Implementasi Carter-Wegman MAC sebagai *Cryptographic Checksum* untuk Mencegah Penipuan Transaksi *E-commerce* secara *Real-time*

Michael Leon Putra Widhi - 13521108[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521108@std.stei.itb.ac.id

***Abstrak*— Keamanan transaksi *e-commerce* menjadi prioritas utama dalam menjaga integritas dan kepercayaan konsumen dalam ekosistem digital. Metode autentikasi pesan konvensional seperti *Hash-based Message Authentication Code* (HMAC) meskipun banyak digunakan, memiliki keterbatasan dalam efisiensi komputasi dan ketahanan terhadap manipulasi data dalam skala besar. Pada tulisan ini dikaji terkait implementasi Carter-Wegman *Message Authentication Code* (MAC) sebagai *cryptographic chekcsum* yang lebih efisien dan kuat dalam konteks transaksi *e-commerce* secara *real-time*. Carter-Wegman MAC dibuat dengan memanfaatkan kombinasi fungsi *hash* polinomial dan enkripsi AES dalam mode Galois/Counter Mode (GCM) untuk menyediakan mekanisme autentikasi yang lebih cepat dan tahan manipulasi. Implementasi ini bertujuan untuk mendeteksi dan mencegah penipuan transaksi dengan memastikan integritas data melalui verifikasi yang cepat dan akurat. Hasil pengujian pada berbagai skenario, termasuk perubahan pesan dan modifikasi MAC oleh pihak yang tidak berwenang, menunjukkan bahwa metode ini secara efektif dapat mendeteksi manipulasi data dan memberikan perlindungan yang lebih baik dibandingkan dengan HMAC. Selain itu, Carter-Wegman MAC menunjukkan efisiensi komputasi yang lebih tinggi, menjadikannya solusi yang cocok untuk kebutuhan *e-commerce* modern yang mengharuskan proses transaksi yang cepat dan aman. Temuan ini memberikan kontribusi penting bagi pengembangan strategi keamanan transaksi yang lebih andal dalam industri *e-commerce*.**

***Kata Kunci*— MAC, keamanan data, transaksi daring, e-commerce, checksum, kriptografi**

I. PENDAHULUAN

Transaksi *e-commerce* saat ini sangat rentan terhadap berbagai ancaman keamanan, termasuk manipulasi data, penyusupan pihak ketiga, dan penipuan transaksi. Perlindungan integritas data menjadi semakin penting seiring dengan meningkatnya volume transaksi dan kompleksitas jaringan digital. Metode autentikasi pesan seperti *Hash-based Message Authentication Code* (HMAC) telah menjadi standar dalam industri untuk memastikan data tidak dimodifikasi oleh pihak yang tidak berwenang. Namun, HMAC memiliki keterbatasan dalam hal efisiensi komputasi dan fleksibilitas, terutama ketika menangani transaksi dalam jumlah besar atau berkecepatan tinggi. Kebutuhan akan metode yang lebih efisien dan aman memotivasi eksplorasi pendekatan alternatif yang dapat memenuhi tuntutan modern ini.

Carter-Wegman *Message Authentication Code* (MAC) muncul sebagai solusi potensial yang menggabungkan efisiensi dari fungsi *hash* polinomial dengan kekuatan enkripsi AES. Pendekatan ini menawarkan keuntungan dalam hal kecepatan perhitungan dan keandalan deteksi perubahan data. Carter-Wegman MAC menggunakan fungsi *hash* polinomial untuk menghasilkan nilai *hash* dari pesan yang kemudian dienkripsi menggunakan AES dalam mode Galois/Counter Mode (GCM). Mode GCM tidak hanya menyediakan enkripsi data tetapi juga memastikan autentikasi pesan yang kuat. Implementasi ini mengatasi keterbatasan HMAC dengan memberikan efisiensi yang lebih tinggi dan kemampuan untuk menangani transaksi *real-time* secara lebih efektif.

Penelitian ini bertujuan untuk mengimplementasikan Carter-Wegman MAC dalam konteks transaksi *e-commerce* dan mengevaluasi keefektifannya dalam mencegah penipuan dan menjaga integritas data. Melalui serangkaian pengujian pada berbagai skenario, dilakukan evaluasi terkait kinerja metode ini dalam mendeteksi modifikasi data dan membandingkannya dengan metode konvensional. Hasil penelitian diharapkan memberikan wawasan baru tentang bagaimana Carter-Wegman MAC dapat diadopsi sebagai *checksums* kriptografis yang lebih efektif dan efisien untuk lingkungan *e-commerce* yang terus berkembang.

.

II. TEORI DASAR

A. Transaksi Daring

Transaksi daring, atau transaksi elektronik, merujuk pada pertukaran barang, jasa, atau informasi yang dilakukan melalui internet. Dalam era digital saat ini, transaksi daring telah menjadi bagian integral dari kehidupan sehari-hari, mencakup berbagai kegiatan seperti pembelian produk dari *e-commerce*, pembayaran tagihan, transfer dana antar rekening, dan banyak lagi. Dengan kemudahan akses dan kecepatan transaksi yang ditawarkan, transaksi daring memungkinkan konsumen dan bisnis untuk berinteraksi dan melakukan pertukaran dengan efisien tanpa keterbatasan geografis. Nilai perputaran uang pada lingkungan *e-commerce* global diperkirakan mencapai lebih dari 5 triliun USD pada tahun 2022 [1], mencerminkan pertumbuhan signifikan dalam adopsi transaksi daring.

Transaksi daring melibatkan beberapa komponen penting untuk memastikan keamanannya, termasuk autentikasi, otorisasi, dan enkripsi data [2]. Autentikasi memastikan bahwa hanya pengguna yang sah yang dapat mengakses akun atau melakukan transaksi, sedangkan otorisasi mengatur hak akses pengguna berdasarkan kredensial mereka. Enkripsi, di sisi lain, berfungsi untuk melindungi data selama transmisi dari penyadapan atau modifikasi oleh pihak yang tidak berwenang. Protokol keamanan seperti SSL/TLS digunakan secara luas untuk menjaga kerahasiaan dan integritas data selama transaksi daring. Dengan adopsi teknologi ini, transaksi daring tidak hanya menawarkan kenyamanan tetapi juga berusaha memberikan tingkat keamanan yang tinggi untuk melindungi pengguna dan bisnis dari risiko dan ancaman siber.

B. Penipuan Transaksi Daring

Penipuan transaksi daring adalah tindakan ilegal yang bertujuan untuk mencuri uang atau informasi pribadi dari individu atau organisasi melalui manipulasi atau eksploitasi sistem transaksi daring. Penipuan ini dapat terjadi dalam berbagai bentuk, seperti pencurian identitas, *phishing*, *card-not-present* (CNP) *fraud*, dan serangan *man-in-the-middle* (MITM). Pencurian identitas, misalnya, melibatkan penggunaan informasi pribadi yang dicuri untuk melakukan transaksi atas nama korban, sedangkan phishing melibatkan upaya penipu untuk memperoleh informasi sensitif dengan menyamar sebagai entitas yang sah. Menurut laporan dari Javelin Strategy & Research [3], kasus penipuan identitas daring menyebabkan kerugian finansial hingga miliaran dolar setiap tahunnya.

