BAB II
TINJAUAN PUSTAKA & DASAR TEORI

2.1 Tinjauan Pustaka

Dalam penelitian ini, berbagai jurnal telah ditinjau untuk memahami metode dan hasil penelitian sebelumnya terkait penggunaan *hashchain* dan algoritma kriptografi. Tinjauan pustaka ini berguna untuk mengidentifikasi pendekatan yang telah digunakan, mengevaluasi keefektifan metode tersebut, dan menentukan relevansi serta kontribusi penelitian ini terhadap peningkatan keamanan dan privasi data dalam sistem informasi. berikut beberapa penelitian yang relevan dan signifikan dalam bidang ini.

1. Algoritma *hashchain* satu arah dalam enkripsi selektif data sensitif

   Sebuah studi oleh Nagaraj & Mohanraj (2020) dalam konferensi *ICACCCN* menyoroti penggunaan algoritma *hashchain* satu arah dalam enkripsi selektif data sensitif dalam aliran, sementara data lainnya dikirim dalam bentuk teks biasa. Hasilnya menunjukkan pentingnya *hashchain* dalam memastikan integritas data dan mendeteksi upaya manipulasi.

   Meskipun metode *hashchain* satu arah efektif dalam menjaga integritas data, keterbatasannya terletak pada sifat yang tidak dapat dikembalikan ke bentuk asli, sehingga menjadi kendala dalam sistem yang memerlukan dekripsi untuk analisis atau pemrosesan ulang. Dalam konteks penelitian ini, konsep hashchain tetap relevan karena mampu mendeteksi manipulasi data, namun penerapannya perlu dikombinasikan dengan algoritma kriptografi yang mendukung proses enkripsi dan dekripsi, seperti RSA dan AES, agar sistem informasi tugas belajar pegawai UNDIP tidak hanya menjamin integritas tetapi juga kerahasiaan data secara menyeluruh.

2. Analisis Komparatif Algoritma Kriptografi dalam Keamanan Informasi

Meftah dkk. (2024) membandingkan berbagai algoritma kriptografi simetris dan asimetris, termasuk AES, RSA, DES, 3DES, Blowfish, ElGamal, ECC, dan algoritma hashing seperti MD5 dan SHA. Analisis dilakukan berdasarkan faktor-faktor utama seperti kecepatan enkripsi dan dekripsi, tingkat keamanan terhadap serangan kriptografi, ukuran kunci, ukuran blok, jumlah putaran, serta kompleksitas implementasi. AES diidentifikasi sebagai algoritma simetris yang paling efisien dalam hal kecepatan dan penggunaan memori, dengan ukuran blok 128 bit dan variasi panjang kunci 128, 192, dan 256 bit. RSA, sebagai algoritma asimetris, dinilai sangat aman untuk pertukaran kunci tetapi memiliki kelemahan dalam kecepatan enkripsi data berukuran besar.

Perbandingan yang dilakukan memberikan gambaran umum yang bermanfaat mengenai kelebihan dan kelemahan masing-masing algoritma, namun analisisnya terbatas pada perbandingan individual tanpa mengusulkan integrasi untuk mengatasi kelemahan masing-masing algoritma. Dalam konteks ini, penelitian yang dilakukan pada tesis ini melangkah lebih jauh dengan menggabungkan AES dan RSA dalam skema kriptografi *hybrid*, sehingga memanfaatkan kecepatan AES untuk enkripsi data dan keamanan RSA untuk distribusi kunci.

3. Algoritma *hybrid* menggunakan enkripsi simetris dan asimetris

Jintcharadze & Iavich (2020) meneliti penggunaan algoritma *hybrid* dengan enkripsi simetris (*AES* dan *Twofish*) untuk kerahasiaan data, dan enkripsi asimetris (*ElGamal* dan *RSA*) untuk pertukaran kunci. Hasilnya menunjukkan bahwa model *AES+RSA* menawarkan keamanan yang signifikan, sementara *Twofish+RSA* lebih cepat dalam komputasi.

