritma AES pada aplikasi Android, dan pengujian obfuskasi.

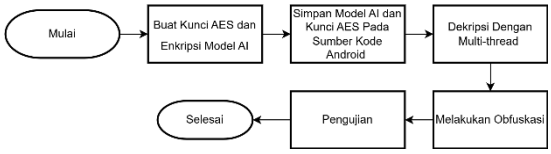


Gambar 2: Tahapan dan pengujian

Buat Kunci AES dan Enkripsi Model AI

Pembuatan kunci AES dibutuhkan untuk penggunaan proses enkripsi maupun dekripsi model AI. Bahasa Python digunakan untuk menghasilkan kunci menggunakan fungsi Token_bytes(32) dari modul Secrets untuk menghasilkan 32 byte atau 256-bit data acak untuk keamanan enkripsi yang tinggi.

Kunci 256-bit tersebut digunakan untuk enkripsi model AI menggunakan algoritma enkripsi AES menjadikannya algoritma dengan AES-256. Seperti gambar 3, sebelum memulai proses enkripsi, model AI perlu dikonversi dalam bentuk biner yang bertujuan untuk memudahkan proses enkripsi. Selanjutnya mulai untuk mengenkripsi data biner tersebut dengan kunci yang sudah dibuat tadi, setelah data berhasil terenkripsi simpan kembali menjadi berkas.



Gambar 3: Flow untuk enkripsi berkas model AI

Simpan Model AI dan Kunci AES Pada Sumber Kode Android

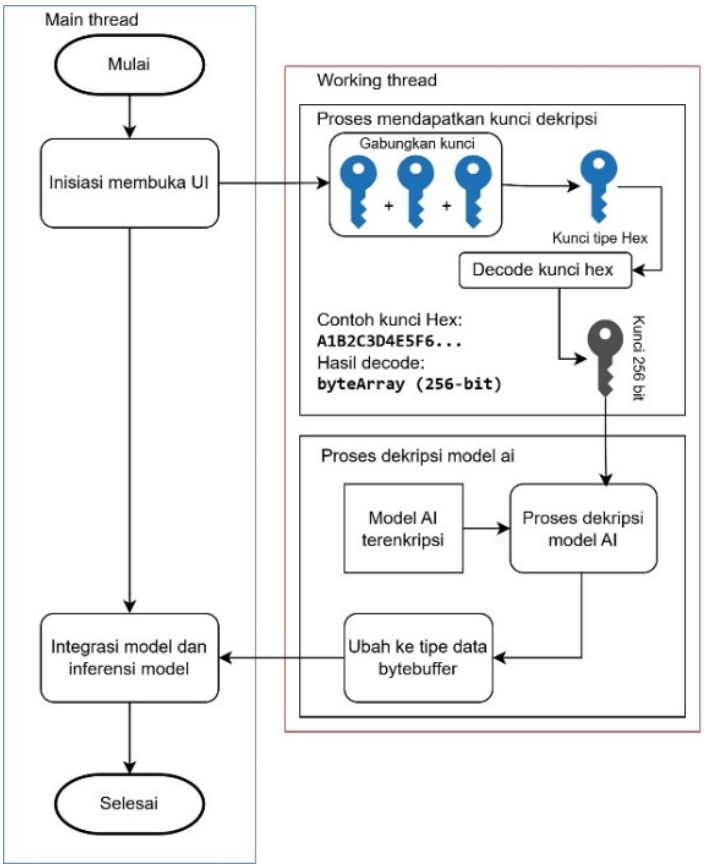
Penyimpanan model AI dan kunci untuk aplikasi offline harus disimpan pada sumber kode Android. Model AI yang disimpan pada folder asset agar model dapat dimuat secara lokal saat aplikasi berjalan sehingga aplikasi dapat berjalan dengan sepenuhnya dalam keadaan offline.

Penyimpanan kunci AES sama seperti halnya menyimpan model, akan disimpan secara lokal di sumber kode Android. Kunci AES bersifat sensitif dan sangat penting untuk proses dekripsi model maka perlunya perlindungan lebih untuk menghindari kebocoran dan eksploitasi kunci. Pada penelitian ini, kunci akan dilakukan pembagian menjadi

3 bagian yang disebar ke 3 berkas Kotlin berbeda. Menyebarkan bagian kunci AES secara langsung di beberapa lokasi akan lebih kompleks dan sulit dilakukan karena kunci AES memiliki tipe data *bytearray*. Pembagian akan dilakukan dengan perubahan tipe data ke bentuk hex string dikarenakan representasi hex string akan memudahkan untuk dipecah, disimpan, dan direkonstruksi kembali, tanpa mengubah integritas dari kunci asli. Penyimpanan kunci untuk dekripsi pada Android akan lebih mudah dan lebih terlindungi.