Salah satu metode umum yang digunakan dalam penipuan transaksi daring adalah serangan *card-not-present*, di mana penipu menggunakan informasi kartu kredit yang dicuri untuk melakukan pembelian tanpa memerlukan kartu fisik. Metode ini semakin sering terjadi seiring dengan meningkatnya volume transaksi *e-commerce*. Selain itu, serangan *man-in-the-middle* memungkinkan penipu untuk menyadap dan memodifikasi komunikasi antara dua pihak, sering kali tanpa sepengetahuan mereka, untuk mencuri informasi atau mengalihkan dana. Ancaman ini menunjukkan pentingnya penerapan langkah-langkah keamanan yang kuat, seperti enkripsi data, autentikasi multi-faktor, dan pemantauan aktivitas transaksi secara *real-time* untuk mendeteksi dan mencegah aktivitas penipuan. Dengan meningkatnya kompleksitas dan frekuensi serangan, perlindungan terhadap penipuan transaksi daring menjadi semakin penting bagi perusahaan *e-commerce* dan penyedia layanan pembayaran.

C. Kriptografi

Kriptografi adalah ilmu dan seni untuk melindungi informasi dengan mengubahnya menjadi format yang tidak dapat dibaca tanpa proses dekripsi yang tepat. Tujuan utama kriptografi adalah untuk memastikan kerahasiaan, integritas, dan autentikasi data dalam komunikasi dan penyimpanan. Kriptografi modern menggunakan berbagai algoritma matematika untuk mengenkripsi dan mendekripsi data, membuat informasi hanya dapat diakses oleh pihak yang berwenang. Misalnya, enkripsi simetris menggunakan kunci yang sama untuk enkripsi dan dekripsi, sedangkan enkripsi asimetris menggunakan pasangan kunci publik dan kunci pribadi. Menurut buku "*Cryptography and Network Security*" oleh William Stallings [4], perkembangan kriptografi telah menjadi fondasi utama dalam menjaga keamanan informasi dalam berbagai aplikasi, mulai dari komunikasi militer hingga transaksi *e-commerce*.

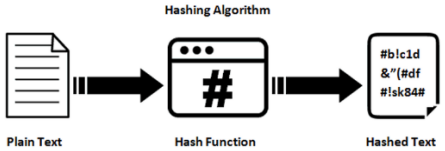
Kriptografi memiliki beberapa konsep dasar, termasuk enkripsi, dekripsi, kunci kriptografi, dan algoritma kriptografi. Enkripsi adalah proses mengubah teks asli (*plaintext*) menjadi teks tersandi (*ciphertext*) menggunakan algoritma dan kunci tertentu. Dekripsi adalah proses sebaliknya, mengubah *ciphertext* kembali menjadi *plaintext* menggunakan kunci yang sesuai. Kunci kriptografi adalah sekumpulan bit yang digunakan dalam algoritma untuk mengubah data dari satu bentuk ke bentuk lain. Algoritma kriptografi adalah prosedur matematis yang digunakan untuk enkripsi dan dekripsi. Dua kategori utama dari algoritma kriptografi adalah enkripsi simetris dan asimetris. Kriptografi simetris, seperti *Advanced Encryption Standard* (AES), cepat dan efisien untuk mengenkripsi data dalam jumlah besar, sedangkan kriptografi asimetris, seperti RSA (Rivest-Shamir-Adleman), digunakan untuk utamanya untuk pertukaran kunci dan *digital signature* karena tingkat keamanan yang lebih tinggi tetapi dengan kecepatan yang lebih rendah.

D. Fungsi Hash

Fungsi *hash* merupakan suatu fungsi matematika yang mampu mengompresi data masukan dengan ukuran yang tidak terbatas menjadi *string* dengan panjang yang telah ditentukan [5]. Hasil dari fungsi *hash* ini disebut sebagai *message-digest* atau *hash-value*. Fungsi *hash* memegang peranan penting dalam dunia kriptografi karena banyak dimanfaatkan untuk berbagai kebutuhan keamanan, seperti otentikasi, tanda tangan digital, dan pembangkit bilangan acak.

Secara umum, terdapat dua jenis fungsi *hash*, yaitu:

1. *Keyed Hash Function*
   Fungsi *hash* yang menggunakan kunci rahasia untuk melakukan kompresi.
2. *Un-keyed Hash Function*
   Fungsi *hash* yang tidak memerlukan kunci rahasia untuk melakukan kompresi.



Gambar 2.1. *Ilustrasi fungsi hash*
Sumber: https://www.researchgate.net/figure/Shows-how-Hash-Functions-work-15_fig5_371119558

Fungsi *hash* memiliki beberapa sifat unik yang membedakannya dari algoritma enkripsi dan dekripsi konvensional. Berikut beberapa sifat penting fungsi *hash* [6]:

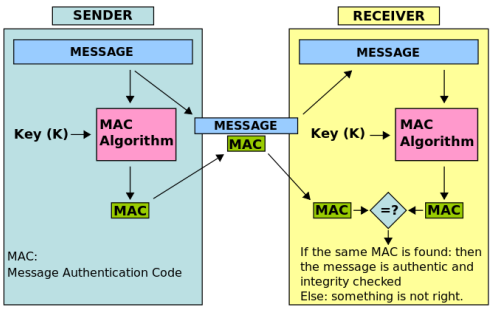
1. Satu Arah (*One-Way*)
   Fungsi *hash* bekerja seperti pemetaan data ke nilai yang lebih ringkas, dan tidak memungkinkan proses sebaliknya, yaitu mengubah nilai *hash* kembali menjadi data asli. Sifat ini sangat penting untuk keamanan data, karena meskipun seseorang mengetahui nilai *hash*, mereka tidak dapat merekonstruksi data awal.
2. Ketahanan terhadap Kolisi (*Collision Resistance*)
   Fungsi *hash* yang baik memiliki kemungkinan sangat kecil untuk menghasilkan nilai *hash* yang sama (kolisi) dari dua *input* data yang berbeda. Hal ini penting untuk menjaga integritas data dan mencegah penipuan.
3. Ketahanan terhadap *Preimage* (*Preimage Resistance*)
   Sangat sulit untuk menemukan data asli dari nilai *hash* yang diberikan. Sifat ini penting untuk melindungi data rahasia, karena meskipun seseorang memiliki nilai *hash*, mereka tidak dapat dengan mudah menemukan data yang terkait dengannya.
4. Ketahanan terhadap *Second Preimage* (*Second Preimage Resistance*)
   Untuk nilai *hash* tertentu, sangat sulit untuk menemukan data lain yang menghasilkan nilai *hash* yang sama. Sifat ini memperkuat ketahanan terhadap *preimage* dan mencegah penipuan di mana seseorang mencoba mengganti data asli dengan data lain yang memiliki nilai *hash* yang sama.