Kombinasi metode kriptografi memperkuat keamanan karena memanfaatkan keunggulan masing-masing algoritma. *AES* dan *Twofish* memberikan enkripsi data yang cepat dan aman, sementara *RSA* dan *ElGamal* menyediakan mekanisme yang aman untuk pertukaran kunci.

Kombinasi kriptografi tersebut dapat melindungi data sensitif. Penggunaan kombinasi *AES* dan *RSA* dalam sistem informasi tugas belajar pegawai UNDIP dapat memberikan keamanan yang signifikan dan kecepatan pemrosesan yang baik, dengan *RSA* untuk pertukaran kunci yang aman dan *AES* untuk enkripsi data yang efisien.

4. Efisiensi algoritma *RSA* dan *AES* berdasarkan waktu eksekusi dan efek *avalanche*

Sholikhatin dkk. (2023) membandingkan efisiensi algoritma *RSA* dan *AES* dengan memperhatikan waktu eksekusi serta menganalisis *avalanche effect*. Penelitian ini menunjukkan bahwa waktu rata-rata enkripsi dan dekripsi *AES* lebih kompetitif, dan efek *avalanche* dari *AES* lebih baik dibandingkan *RSA*.

AES tidak hanya lebih cepat dalam proses enkripsi dan dekripsi, tapi juga memberikan tingkat keamanan yang lebih baik melalui efek *avalanche*, yang memastikan bahwa setiap perubahan kecil pada teks jelas menyebabkan perubahan signifikan pada teks yang dienkripsi. Ini penting untuk mempertahankan kerahasiaan informasi. Efisiensi dan tingkat keamanan tinggi dari *AES* menjadikannya komponen ideal dalam sistem informasi tugas belajar pegawai UNDIP, di mana perlunya enkripsi cepat dan aman adalah kritis untuk mengamankan informasi sensitif.

5. Sistem aman untuk mengamankan data kejahatan menggunakan hybrid: RSA-AES dan Blowfish-triple DES

Agrawal dkk. (2025) mengusulkan penerapan kriptografi hybrid untuk melindungi data kriminal yang disimpan dan ditransfer melalui cloud. Penelitian ini menggabungkan dua pendekatan hybrid: RSA-AES dan Blowfish-Triple DES. RSA digunakan untuk mengenkripsi kunci simetris, sedangkan AES, Blowfish, dan Triple DES digunakan untuk enkripsi data. Hasilnya menunjukkan bahwa algoritma hybrid memberikan kinerja lebih

baik dibandingkan metode tunggal, dengan hybrid RSA-AES unggul dalam efisiensi dan keamanan untuk data teks.

Stabilitas dan efisiensi dekripsi dari algoritma *hybrid*, khususnya dalam menangani *file* besar dengan format data teks, menunjukkan keunggulannya dalam aplikasi yang memerlukan pemrosesan data dalam jumlah besar tanpa penurunan performa signifikan. Penggunaan algoritma *hybrid* dengan *AES* dan *RSA* akan sangat bermanfaat dalam sistem informasi tugas belajar pegawai, yang memiliki data berupa teks.

6. Penyederhanaan *blockchain* dengan skema perlindungan privasi berbasis *hashchain*

Penelitian oleh Zhao dkk. (2019) mengusulkan skema berbasis *hashchain* untuk autentikasi dan integritas data dalam *blockchain*, menemukan bahwa solusi ini efektif dalam menjaga keamanan dan kinerja.

Skema *hashchain* memberikan arsitektur keamanan yang kuat untuk otentikasi dan integritas data, terutama dalam jaringan yang memerlukan keandalan tinggi dan perlindungan terhadap manipulasi data. *Hashchain* dapat digunakan untuk mendukung otentikasi dan integritas dalam sistem informasi tugas belajar pegawai UNDIP, memastikan bahwa setiap transaksi dan perubahan data dapat diverifikasi keabsahannya.