Dekripsi Dengan *Multi-thread*

Model AI yang terenkripsi dan disimpan pada folder aset Android selanjutnya perlu dilakukan proses dekripsi sebelum dilakukan inferensi untuk menjalankan AI pada perangkat Android, proses dekripsi biasanya membutuhkan sumber daya prosesor yang besar terutama untuk model yang ukuran besar. proses dekripsi yang berat jika dijalankan pada thread yang sama dengan proses UI yaitu main-thread maka frame UI akan terhenti (*UI freeze*). Proses dekripsi akan dilakukan dengan multi-thread seperti yang dilihat pada gambar 4, proses dekripsi model AI akan dilakukan dengan menggunakan metode multi threading proses penggabungan kunci dan proses dekripsi akan dilakukan pada work-thread terpisah dengan main-thread. Metode ini memastikan proses dekripsi akan berjalan pada threads yang berbeda saat proses UI dan dekripsi berjalan sehingga antarmuka pengguna (UI) tetap responsif dan tidak terganggu.



Gambar 4: Flow aplikasi bekerja dengan multi-threading

Sebelum proses dekripsi dimulai perlu untuk menggabungkan kunci AES yang terpisah. Setelah kunci digabungkan berikutnya perlu konversi kunci tersebut ke dalam tipe data yang dapat dikenali dan digunakan oleh algoritma AES yaitu dalam tipe data bytearray. Dikarenakan kunci AES yang disimpan pada sumber kode berbentuk hex string maka, diperlukannya decode untuk mengubah kunci ke bentuk bytearray kembali. Proses menggabungkan dan mendecode kunci berjalan pada work-thread. Setelah kunci yang sesuai berhasil didapatkan, maka proses dekripsi model AI dengan algoritma AES 256-bit bisa dijalankan secara real-time di dalam aplikasi yang sedang berjalan.

Output yang dihasilkan setelah proses dekripsi berbentuk bytearray, tipe data bytearray belum bisa diproses oleh *f*ramework TensorFlow Lite. TensorfFow Lite membutuhkan tipe data dalam bentuk bytebuffer untuk keperluan pemrosesan dan inferensi model. Ouput *p*roses dekripsi perlu untuk diubah dari tipe data bytearray ke dalam tipe data bytebuffer agar dapat dibaca dan diproses oleh TFLite.

Melakukan Obfuskasi

Obfuskasi digunakan untuk mengaburkan kode agar menyulitkan penyerang menganalisis atau memahami struktur dari kode. Proses obfuskasi merupakan bagian paling penting pada penelitian ini, dikarenakan obfuskasi akan melindungi fungsi untuk mendapatkan kunci AES. Obfuskasi dapat membantu model AI yang sudah terenkripsi tetap aman dan mempersulit pihak yang mencoba untuk mendapatkan kunci. Penelitian ini akan menggunakan *compiler* R8 secara default Android Studio untuk melakukan tugas obfuskasi dan optimalisasi kode. R8 tidak hanya mengaburkan kode, tapi juga melakukan desugaring, penyusutan, pengoptimalan, dan dexing [12]. Pengguna compiler R8 perlu mengubah *build version* menjadi versi *Release* serta mengatur *isMinifyEnabled* dan *isShrinkResources* menjadi true pada bagian *buildTypes* di berkas *build.gradle* aplikasi seperti pada gambar 5.

Pengujian

Pengujian berkas model AI terenkripsi dilakukan dengan menghitung entropi pada berkas menggunakan rumus 1 merupakan rumus entropi shannon untuk menilai tingkat keacakan data dan membandingkan sebelum dan sesudah enkripsi.