Fungsi *hash* memiliki peran penting dalam menjaga keamanan data, dengan beberapa manfaat utama:

1. Menjaga Integritas Data
   Fungsi *hash* sangat sensitif terhadap perubahan *input*. Perubahan sekecil apapun pada data akan menghasilkan nilai *hash* yang berbeda. Hal ini memungkinkan verifikasi integritas data dengan membandingkan nilai *hash* yang tersimpan dengan nilai *hash* yang dihitung dari data saat ini.
2. Memudahkan Verifikasi
   Nilai *hash* dapat digunakan untuk memverifikasi keaslian pesan atau informasi yang dibagikan antar pihak. Alih-alih membandingkan konten pesan secara langsung, nilai *hash* dapat dibandingkan untuk memastikan pesan tersebut benar dan tidak dimodifikasi.
3. Normalisasi Panjang Data
   Fungsi *hash* dapat menormalisasi data dengan panjang yang berbeda menjadi nilai *hash* dengan panjang yang sama. Hal ini memudahkan penyimpanan data dalam basis data dan memungkinkan perbandingan data yang lebih efisien.

E. Message Authentication Code (MAC)

Message Authentication Code (MAC) adalah kode unik yang dikirimkan bersamaan dengan pesan untuk memastikan keaslian dan integritas pesan. MAC dapat dianalogikan sebagai sidik jari digital untuk pesan. Perbedaan MAC dengan nilai *hash* biasa adalah MAC menggunakan fungsi *hash* tipe *keyed hash function*, yang membutuhkan kunci rahasia (*secret key*) untuk melakukan perhitungan. MAC sering digunakan pada kasus ketika terdapat dua atau lebih pihak yang ingin berkomunikasi lewat media yang tidak aman [7].



Gambar 2.2. *Mekanisme penggunaan MAC*
Sumber: https://upload.wikimedia.org/wikipedia/commons/thumb/0/08/MAC.svg/661px-MAC.svg.png

Fungsi *hash* biasa memiliki kelemahan yang tidak dimiliki MAC, yaitu tidak aman terhadap perubahan pesan. Seorang penyerang dapat mengubah pesan dan menghitung ulang nilai *hash* nya, sehingga penerima pesan tidak dapat mendeteksi perubahan tersebut. Sedangkan MAC, dengan menggunakan kunci rahasia (*secret key*), berhasil untuk mengatasi kelemahan ini. Kunci rahasia hanya dimiliki oleh pihak-pihak yang saling percaya, sehingga penyerang tidak dapat menghitung nilai MAC yang benar tanpa mengetahui kunci rahasia tersebut.

Selain menggunakan fungsi *hash*, MAC juga dapat diimplementasikan mengggunakan algoritma *block cipher* seperti AES, DES [7]. Caranya adalah dengan menggunakan hanya sebagian hasil enkripsi saja. Misalkan suatu algoritma *block cipher* memiliki panjang blok sebesar x bit. Untuk mendapatkan nilai MAC dengan panjang x bit, dapat dilakukan enkripsi terhadap seluruh pesan dan diambil hasil enkripsi pada blok terakhir saja. MAC biasanya didapatkan dengan menerapkan menggunakan persamaan berikut:

$$MAC_K(m) = H(m + k)$$

Dengan H adalah suatu fungsi hash tertentu (MD5, SHA-256, SHA-512, dan lain-lain), m adalah isi pesan asli, s adalah *secret key* yang digunakan, dan + merupakan operasi *append*.

Meskipun MAC menawarkan otentikasi yang lebih aman, MAC sendiri memiliki kelemahan terhadap *length extension attack* [8]. Penyerang dapat memanfaatkan kelemahan ini untuk menghitung MAC dari pesan lain (m') tanpa mengetahui isi pesan aslinya (m).

F. Carter-Wegman Message Authentication Code (MAC)

Carter-Wegman *Message Authentication Code* (MAC) adalah metode autentikasi kriptografi yang menggabungkan konsep *hashing* universal dan enkripsi untuk menghasilkan kode autentikasi pesan yang aman. Metode ini dikembangkan oleh Larry Carter dan Mark Wegman pada tahun 1981, dan dikenal karena efisiensi dan kekuatannya

dalam memastikan integritas dan autentikasi pesan [9]. Prinsip dasar dari Carter-Wegman MAC adalah menggunakan fungsi *hash* universal untuk memetakan pesan ke nilai *hash* yang unik dengan probabilitas kolisi yang sangat rendah, kemudian mengenkripsi nilai *hash* tersebut menggunakan kunci rahasia untuk menghasilkan MAC. Pendekatan ini menawarkan perlindungan yang kuat terhadap pemalsuan pesan karena penyerang harus mengetahui kunci rahasia dan fungsi *hash* yang digunakan untuk menghasilkan MAC yang valid.

Keunggulan utama dari Carter-Wegman MAC dibandingkan dengan metode MAC tradisional, seperti HMAC, adalah efisiensinya dalam hal komputasi. Carter-Wegman MAC memiliki *overhead* komputasi yang rendah karena proses *hashing* yang cepat dan ringan. Selain itu, metode ini juga menawarkan keamanan yang kuat dengan kombinasi *hashing* universal dan enkripsi kunci simetris, yang membuatnya sulit untuk ditembus oleh serangan kriptanalisis konvensional. Menurut "Introduction to Modern Cryptography" oleh Jonathan Katz dan Yehuda Lindell [10], metode Carter-Wegman secara teoretis memberikan keamanan yang lebih tinggi dengan asumsi *hash* universal dan enkripsi yang kuat, menjadikannya pilihan yang menarik untuk autentikasi pesan dalam berbagai aplikasi keamanan siber.

III. ANALISIS PERMASALAHAN DAN DESAIN

A. Universal Hash Family

Universal hash family adalah kumpulan fungsi *hash* yang dirancang untuk meminimalkan probabilitas kolisi antara pesan yang berbeda. Fungsi *hash universal* pertama kali diperkenalkan oleh Carter dan Wegman pada tahun 1979 [9]. Fungsi-fungsi ini memiliki sifat bahwa, untuk setiap pasangan input yang berbeda, probabilitas bahwa dua input tersebut menghasilkan output yang sama (kolisi) sangat kecil.