7. *Hash-Chain Fog/Edge*: *Hashchain* Berbasis Mode untuk Protokol Otentikasi Timbal Balik yang Aman Menggunakan *Zero-Knowledge Proofs* di *Fog/Edge*

Pardeshi dkk. (2022) membahas pengembangan protokol autentikasi mutual yang aman pada arsitektur *fog/edge computing*. Penelitian ini mengusulkan protokol *Hash-Chain Fog/Edge* (HCFE) yang memanfaatkan *mode-based hash chain* untuk autentikasi dinamis antar perangkat *IoT*. HCFE menggunakan blockchain untuk manajemen identitas terdistribusi, mengurangi ketergantungan pada server terpusat, serta meminimalkan beban komputasi agar sesuai dengan perangkat *IoT* yang memiliki

keterbatasan sumber daya. Hasil eksperimen menunjukkan bahwa HCFE efisien, aman terhadap serangan aktif maupun pasif, dan memiliki biaya komputasi rendah.

Pendekatan HCFE memiliki relevansi dengan penelitian yang mengintegrasikan algoritma RSA dan AES dalam sistem informasi berbasis hashchain. Keterkaitan utama terletak pada pemanfaatan struktur hash-chain untuk menjaga integritas data dan konsep autentikasi berbasis kriptografi yang efisien. Prinsip pengelolaan identitas terdistribusi dan penggunaan hash-chain untuk pencatatan autentikasi dapat diadaptasi untuk memperkuat model keamanan pada sistem informasi tugas belajar. Selain itu, mekanisme hash-chain yang saling terhubung antar blok dapat melengkapi enkripsi hybrid RSA-AES, sehingga tidak hanya menjaga kerahasiaan data, tetapi juga memastikan integritas dan keaslian interaksi antar entitas dalam sistem.

8. Pendekatan enkripsi hibrida dengan *RSA* dan *AES* untuk peningkatan keamanan dan kinerja

Durge & Deshmukh (2025) mengusulkan metode enkripsi *hybrid* untuk meningkatkan keamanan data pada lingkungan cloud. Pendekatan ini menggabungkan RSA untuk enkripsi kunci dan AES untuk enkripsi data, sehingga menciptakan dua lapisan keamanan. Proses dimulai dengan konversi teks ke bentuk *byte*, kemudian dienkripsi menggunakan RSA dan AES secara berlapis. Penelitian ini menekankan keunggulan kombinasi algoritma simetris dan asimetris dalam mengatasi kelemahan masing-masing, serta menguji performa melalui parameter seperti throughput, kompleksitas waktu, dan ketahanan terhadap serangan. Hasilnya menunjukkan bahwa metode hybrid RSA-AES memberikan tingkat keamanan yang sangat tinggi dan efisiensi yang lebih baik dibandingkan penggunaan algoritma tunggal.

Penelitian ini memberikan kontribusi penting dengan menunjukkan efektivitas enkripsi hybrid RSA-AES dalam meningkatkan keamanan dan performa. Namun, fokus penelitian terbatas pada aspek kerahasiaan dan

efisiensi, tanpa membahas mekanisme integritas data atau deteksi manipulasi.

9. Pendekatan kombinasi *RSA* dan *AES* untuk mengenkripsi data aplikasi

Aminuddin (2020) menggunakan pendekatan kombinasi *RSA* dan *AES* untuk mengenkripsi data aplikasi. Hasil penelitian menunjukkan bahwa kombinasi ini meningkatkan keamanan dan efisiensi waktu enkripsi dan dekripsi.