$$H(x) = -\sum_{i=0}^{255} P(x_i) log_2 P(x_i) \quad (1)$$

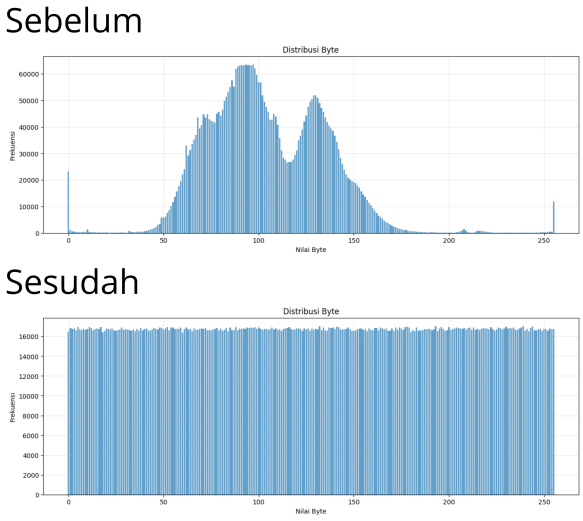
Dengan H(x) adalah entropi dalam satuan bit per byte, ∑adalah penjumlahan, i adalah indeks nilai byte (0-255), P(x_i) adalah probabilitas byte x_i dalam berkas. Entropi diukur dengan nilai pada skala 0-8 semakin tinggi nilai entropi maka semakin acak data tersebut menandakan berkas tersebut terenkripsi [13]. Dengan membandingkan tingkat entropi sebelum dan sesudah enkripsi, sehingga dapat mengevaluasi efektivitas algoritma enkripsi.

Pengujian performa algoritma AES pada perangkat Android dilakukan dengan proses dekripsi model AI yaitu MobileNet V1 TensorFlow Lite dengan ukuran 4.188 MB. Pengujian ini dilakukan dengan mengukur dan membandingkan waktu dekripsi pada 3 perangkat dengan sistem operasi Android dan CPU yang berbeda. Tujuannya untuk mengevaluasi efektivitas penggunaan sumber daya algoritma AES pada berbagai perangkat dengan spesifikasi yang berbeda.

Pengujian obfuskasi dilakukan dengan membandingkan hasil *reverse engineering* sebelum dan sesudah proses menggunakan tools JADX, yang berfungsi untuk mengekstrak sumber kode dari paket aplikasi Android (APK) [14]. Hasil perbandingan dianalisis berdasarkan keterbacaan kode, penamaan variabel atau fungsi, dan struktur kode. Analisis dilakukan untuk mengevaluasi teknik obfuskasi menggunakan R8 dapat mencegah eksposur kode dan kunci AES setelah proses *reverse engineering*.

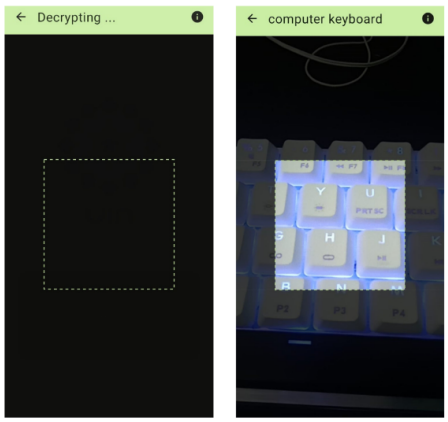
Hasil dan Pembahasan

Hasil pengujian berkas model AI yang dianalisis dengan menghitung entropi dengan rumus shannon menghasilkan nilai pada berkas yang sudah terenkripsi memiliki nilai entropi 8.0000 bit per byte, berbeda dengan nilai berkas sebelum dilakukan enkripsi dengan nilai entropi 6.8372 bit per byte. Dilihat dari gambar 5, distribusi byte pada berkas TFLite sebelum dienkripsi, tidak merata terdapat dua puncak yang menandakan adanya pola dalam berkas. Setelah dienkripsi, distribusi byte menjadi seragam, menunjukkan bahwa data yang telah diproses tidak lagi memiliki pola yang mudah dikenali. Peningkatan nilai entropi sebelum dan sesudah enkripsi memperlihatkan bahwa keacakan pada berkas MobileNet V1 TensorFlow Lite meningkat yang menandakan proses enkripsi menggunakan algoritma AES berhasil dilakukan dengan baik, sehingga berkas tidak dapat diakses atau dimanipulasi tanpa hak akses yang sah.



Gambar 5: Kode mengaktifkan obfuskasi bawaan Android

Pada gambar 6, menunjukkan implementasi model AI MobileNet V1 pada aplikasi Android dengan kombinasi AES dan Obfuskasi. Proses dekripsi model TFLite berjalan secara paralel dengan proses antarmuka pengguna (UI) yaitu membuka kamera. Proses inferensi TensoFlow Lite berlangsung sangat cepat sehingga antarmuka pengguna atau kamera dapat langsung menampilkan hasil deteksi objek secara real-time tanpa ada jeda waktu. Kelebihan aplikasi ini dijalankan tanpa memerlukan koneksi internet.