Dalam notasi formal, sebuah keluarga fungsi *hash* H disebut universal jika, untuk setiap $x \neq y$, probabilitas

$$\Pr\left[h(x) = h(y)\right] \leq \frac{1}{|R|}$$

dengan h dipilih secara acak dari H dan R adalah rentang nilai *hash*. Persamaan ini juga dapat dituliskan dengan notasi berikut. Sebuah keluarga fungsi $H = \{h: U \rightarrow [m]\}$ disebut *universal hash family* jika,

$$\forall x, y \in U, x \neq y : \left|\{h \in H : h(x) = h(y)\}\right| \leq \frac{|H|}{m}$$

Dengan kata lain, peluang atau nilai dari dua kunci berbeda mana pun paling mungkin bertabrakan adalah $1/m$ saat fungsi *hash* h diambil secara seragam dan acak dari H. Nilai ini adalah peluang kolisi yang kita harapkan jika fungsi *hash* menetapkan kode *hash* yang benar-benar acak ke setiap kunci.

Dalam konteks Carter-Wegman MAC, *universal hash family* digunakan untuk memetakan pesan ke nilai *hash* sebelum dienkripsi dengan kunci rahasia. Proses ini dimulai dengan memilih fungsi *hash* h dari keluarga *hash* universal H secara acak, dan kemudian menghitung nilai *hash* dari pesan x sebagai $h(x)$. Dengan menggunakan fungsi *hash* universal, probabilitas bahwa dua pesan berbeda menghasilkan nilai *hash* yang sama tetap rendah, bahkan jika penyerang dapat memilih pesan yang akan di-*hash*. Efisiensi komputasi dari fungsi *hash* ini juga menjadikannya ideal untuk aplikasi dengan kebutuhan pemrosesan data yang tinggi, seperti transaksi *e-commerce*. Selain itu, keunggulan *universal hash* dalam mencegah tabrakan memberikan tingkat keamanan tambahan ketika digunakan dalam skema *Message Authentication Code* (MAC), menjadikannya komponen penting dalam desain protokol keamanan modern.

B. Pembangkitan Bilangan Acak

Pembangkitan bilangan acak adalah proses menghasilkan angka atau bit yang tidak dapat diprediksi dan tidak mengikuti pola tertentu. Bilangan acak sangat penting dalam kriptografi karena mereka digunakan untuk menghasilkan kunci, inisialisasi vektor, dan elemen acak lainnya yang krusial untuk keamanan. Ada dua jenis utama pembangkitan bilangan acak: *True Random Number Generators (TRNGs)*, yang menggunakan sumber fisik (seperti noise elektrik) untuk menghasilkan bilangan acak, dan *Pseudorandom Number Generators (PRNGs)*, yang menggunakan algoritma deterministik untuk menghasilkan urutan bilangan yang tampak acak. TRNG umumnya dianggap lebih aman karena bilangan yang dihasilkannya benar-benar acak dan tidak dapat diprediksi, sementara PRNG lebih efisien dan sering digunakan dalam aplikasi yang membutuhkan banyak bilangan acak.

Dalam konteks Carter-Wegman MAC, pembangkitan bilangan acak memiliki peran penting dalam memastikan keamanan algoritma. Carter-Wegman MAC menggunakan kunci rahasia yang harus dihasilkan secara acak untuk menjaga keamanan autentikasi pesan. Misalnya, untuk menghasilkan kunci *k* untuk fungsi *hash* universal dan kunci enkripsi K, pembangkitan bilangan acak yang aman dan berkualitas tinggi sangat diperlukan. Bilangan acak yang dihasilkan harus memiliki entropi yang tinggi untuk mencegah penyerang dari memprediksi atau mereproduksi kunci yang digunakan dalam proses *hashing* dan enkripsi.

Bilangan acak juga digunakan untuk memilih fungsi *hash* h dari keluarga *hash* universal H. Dengan memilih h secara acak, probabilitas tabrakan antara dua pesan yang berbeda menjadi sangat kecil, sehingga memperkuat keamanan skema MAC. Tanpa bilangan acak yang kuat, keamanan keseluruhan dari Carter-Wegman MAC dapat terkompromi karena penyerang mungkin dapat menebak atau menemukan pola dalam kunci atau fungsi *hash* yang digunakan. Oleh karena itu, penggunaan pembangkit bilangan acak yang aman dan andal adalah salah satu pilar utama dalam implementasi Carter-Wegman MAC untuk memastikan bahwa pesan tetap terlindungi dari modifikasi dan pemalsuan.

C. Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) adalah algoritma enkripsi simetris yang diadopsi sebagai standar enkripsi oleh National Institute of Standards and Technology (NIST) pada tahun 2001 [11]. AES dirancang oleh Joan Daemen dan Vincent Rijmen dan dikenal juga sebagai Rijndael. Algoritma ini menggunakan blok ukuran tetap 128-bit dan mendukung panjang kunci 128, 192, dan 256 bit. AES dianggap sangat aman dan telah diuji secara luas oleh komunitas kriptografi global, menjadikannya standar *de facto* untuk enkripsi data di berbagai aplikasi, mulai dari keamanan data pribadi hingga komunikasi militer. Salah satu keunggulan AES adalah efisiensi komputasinya, yang memungkinkan implementasi cepat baik dalam perangkat keras maupun perangkat lunak.

AES bekerja melalui serangkaian transformasi yang diulang dalam beberapa putaran, tergantung pada panjang kunci yang digunakan. Proses enkripsi AES meliputi substitusi byte (SubBytes), pergeseran baris (ShiftRows), pencampuran kolom (MixColumns), dan penambahan kunci (AddRoundKey) [12]. Dekripsi adalah kebalikan dari proses ini, menggunakan transformasi yang sesuai untuk mengembalikan data terenkripsi ke bentuk aslinya. Keamanan AES didasarkan pada kompleksitas matematis dari transformasi yang digunakan, yang membuatnya tahan terhadap berbagai serangan kriptanalisis yang dikenal.

D. Realitas Transaksi E-commerce

Transaksi daring dalam *e-commerce* melibatkan berbagai aktivitas, mulai dari pemilihan produk hingga pembayaran. Pembelanjaan *online* memungkinkan konsumen untuk membeli barang dan jasa melalui platform digital seperti situs *web* dan aplikasi *mobile*. Proses ini mencakup beberapa tahap penting: *browsing* produk, menambahkannya ke keranjang belanja, memasukkan informasi pengiriman, dan melakukan pembayaran. Data yang dikirimkan selama transaksi ini mencakup informasi pribadi seperti nama, alamat, nomor telepon, serta informasi sensitif seperti detail kartu kredit atau metode pembayaran lainnya.

Sebagai bagian dari proses pembelanjaan *online*, sejumlah data dikirimkan dan disimpan sebagai bukti transaksi. Data ini termasuk:

1. Informasi Pelanggan: Nama, alamat pengiriman, email, dan nomor telepon.
2. Detail Pembayaran: Nomor kartu kredit/debit, tanggal kedaluwarsa, CVV, atau informasi akun pembayaran elektronik.
3. Detail Transaksi: Nomor pesanan, daftar barang yang dibeli, harga masing-masing barang, total biaya, pajak, dan biaya pengiriman.
4. Data Pengiriman: Informasi kurir, nomor pelacakan, dan status pengiriman.