Kombinasi *RSA* dan *AES* memberikan tingkat keamanan yang lebih tinggi dengan waktu eksekusi yang optimal, menjadikannya solusi ideal untuk aplikasi yang membutuhkan tindakan keamanan berlapis. Kombinasi *AES* dan *RSA* sesuai untuk meningkatkan keamanan dan efisiensi sistem informasi tugas belajar pegawai UNDIP, yang memerlukan pengamanan data sensitif pegawai dengan cepat dan aman.

10. Keamanan transmisi data dengan algoritma *hybrid AES* dan *RSA*

Dr Asha Ambhaikar (2021) menggabungkan *AES* dan *RSA* untuk memberikan keamanan yang lebih baik dalam transmisi data, menemukan bahwa algoritma hybrid lebih aman dan efisien daripada penggunaan individual.

Kombinasi metode memberikan keamanan yang lebih baik tanpa mengorbankan kinerja, suatu keunggulan penting dalam pengelolaan data yang sensitif. Penggunaan algoritma hybrid *AES* dan *RSA* dapat memberikan keamanan yang lebih tinggi untuk data tugas belajar pegawai UNDIP.

11. Keamanan Informasi Jaringan dan Perlindungan Data Berbasis Teknologi Enkripsi Data

Ping (2022) melakukan pendekatan keamanan berbasis kriptografi untuk meningkatkan perlindungan data dalam jaringan nirkabel dan *cloud*. Penelitian ini menekankan penggunaan algoritma kriptografi modern, termasuk AES dan RSA, serta teknik *hybrid* untuk mengatasi kelemahan

masing-masing algoritma. Fokus utama adalah pada efisiensi enkripsi, manajemen kunci, dan ketahanan terhadap serangan seperti *brute force* dan *man-in-the-middle*.

Penelitian ini memberikan kontribusi penting dalam mendukung konsep enkripsi *hybrid* untuk meningkatkan keamanan data, namun pembahasan masih terbatas pada aspek kerahasiaan dan efisiensi tanpa mengintegrasikan mekanisme verifikasi integritas data.

12. Pendekatan *hybrid RSA* dan *AES* untuk meningkatkan keamanan data

Liu dkk. (2018) menggabungkan *RSA* dan *AES* dalam pendekatan hybrid untuk meningkatkan keamanan data, menemukan bahwa strategi ini menyediakan keamanan yang kuat tanpa mengorbankan performa.

Penggunaan metode hybrid memberikan keamanan yang robust sementara tetap mempertahankan performa operasi yang tinggi. Implementasi kombinasi *RSA* dan *AES* akan memberikan keamanan yang kuat untuk sistem informasi tugas belajar pegawai UNDIP, memastikan data tetap aman dari berbagai jenis serangan.

2.2 Dasar Teori

Dalam subbab ini akan dibahas mengenai konsep dasar yang menjadi landasan penelitian, mencakup teori-teori yang relevan dengan topik kajian. Pemahaman yang mendalam mengenai teori yang digunakan sangat penting untuk memberikan kerangka berpikir yang kuat dan mendukung analisis serta interpretasi hasil penelitian.

Subbab ini disusun untuk memberikan gambaran mengenai berbagai konsep dan prinsip dasar yang mendasari implementasi dan pengembangan sistem informasi tugas belajar pegawai berbasis *hashchain* menggunakan algoritma *RSA* dan *AES*.

2.2.1 Sistem Informasi

Sistem informasi merupakan bentuk perkembangan teknologi informasi, yang memiliki tujuan untuk mempermudah pekerjaan manusia dalam pengolahan data untuk mendapatkan suatu informasi (Tajuddin dkk., 2020). Sistem informasi adalah serangkaian elemen atau komponen yang saling terkait yang mengumpulkan (*input*), memanipulasi (proses), menyimpan, dan menyebarkan (*output*) data dan informasi, serta menyediakan reaksi korektif (mekanisme umpan balik) untuk mencapai tujuan tertentu (Rizkiaputri & Auliandri, 2020).