Gambar 6: Proses dekripsi dan membuka kamera pada aplikasi

Pengujian performa dekripsi menggunakan algoritma AES pada 3 perangkat Android dan dilakukan dengan mengukur waktu ketika proses penggabungan kunci dan proses dekripsi dijalankan. Menghasilkan waktu pada perangkat Redmi 9C dengan spesifikasi Android 10 dan prosesor MediaTek Helio G35 (8-core, 2,3 GHz). Perangkat ini mampu melakukan tugas dekripsi selama 0,7 hingga 1,5 detik. Lalu pada perangkat Samsung Edge S7 dengan spesifikasi Android 8 dan prosesor Exynos Octa 8890 (4x2.3 GHz Mongoose & 4x1.6 GHz Cortex-A53), waktu yang dibutuhkan dekripsi model memerlukan waktu 0.5 hingga 1.2 detik. Pada perangkat Poco C75 dengan Android 14 dengan prosesor Mediatek Helio G81 Ultra octa core(2x2.0 GHz Cortex-A75 & 6x1.8 GHz Cortex-A55) membutuhkan waktu selama 0.5 hingga 1 detik untuk melakukan dekripsi menggunakan algoritma AES.

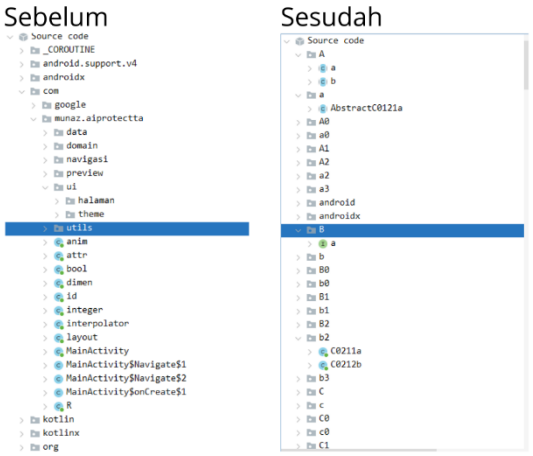
Tabel 1: Hasil pengujian waktu dekripsi pada 3 perangkat

Jenis perangkat	Prosesor	Android	Waktu (detik)
Remi 9C	MediaTek Helio G35 Octa core	10	0.7 – 1.5
Samsung S7 Edge	Exynos 8 Octa 8890	8	0.5 – 1.2
Poco C75	Mediatek Helio G81 Ultra	14	0.5 – 1

Berdasarkan tabel 1 yang menunjukan hasil pengujian, waktu yang didapatkan pada tiga perangkat yang diuji berkisar antara 0,5 detik hingga 1,5 detik menunjukkan bahwa proses dekripsi dilakukan secara efisien dan kecepatan optimal. Proses penggabungan dan decode hex untuk mendapatkan kunci AES 256-bit hanya membutuhkan waktu sekitar 0.002 detik yang tidak mempengaruhi secara signifikan terhadap waktu dekripsi algoritma AES. Algoritma AES mampu menjalankan proses dekripsi dengan baik pada perangkat seluler dengan berbagai spesifikasi tanpa ketergantungan perangkat keras. Model AI MobileNet V1 setelah dilakukan dekripsi berhasil melakukan klasifikasi gambar secara real-time.

Hasil pengujian obfuskasi dengan membandingkan sebelum dan sesudah proses obfuskasi yang dilakukan dengan teknik reverse engineering menggunakan JADX menunjukkan bahwa obfuskasi yang dilakukan dengan R8 pada aplikasi Android berhasil diterapkan, sehingga sumber kode yang dihasilkan menjadi lebih sulit untuk dianalisis dan dipahami. Gambar 7 menunjukkan perbandingan sesudah dan sebelum obfuskasi R8 diimplementasikan, sebelum obfuskasi, kode aplikasi mudah dipahami dengan penamaan variabel dan fungsi yang jelas serta terstruktur. Setelah obfuskasi, kode menjadi sangat sulit dipahami karena adanya perubahan struktur kode, nama kelas, nama variabel yang diubah menjadi tidak bermakna seperti A, B, D, d dan fungsi yang berbeda dari sebelumnya. Obfuskasi sangat efektif dalam mencegah kode untuk sulit dipahami dan mencegah logika untuk mendapatkan kunci dekripsi AES terekspos secara langsung
.