Vulnerabilitas dalam transaksi daring dapat timbul dari berbagai sumber, termasuk kelemahan dalam aplikasi *e-commerce*, kesalahan konfigurasi, atau serangan eksternal. Beberapa vulnerabilitas umum adalah:

1. Injeksi SQL: Penyerang dapat memasukkan kode SQL berbahaya melalui formulir input untuk mendapatkan akses tidak sah ke basis data.
2. *Cross-Site Scripting* (XSS): Penyerang dapat menyuntikkan skrip berbahaya ke dalam halaman *web* yang kemudian dijalankan oleh browser pengguna.
3. *Man-in-the-Middle* (MITM) *Attacks*: Penyerang dapat menyadap dan memodifikasi komunikasi antara pengguna dan server tanpa diketahui.
4. *Phishing*: Upaya penipuan di mana penyerang menyamar sebagai entitas yang sah untuk mencuri informasi sensitif dari pengguna.

Ancaman keamanan dalam e-commerce dapat mengakibatkan kerugian finansial dan kerusakan reputasi. Beberapa ancaman utama meliputi pencurian identitas, *fraudulent transactions* (transaksi palsu yang dilakukan dengan menggunakan informasi kartu kredit yang dicuri), *data breaches* (peretas mendapatkan akses ke basis data yang berisi informasi sensitif pelanggan), dan *Distributed Denial of Service (DDoS) Attack*, sebuah serangan yang mengganggu layanan *e-commerce* dengan membanjiri server dengan lalu lintas yang berlebihan.

Untuk mengatasi ancaman dan memastikan keamanan transaksi daring, aplikasi *e-commerce* menggunakan berbagai algoritma dan teknik kriptografi:

1. *Transport Layer Security* (TLS)
   Mengamankan komunikasi antara pengguna dan *server* dengan mengenkripsi data yang ditransmisikan.
2. *Secure Hash Algorithm* (SHA-256): Digunakan untuk *hashing* data penting dan memastikan integritasnya.
3. RSA: Algoritma enkripsi asimetris yang digunakan untuk pertukaran kunci dan *digital signatures*.
4. *Advanced Encryption Standard* (AES): Digunakan untuk enkripsi data sensitif selama penyimpanan dan transmisi.
5. HMAC (*Hash-based Message Authentication Code*): Digunakan untuk memverifikasi integritas dan autentikasi pesan.

Dengan implementasi teknologi dan algoritma ini, aplikasi *e-commerce* berusaha untuk melindungi data pengguna dan memastikan transaksi yang aman dan terpercaya.

E. Hash-based Message Authentication Code (HMAC)

HMAC menawarkan beberapa kelebihan dibandingkan MAC lain, terutama dalam hal keamanan dan fleksibilitas:

1. Kemudahan Modifikasi Fungsi *Hash*
   Fungsi *hash* yang digunakan dalam HMAC bersifat "*black box*", artinya dapat diubah dengan mudah sesuai kebutuhan. Hal ini memungkinkan penggunaan fungsi *hash* yang lebih baru dan lebih kuat di masa depan.
2. Kekuatan Kriptografi yang Terbukti
   Kelebihan utama HMAC terletak pada kekuatan kriptogramatiknya. Jika HMAC terbukti tidak aman, maka dipastikan kelemahannya terletak pada fungsi

hash yang digunakan, bukan pada mekanisme pembuatan MAC. Hal ini mempermudah proses identifikasi dan perbaikan kelemahan.

3. Perlindungan Terhadap Serangan UFCMA

   Baik MAC maupun HMAC melindungi pesan dari serangan "*unforgeable under chosen-message attacks*" (UFCMA). Hal ini berarti, meskipun penjahat dapat menyadap pesan yang dipertukarkan antara dua pihak, mereka tidak dapat membuat MAC yang valid untuk pesan tersebut, bahkan jika mereka mengetahui isi pesannya.

4. Fitur Keamanan Tambahan: *Pseudorandom Function* (PRF)

   HMAC memiliki fitur keamanan tambahan, yaitu *pseudorandom function* (PRF). Artinya, jika penjahat tidak memiliki kunci rahasia yang dimiliki kedua pihak yang berkomunikasi, semua pesan yang mereka sadap akan memiliki MAC yang benar-benar acak, meskipun mereka mengetahui isi pesannya. Hal ini mempersulit penjahat untuk memalsukan pesan atau menipu salah satu pihak.

Mekanisme dari HMAC didefinisikan sebagai berikut:

a. Terdapat fungsi *hash* H, *secret key* K. H adalah fungsi *hash* yang melakukan iterasi terhadap *block* data dengan panjang B byte dan memiliki keluaran dengan panjang L *byte*. Panjang dari $L \leq B$.
b. Didefinisikan dua konstanta yaitu $ipad$ (*inner padding*) dan $opad$ (*outer padding*) dengan persamaan:
   $ipad$ = *byte* 0x36 diulang sebanyak B kali
   $opad$ = *byte* 0x36 diulang sebanyak B kali
c. Untuk menghitung HMAC dari suatu pesan m digunakan persamaan berikut.

$$HMAC(K,m) = H((K' \oplus \text{opad}) + \text{H}(\text{K}' \oplus \text{ipad}) + \text{m}))$$

Langkah rinci untuk pembuatan HMAC adalah sebagai berikut:

1) *Append* bit 0 ke K hingga panjangnya B.
2) Lakukan XOR hasil 1) dengan $ipad$.
3) *Append* pesan m ke hasil yang didapat dari 2).
4) Lakukan fungsi *hash* H terhadap hasil 3).
5) Lakukan XOR hasil (1) dengan $opad$.
6) *Append* hasil 4) ke hasil 5).
7) Lakukan fungsi *hash* H terhadap hasil 6).

Jadi hal utama yang membedakan HMAC adalah adanya dua ronde fungsi *hash* yang dilakukan terhadap pesan asli.

F. Skema Implementasi Cryptographic Checksum

Skema implementasi sistem *cryptographic checksum* akan dilakukan dengan menggunakan Carter-Wegman MAC dengan pendekatan *universal hashing* dan AES untuk menggantikan HMAC. Sebelum menelisik lebih lanjut terkait skema implementasinya, perlu untuk memperhatiakn beberapa konsiderasi dan komponen pendukung dari skema ini.

1. Memahami Kekurangan HMAC dalam Skala Besar

   HMAC (*Hash-based Message Authentication Code*) adalah teknik yang sangat populer untuk verifikasi integritas dan autentikasi data yang *robust* dan umum diimplementasikan dalam konteks transaksi *e-commerce*. Namun, HMAC dapat menjadi kurang efisien untuk transaksi dalam skala besar karena:

   a. *Overhead* Komputasi
      HMAC menggunakan fungsi *hash* kriptografi seperti SHA-256, yang dapat menjadi lambat ketika volume transaksi tinggi.
   b. Konsumsi Memori
      HMAC memerlukan penyimpanan *hash state* yang bisa menjadi overhead tambahan.
   c. Pengelolaan Kunci
      Mengelola kunci simetrik untuk setiap transaksi bisa menjadi kompleks dalam lingkungan dengan skala besar.