Sistem informasi memiliki komponen baik itu data, proses, dan antarmuka yang saling berkaitan, berinteraksi, mendukung, memperbaiki dan mempunyai kesinambungan antara satu komponen dengan komponen lainnya dalam suatu bisnis, termasuk memecahkan suatu soal dan kebutuhan pembuat keputusan manajemen (Rinderle-Ma, Mangler, & Ritter, 2024).

Sistem informasi memiliki 4 komponen, yaitu (Rizkiaputri & Auliandri, 2020):

1. Komponen *input*

   Input adalah data mentah yang dikumpulkan dari berbagai sumber yang akan diolah lebih lanjut dalam sistem informasi. Input ini bisa berupa data manual atau data otomatis yang masuk ke sistem.

2. Komponen proses

   Pemrosesan adalah tahap di mana data mentah diolah dengan menggunakan berbagai metode untuk menghasilkan informasi yang berguna termasuk perhitungan matematis, validasi data, dan pembuatan keputusan berdasar data.

3. Komponen *output*

   Output adalah hasil akhir dari pemrosesan yang disajikan dalam bentuk yang dapat digunakan oleh manusia atau sistem lain.

4. Komponen *feedback*

   Feedback adalah proses di mana hasil dari sistem digunakan untuk meningkatkan atau memperbaiki proses di masa mendatang. Ini membantu dalam mengidentifikasi dan memperbaiki kesalahan sehingga sistem dapat berjalan lebih efisien dan akurat.

2.2.2 *Unified Modelling Language* (*UML*)

UML adalah sekumpulan media yang memiliki fungsi dalam menggambarkan, mendeskripsikan dan menjelaskan permodelan sistem perangkat lunak berdasarkan objek agar dapat dipahami oleh pemangku kepentingan dalam organisasi dan memudahkan programmer dalam mengerti arah alur sistem (O'Regan, 2022).

UML terdiri dari serangkaian elemen grafis yang digunakan untuk membuat diagram. Tujuan dari diagram ini adalah untuk merepresentasikan pandangan yang berbeda dari sistem dan serangkaian tampilan yang disebut model (Kusnadi dkk., 2019). *UML* memiliki fungsi untuk menggambarkan alur kerja dan apa yang dijalankan atau yang dilakukan oleh sistem, namun tidak menjelaskan bagaimana pengimplementasian sistem tersebut.

2.2.2.1 *Use case Diagram*

Use case Diagram merupakan diagram yang menunjukkan hubungan timbal balik antara sistem dan pengguna. *Use case Diagram* menunjukkan siapa yang akan pengguna dari sistem dan bagaimana pengguna akan terhubung dengan sistem (O'Regan, 2022). Deskripsi *use case* digunakan untuk melengkapi representasi tekstual dari urutan unsur-unsur pokok dari *use case* diagram antara lain (Kusnadi dkk., 2019):

1. *Actor*

   Actor merupakan semua hal yang berkaitan dengan siapa saja yang menggunakan sistem, dalam hal ini bisa merupakan sistem, manusia, atau perangkat yang mempunyai peran dalam keberlangsungan sistem.

2. *Usecase*

   Use case merepresentasikan satu tujuan sistem dan menjelaskan serangkaian operasi dan interaksi pengguna untuk mencapai tujuan tersebut. Kasus penggunaan terdiri dari serangkaian langkah yang saling terkait (skenario) yang dirancang secara otomatis atau manual dalam menyelesaikan suatu tujuan dalam proses bisnis.