Obfuskasi R8 mempengaruhi ukuran pada berkas APK, ukuran APK sebelum obfuskasi sebesar 86 MB sedangkan setelah dilakukan obfuskasi sebesar 76 MB. Ini merupakan salah satu keunggulan menggunakan obfuskasi yaitu dapat menghemat ukuran dan membuang kode atau library yang tidak digunakan.



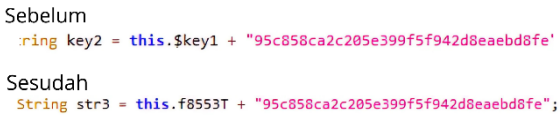
Gambar 7: Sebelum dan sesudah obfuskasi

Obfuskasi R8 memiliki beberapa keterbatasan yang perlu dipertimbangkan seperti tidak melindungi terhadap *aset non-code*, seperti berkas gambar, atau resource lainnya, tidak mengubah penamaan package library pihak ketiga yang terintegrasi dalam aplikasi, dan tidak menyembunyikan nilai dari variabel seperti string atau variabel konstan. Pada gambar 8 dapat dilihat sebelum dan sesudah di obfuskasi *package resources* yang merupakan aset *non-code* tidak terlihat perubahan signifikan, khususnya terlihat pada berkas model AI yaitu (*.tflite*) yang tidak mengalami perubahan dan

tidak terpengaruh dengan obfuskasi. Ini memperlihatkan kekurangan obfuskasi dalam melindungi model AI yang disimpan di dalam aplikasi.

Gambar 8: Resource yang tetap terlihat sesudah dan sebelum obfuskasi

Obfuskasi R8 tidak memiliki kemampuan untuk menyembunyikan nilai dari sebuah variabel yang ditulis secara langsung seperti nilai string atau nilai konstan dalam kode. Pada gambar 9 memperlihatkan perbandingan antara sesudah dan sebelum obfuskasi pada satu bagian kunci dekripsi yang disimpan dalam variabel tipe data string masih terlihat dengan jelas, ini merupakan sebuah kekurangan obfuskasi yang dapat meningkatkan risiko kebocoran kunci.



Gambar 9: bagian kunci hex string terlihat sesudah dan sebelum obfuskasi

Pentingnya melakukan enkripsi model AI dan kombinasi membagi dengan menyebarkan kunci membuat kunci lebih aman, sehingga proses untuk menganalisis, menggabungkan, dan mengubah kunci ke bentuk aslinya menjadi jauh lebih sulit. Kesulitan ini dikarenakan struktur kode, nama variabel, kelas dan fungsi yang berkaitan dengan kunci diacak sehingga memperpanjang waktu dan upaya yang dibutuhkan untuk memecahkan sistem keamanan. Metode ini secara signifikan meningkatkan kompleksitas bagi pihak yang mencoba mengekstrak kunci dari aplikasi.

Penutup

Peningkatan keamanan menggunakan kombinasi AES dan obfuskasi, menghasilkan pengamanan model AI pada aplikasi offline Android yang baik dan efektif. Algoritma AES yang terbukti cepat dan aman terutama saat di implementasi dengan multi-thread sehingga mengalami peningkatan kinerja aplikasi terutama ketika UI dan proses dekripsi berjalan secara paralel sehingga pengguna tidak merasa terganggu dan meningkatnya efisiensi performa aplikasi. Obfuskasi berhasil mempersulit analisis pada sumber kode dengan mengubah struktur kode, walau bagian kunci terlihat dan model AI terlihat ketika dilakukan *reverse engineering* atau *decompiling*. Gabungan enkripsi model AI dan penyebaran kunci di sumber kode, membuat kunci AES untuk melakukan proses dekripsi model AI sulit untuk didapatkan oleh pihak yang tidak berkepentingan atau pihak yang tidak memiliki hak akses. Dengan kombinasi kriptografi algoritma AES, pembagian kunci menggunakan multithreading, dan obfuskasi dapat menjadi solusi untuk pengamanan model AI pada aplikasi *Offline* pada sistem operasi Android.