2. Konsep Dasar Carter-Wegman MAC dengan *Universal Hashing*

   Carter-Wegman MAC menggunakan konsep *universal hashing*, yaitu memilih fungsi *hash* dari keluarga fungsi *hash* universal yang memiliki probabilitas kolisi yang sangat rendah. Keluarga fungsi *hash* universal memastikan bahwa dua masukan berbeda jarang menghasilkan *hash* yang sama, bahkan jika fungsi hash dipilih secara acak.

   Keuntungan *Universal Hashing* adalah sebagai berikut

   a. Efisiensi
      Hashing universal lebih cepat dibandingkan dengan fungsi *hash* kriptografi yang berat.
   b. Keamanan Terbukti
      Sifat teoritisnya memungkinkan analisis keamanan yang kuat.

3. Pembangkit Bilangan Acak untuk Kunci

   Kunci dalam Carter-Wegman MAC harus dihasilkan secara acak untuk memastikan keamanan. RNG yang kuat diperlukan untuk menghasilkan kunci yang unik dan tidak dapat diprediksi. Oleh sebab itu, penting untuk mempertimbangkan panjang kunci yang cukup untuk menahan serangan *brute force*, umumnya 128 atau 256 bit.

4. Menggunakan AES untuk Pengamanan Tambahan

   Advanced Encryption Standard (AES) digunakan dalam kombinasi dengan *universal hashing* untuk mengamankan hasil *hashing*. AES memberikan lapisan keamanan tambahan dengan mengenkripsi hasil *hash* sebelum menjadi MAC akhir.

   Proses Penggunaan AES adalah sebagai berikut.

   a. *Hashing* Pesan
      Gunakan *universal hashing* untuk menghasilkan nilai *hash* dari pesan.
   b. Enkripsi *Hash*
      Enkripsi nilai *hash* tersebut menggunakan AES dengan kunci rahasia yang berbeda dari kunci *hash*.
   c. MAC Final
      Luaran dari enkripsi AES menjadi MAC yang dikirim bersama pesan.

5. Prosedur Implementasi

Berikut adalah alur implementasi dari skema *cryptographic checksum* yang dibangun.

Langkah 1: Persiapan Kunci

a. Kunci *Hash*: Hasilkan kunci *hash* acak untuk fungsi *hash universal*.
b. Kunci AES: Hasilkan kunci AES acak untuk enkripsi hasil *hash*.

Langkah 2: *Universal Hashing*

a. Pilih Fungsi *Hash*: Pilih fungsi *hash* dari keluarga fungsi *hash universal*.
b. *Hash* Pesan: *Hash* pesan dengan kunci *hash* untuk mendapatkan nilai *hash* sementara.

Langkah 3: Enkripsi dengan AES

Enkripsi *Hash*: Enkripsi nilai *hash* sementara menggunakan AES dengan kunci AES.

Langkah 4: Penggunaan dalam *E-commerce*

a. Pengiriman MAC: Sertakan MAC hasil enkripsi bersama dengan pesan saat transaksi.
b. Verifikasi di Server: Di sisi server, gunakan kunci hash dan kunci AES yang sama untuk menghitung dan memverifikasi MAC pesan yang diterima.

IV. IMPLEMENTASI

Pada makalah ini, penulis telah berhasil membuat implementasi analisis permalasahan dan solusi diatas dalam bahasa pemrograman Python. Alasan pemilihan Bahasa pemrograman python adalah karena banyaknya *library* yang dapat memudahkan proses pengimplementasian algoritma seperti Crypto untuk menggunakan beragam fungsionalitas terkait kriptografi. Berikut merupakan penjelasan lengkap mengenai implementasi solusi dalam Python.

A. Universal Hash Function

Universal hash function diimplementasikan pada kelas **HashFunction**. Kelas ini dirancang untuk menangani proses *hashing* polinomial dari pesan yang diberikan. Pada inisialisasi, kelas ini menerima sebuah kunci *hash* (key) dalam bentuk string, yang digunakan dalam perhitungan *hashing*. Fungsi *hashing* polinomial yang disediakan oleh kelas ini bekerja dengan menggabungkan karakter-karakter pesan dan kunci secara siklis, kemudian menerapkan operasi matematika modular untuk menghitung nilai *hash*. Algoritma ini memanfaatkan konstanta polinomial *p* untuk meningkatkan daya tahan terhadap kolisi *hash* dan memastikan distribusi nilai *hash* yang lebih seragam. Tujuan utama dari kelas ini adalah menghasilkan representasi numerik unik dari pesan yang akan digunakan lebih lanjut dalam proses enkripsi untuk menjaga integritas pesan.

Kelas **HashFunction**

```
class HashFunction:
    """Class to handle polynomial hashing."""

    tabnine: test | explain | document | ask
    def __init__(self, key: str):
        self.key = key

    tabnine: test | explain | document | ask
    def polynomial_hash(self, message: str, prime: int = int(1e9 + 9)) -> int:
        p = 31
        hash_value = 0
        p_power = 1

        for i, char in enumerate(message):
            hash_value = (hash_value + (ord(char) + ord(self.key[i % len(self.key)])) * p_power) % prime
            p_power = (p_power * p) % prime

        return hash_value
```

Gambar 4.1. *Implementasi Kelas HashFunction.*
Sumber: Dokumentasi Pribadi

B. Cipher AES mode Galois/Counter Mode (GCM)

Bagian ini diimplementasikan pada kelas **AESCipher**. Kelas ini bertanggung jawab atas proses enkripsi dan dekripsi menggunakan *Advanced Encryption Standard* (AES) dengan mode Galois/Counter Mode (GCM). Saat diinisialisasi, kelas ini menerima kunci AES (*key*) dalam bentuk *byte array*. Metode enkripsi dari kelas ini mengambil teks biasa (*plaintext*) dan mengenkripsinya dengan kunci yang diberikan, menghasilkan *ciphertext* yang dikombinasikan dengan *nonce* dan *tag* autentikasi untuk memastikan keamanan data. Metode dekripsi akan mendekripsi *ciphertext* yang diterima, memisahkan *nonce*, data terenkripsi, dan *tag*, kemudian menyiapkan cipher untuk verifikasi autentikasi. Kelas ini sangat penting dalam menyediakan kerahasiaan dan integritas data melalui proses enkripsi simetris yang kuat.

Kelas **AESCipher**

```
class AESCipher:
    """Class to handle AES encryption and decryption."""

    tabnine: test | explain | document | ask
    def __init__(self, key: bytes):
        self.key = key

    tabnine: test | explain | document | ask
    def encrypt(self, plaintext: str) -> bytes:
        cipher = AES.new(self.key, AES.MODE_GCM)
        cipher.update(str(plaintext).encode())
        ciphertext, tag = cipher.encrypt_and_digest(b"")
        return cipher.nonce + ciphertext + tag

    tabnine: test | explain | document | ask
    def decrypt(self, ciphertext: bytes):
        iv = ciphertext[:16]
        encrypted_data = ciphertext[16:-16]
        tag = ciphertext[-16:]
        cipher = AES.new(self.key, AES.MODE_GCM, nonce=iv)
        return cipher, encrypted_data, tag
```

Gambar 4.2. *Implementasi Kelas AESCipher.*
Sumber: Dokumentasi Pribadi

C. Carter-Wegman MAC

Carter-Wegman MAC diimplementasikan pada kelas **MACGenerator**. Kelas ini mengintegrasikan fungsi *hashing* dan enkripsi untuk menghasilkan dan memverifikasi menggunakan Carter-Wegman *Message Authentication Codes* (MAC). Pada saat inisialisasi, kelas ini menerima objek **HashFunction** dan **AESCipher**, yang digunakan secara bersama untuk proses autentikasi pesan. Dalam metode *generate_mac*, kelas ini menghitung nilai *hash* dari pesan yang diberikan menggunakan **HashFunction**, kemudian mengenkripsi nilai *hash* tersebut menggunakan

AESCipher untuk menghasilkan MAC. Metode *verify_mac* bekerja dengan mendekripsi MAC yang diterima, kemudian memverifikasi keasliannya dengan membandingkan nilai *hash* yang dihitung dari pesan asli dengan *tag* autentikasi dari hasil dekripsi. Kelas ini sangat penting untuk memastikan bahwa pesan tidak dimodifikasi dan diterima dari sumber yang benar, dengan memanfaatkan kekuatan gabungan dari *hashing* polinomial dan enkripsi AES.

Kelas **MACGenerator**

```python
class MACGenerator:
    """Class to generate and verify MAC using hashing and AES."""

    tabnine: test | explain | document | ask
    def __init__(self, hash_function: HashFunction, aes_cipher: AESCipher):
        self.hash_function = hash_function
        self.aes_cipher = aes_cipher

    tabnine: test | explain | document | ask
    def generate_mac(self, message: str) -> bytes:
        hash_value = self.hash_function.polynomial_hash(message)
        return self.aes_cipher.encrypt(hash_value)

    tabnine: test | explain | document | ask
    def verify_mac(self, message: str, received_mac: bytes) -> bool:
        hash_value = self.hash_function.polynomial_hash(message)
        cipher, encrypted_data, tag = self.aes_cipher.decrypt(received_mac)
        cipher.update(str(hash_value).encode())
        try:
            cipher.verify(tag)
            return True
        except ValueError:
            return False
```

Gambar 4.3. *Implementasi Kelas MACGenerator.*
Sumber: Dokumentasi Pribadi

V. PENGUJIAN

Proses pengujian dilakukan dengan mempertimbangkan berbagai kemungkinan kasus kesalahan pesan yang dapat terjadi jika *cryptographic checksum* ini digunakan. Berikut adalah detail dari pengujiannya. Misalkan terdapat sebuah data hasil transaksi *e-commerce* sebagai berikut.

```
{
    "Name": "Michael Leon",
    "ID_Invoice": "Invoice/234/8i",
    "Issuer": "Toko Baju X",
    "Buy_Date": "17/08/2100",
    "Price": 200000
}
```

Maka data tersebut dapat disimpan sebagai JSON untuk kemudian dikirmkan kepada penerima *invoice*. Berikut adalah berbagai kemungkinan ancaman keamanan yang mungkin terjadi dan bagaimana skema *cryptographic checksum* yang dibangun dapat mengatasinya.

Kondisi Normal

```python
# 1. Normal Condition
print("\n=== Normal Condition ===")
mac = mac_generator.generate_mac(message)
print('Generated MAC:', mac.hex())
is_valid = mac_generator.verify_mac(message, mac)
print('Is MAC valid?', is_valid)
```

```
=== Normal Condition ===
Generated MAC:
2ed09fefac01701c09ebc13eaa2627f4026113410295130261143da3be839a59
Is MAC valid? True
```

Konten Dimodifikasi

```python
# 2. Message Modified
print("\n=== Message Modified ===")
modified_message = '{"Name": "Michael Leon","ID_Invoice": "Invoice/234/8i","Issuer": "Toko Baju X","Buy_Date": "17/08/2100","Price": 300000}'
is_valid = mac_generator.verify_mac(modified_message, mac)
print('Is MAC valid with modified message?', is_valid)
```

```
=== Message Modified ===
Is MAC valid with modified message? False
```

MAC Dimodifikasi

```python
# 3. MAC Modified
print("\n=== MAC Modified ===")
modified_mac = bytearray(mac)
modified_mac[0] = (modified_mac[0] + 1) % 256  # Modify the first byte of the nonce
is_valid = mac_generator.verify_mac(message, bytes(modified_mac))
print('Is MAC valid with modified MAC?', is_valid)
```

```
=== MAC Modified ===
Is MAC valid with modified MAC? False
```

Kunci Hash Salah

```python
# 4. Wrong Hash Key
print("\n=== Wrong Hash Key ===")
wrong_hash_key = get_random_bytes(16).hex()
wrong_hash_function = HashFunction(wrong_hash_key)
wrong_mac_generator = MACGenerator(wrong_hash_function, aes_cipher)
is_valid = wrong_mac_generator.verify_mac(message, mac)
print('Is MAC valid with wrong hash key?', is_valid)
```

```
=== Wrong Hash Key ===
Is MAC valid with wrong hash key? False
```

Kunci AES Salah

```python
# 5. Wrong AES Key
print("\n=== Wrong AES Key ===")
wrong_aes_key = get_random_bytes(16)
wrong_aes_cipher = AESCipher(wrong_aes_key)
wrong_mac_generator = MACGenerator(hash_function, wrong_aes_cipher)
is_valid = wrong_mac_generator.verify_mac(message, mac)
print('Is MAC valid with wrong AES key?', is_valid)
```

```
=== Wrong Hash Key ===
Is MAC valid with wrong hash key? False
```

Penanganan Initial Vector (IV) yang Salah

```python
# 6. Wrong IV Handling (simulated by modifying nonce)
print("\n=== Wrong IV Handling ===")
modified_nonce_mac = bytearray(mac)
modified_nonce_mac[:16] = get_random_bytes(16)  # Modify the nonce
is_valid = mac_generator.verify_mac(message, bytes(modified_nonce_mac))
print('Is MAC valid with wrong IV handling?', is_valid)
```

```
=== Wrong IV Handling ===
Is MAC valid with wrong IV handling? False
```

Dapat diamati bahwa skema *cryptographic chekcsum* tidak akan mengizinkan akses dengan nilai yang benar pada data yang telah dimodifikasi, MAC yang dimodifikasi, hingga salah dalam pemrosesan *initial vector* (IV).