Saran penelitian selanjutnya diperlukan metode pengamanan kunci atau pengamanan aplikasi yang perlu ditingkatkan lebih baik dikarenakan adanya kemungkinan kunci bocor walau sudah dilakukan obfuskasi menggunakan R8. Penelitian berikutnya dapat menggunakan obfuskasi pihak ketiga yang memiliki fitur lebih baik. Perlunya metode pengamanan untuk model AI yang berukuran besar dikarenakan akan menyebabkan proses dekripsi yang lama dan beban besar pada perangkat Android sehingga mengganggu kenyamanan pengguna. Perlunya solusi yang efektif dan cepat untuk melakukan dekripsi untuk model yang besar.

Daftar Pustaka

[1] Sebastian P. Bayer, Tommaso Frassetto, Patrick Jauernig and Korbinian Riedhammer, "Offline model guard: Secure and private ML on mobile devices," in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, pp. 460–465, DOI:10.23919/DATE48585.2020.9116560, 2020.

[2] Y. Deng, "Deep learning on mobile devices: a review," Mobile Multimedia/Image Processing, Security, and Applications 2019, vol. 10993, pp. 52–66, 2019.

[3] A. Verma, "Encryption and real time decryption for protecting machine learning models in android applications," arXiv preprint arXiv:2109.02270, 2021.

[4] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in The World Wide Web Conference, pp. 2125–2136, 2019.

[5] Z. Deng, K. Chen, G. Meng, X. Zhang, K. Xu, and Y. Cheng, "Understanding Real-world Threats to Deep Learning Models in Android Apps," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, in CCS '22. New York, NY, USA: Association for

Computing Machinery, pp. 785–799. doi: 10.1145/3548606.3559388, 2022.

[6] F. N. Pabokory, I. F. Astuti, dan A. H. Kridalaksana, "Implementasi Kriptografi Pengamanan Data Pada Pesan Teks, Isi File Dokumen, Dan File Dokumen Menggunakan Algoritma Advanced Encryption Standard," Informatika Mulawarman: Jurnal Ilmiah Ilmu Komputer, vol. 10, no. 1, pp. 20–31, 2016.

[7] N. B. N. Putra, F. A. Raihana, W. M. A. Mondong, dan A. R. Kardian, "Analisis Enkripsi Kriptografi Asimetris Algoritma RSA Berbasis Pemrograman Batch pada Media Flashdisk," Jurasik (Jurnal Riset Sistem Informasi dan Teknik Informatika), vol. 8, no. 1, pp. 142–154, 2023.

[8] D. A. Meko, "Perbandingan Algoritma DES, AES, IDEA Dan Blowfish dalam Enkripsi dan Dekripsi Data," Jurnal Teknologi Terpadu (JTT), vol. 4, no. 1, 2018.

[9] M. P and M. Samreetha, "A Review of Encryption and Decryption of Text Using the AES Algorithm," International Journal of Scientific Research and Engineering Trends, vol. 10, pp. 400–404, Apr. 2024.

[10] N. W. Hidayatulloh, M. Tahir, H. Amalia, N. A. Basyar, A. F. Prianggara, and M. Yasin, "Mengenal Advance Encryption Standard (AES) sebagai Algoritma Kriptografi dalam Mengamankan Data," Digital Transformation Technology, vol. 3, no. 1, pp. 1–10, 2023.

[11] Shuaike Dong, Menghao Li, Wenrui Diao, and Liu Xiangyu, "Understanding android obfuscation techniques: A large-scale investigation in the wild," in Security and privacy in communication networks: 14th international conference, secureComm 2018, Singapore, Singapore, August 8-10, 2018, proceedings, part i, Springer, pp. 172–192, 2018.

[12] G. You, G. Kim, S. J. Cho, and H. Han, "A comparative study on optimization, obfuscation, and deobfuscation tools in android," Journal of Internet Services and Information Security, vol. 11, no. 1, pp. 2–15, doi: 10.22667/JISIS.2021.02.28.002, 2021.

[13] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Comparison of Entropy Calculation Methods for Ransomware Encrypted File Identification," Entropy, vol. 24, no. 10, doi: 10.3390/e24101503, 2022.

[14] L. Luo, Felix Pauck, Goran Piskachev, and Manuel Benz, "TaintBench: Automatic real-world malware benchmarking of Android taint analyses," Empir Softw Eng, vol. 27, no. 1, p. 16, doi: 10.1007/s10664-021-10013-5, 2021.