Analisis dan Hasil

Dalam rangka menghasilkan hasil analisis yang lebih komprehensif, diperlukan hasil dari berbagai pendekatan. Berikut adalah penjelasan detailnya.

1. **Keamanan**

   Aspek ini dijamin dengan menggunakan AES dalam kombinasi *dengan universal hashing* untuk memberikan keamanan tambahan. *Hash* sementara yang dihasilkan oleh *hashing* universal dienkripsi dengan AES sebelum digunakan sebagai MAC akhir, melindungi terhadap berbagai serangan yang mungkin menargetkan fungsi hash.

2. **Kinerja**

   Pendekatan ini lebih efisien daripada HMAC untuk transaksi skala besar karena:

   a. *Hashing* Lebih Cepat: *Universal hashing* umumnya lebih cepat dibandingkan dengan fungsi *hash* kriptografi yang digunakan dalam HMAC.
   b. Enkripsi Terpisah: AES hanya digunakan pada hasil *hash*, mengurangi *overhead* komputasi dibandingkan dengan penggunaan enkripsi langsung pada seluruh pesan.

3. **Skalabilitas**

   Implementasi ini cocok untuk digunakan pada platform *e-commerce* skala besar karena beberapa hal berikut.

   a. Pengurangan *Overhead*: Mengurangi beban komputasi pada server saat menangani volume transaksi yang besar.
   b. Manajemen Kunci: Memisahkan kunci untuk *hashing* dan enkripsi memudahkan pengelolaan kunci dalam skala besar.

4. **Pertimbangan**

   Terlepas dari berbagai aspek keuntungan yang dijelaskan sebelumnya, terdapat beberapa hal yang perlu untuk dipertimbangkan.

   a. Optimasi Fungsi *Hash*
      Mengeksplorasi keluarga fungsi *hash* yang lebih efisien untuk berbagai konteks transaksi.
   b. Peningkatan Keamanan AES
      Mengkaji metode enkripsi tambahan bila diperlukan untuk meningkatkan keamanan dengan mempertimbangkan *overhead* komputasi.
   c. Uji Coba dalam Skala Nyata
      Implementasi dan pengujian dalam skala besar pada platform *e-commerce nyata* untuk evaluasi lebih lanjut.

VI. KESIMPULAN

Implementasi Carter-Wegman MAC sebagai *cryptographic checksum* dalam konteks transaksi *e-commerce real-time* telah terbukti efektif dalam meningkatkan keamanan dan integritas data transaksi. Dengan memanfaatkan kombinasi fungsi *hash* polinomial dan enkripsi AES dalam mode Galois/Counter Mode (GCM), metode ini berhasil menciptakan mekanisme otentikasi yang cepat dan kuat terhadap berbagai ancaman keamanan. Dibandingkan dengan penggunaan HMAC konvensional, Carter-Wegman MAC menawarkan kinerja yang lebih baik dalam hal efisiensi komputasi dan kapasitas untuk menangani transaksi dalam skala besar, tanpa mengorbankan tingkat keamanan yang diperlukan. Selain itu, fleksibilitas dalam pengaturan kunci hash dan kunci AES memungkinkan penyesuaian yang lebih baik terhadap kebutuhan spesifik dari berbagai aplikasi *e-commerce*, memastikan perlindungan data yang sesuai dengan standar industri terkini.

Namun, penerapan Carter-Wegman MAC juga memerlukan perhatian khusus pada manajemen kunci dan penanganan *nonce* untuk memastikan bahwa sistem tetap aman. Pengujian pada berbagai skenario, termasuk perubahan pesan dan modifikasi MAC, menunjukkan bahwa metode ini mampu mendeteksi dan menolak transaksi yang tidak sah secara efektif, menambah lapisan perlindungan penting terhadap manipulasi data. Dengan demikian, Carter-Wegman MAC menawarkan solusi yang layak dan lebih aman untuk mitigasi risiko penipuan dalam transaksi *e-commerce*, memfasilitasi verifikasi transaksi yang andal dan melindungi integritas data pengguna dalam lingkungan digital yang semakin kompleks.

VII. UCAPAN TERIMA KASIH

Puji Syukur hanya kepada Tuhan Yang Maha Esa karena hanya atas berkat dan limpahan rahmatNya, penulis dapat menyelesaikan makalah ini dengan baik. Terima kasih juga penulis sampaikan kepada Bapak Dr. Ir. Rinaldi, M.T. sebagai dosen pengampu dalam mata kuliah IF4020 Kriptografi atas ilmu yang telah diberikan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Tidak lupa terima kasih juga penulis sampaikan kepada orang tua yang senantiasa memberikan dukungan dan motivasi kepada penulis.

VIII. LAMPIRAN

Penulis berhasil melakukan implementasi fisik lengkap dengan pengujian dari sistem *cryptographic checksum* yang telah disusun dan mengarsipkannya dalam sebuah tautan yang dapat diakses secara publik. Uji coba dan pengembangan lebih lanjut dapat dilakukan dengan mengakses laman github penulis berikut :

https://github.com/mikeleo03/Carter-Wegman-MAC

REFERENSI

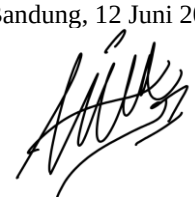
[1] Statista. (2022). Worldwide retail e-commerce sales, diambil dari dari https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/ (diakses pada 1 Juni 2024).
[2] Laudon, K. C., & Traver, C. G. (2020). E-commerce 2020: Business, Technology, and Society. Pearson.
[3] Javelin Strategy & Research. (2022). 2022 Identity Fraud Study: The Virtual Battleground, dari https://www.javelinstrategy.com/coverage-area/2022-identity-fraud-study (diakses pada 1 Juni 2024).
[4] Stallings, W. (2020). Cryptography and Network Security: Principles and Practice. Pearson.
[5] R. Sobti and G. Geetha, “Cryptographic Hash functions - a review,” IJCSI Int. J. Comput. Sci. Issues, vol. 9, no. 2, pp. 461–479, 2012.
[6] R. Munir, “Fungsi Hash,” 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/24-Fungsi-hash-2024.pdf (diakses pada 1 Juni 2024).
[7] H. Krawczyk, “Keying Hash Functions for Message Authentication,” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1995.
[8] R. Munir, “Message - Authentication Code Algorithms,” Cryptography for Developers, 2020.

https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2020-2021/MAC-2020.pdf (diakses pada 1 Juni 2024).
[9] Carter, L., & Wegman, M. N. (1981). Universal classes of hash functions. Journal of Computer and System Sciences, 18(2), 143-154.
[10] Katz, J., & Lindell, Y. (2020). Introduction to Modern Cryptography. CRC Press.
[11] NIST. (2015). Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Special Publication 800-90A. National Institute of Standards and Technology.
[12] Daemen, J., & Rijmen, V. (2002). The Design of Rijndael: AES - The Advanced Encryption Standard. Springer.

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



Michael Leon Putra Widhi
13521108