3. Batasan sistem

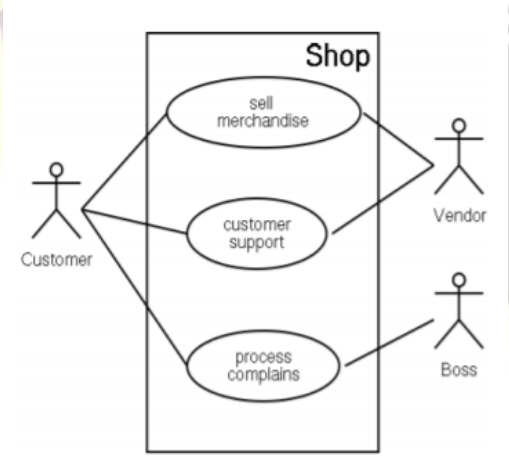
Batasan sistem digambarkan sebagai kotak dengan di dalamnya terdapat semua *use case*. Aktor digambarkan di luar kotak batasan sistem.

4. Koneksi

   Aktor dan *use case* dihubungkan dengan garis lurus, sedangkan *use case* satu dengan yang lainnya dihubungkan dengan panah.

5. Hubungan antar *use case*

   Hubungan antar *use case,* terdiri dari dua yaitu *uses* dan *extends.* Hubungan *uses/include* digunakan untuk suatu proses pada satu *use case* melibatkan suatu *use case* lainnya, paling sedikit sekali. Sedangkan hubungan *extends* dapat digunakan dalam situasi di mana sistem yang ada memiliki *use case* (proses) di mana proses ini memiliki beberapa cabang proses yang memiliki kesamaan, tetapi setiap cabang proses memiliki sesuatu yang berbeda yang tidak memungkinkan untuk mengelompokkan mereka di dalam *use case* yang sama.

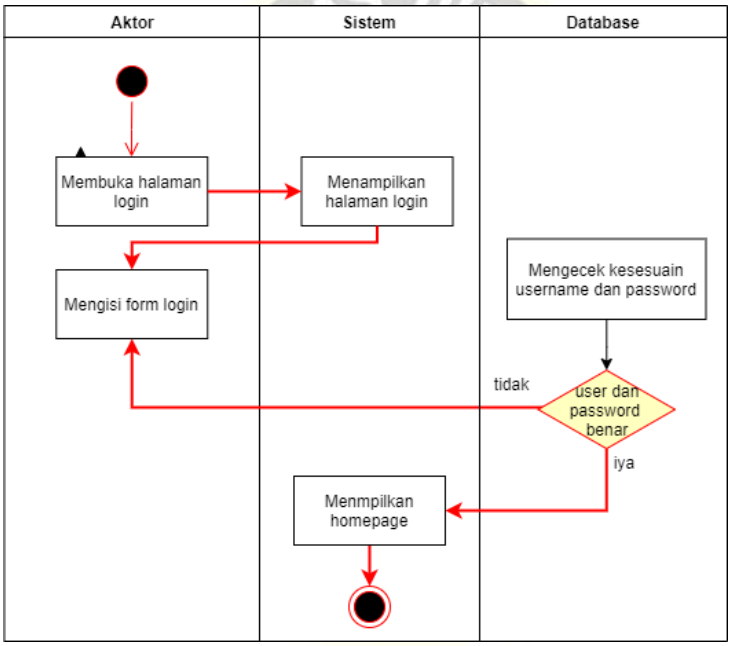


Gambar 2.1 Contoh *use case diagram*

Gambar 2.1 merupakan contoh *usecase.* Dalam *usecase* terdapat 3 *user* dalam sistem informasi yang digunakan oleh *use case diagram* tersebut. Terdapat juga 3 *cases* atau proses yang melibatkan masing-masing *user.* Dapat dilihat, pada *case sell merchandise* dan *costumer support* melibatkan 2 *user* yaitu *vendor* dan *costumer*. Sedangkan pada *case process complains* melibatkan 2 *user* yang berbeda yaitu bos dan *costumer.*

2.2.2.2 *Activity Diagram*

Diagram aktivitas menunjukkan aliran dari satu aktivitas ke aktivitas lainnya dalam suatu sistem. Diagram aktivitas menggambarkan serangkaian aktivitas, aliran sekuensial atau percabangan dari satu aktivitas ke aktivitas berikutnya, serta objek-objek yang bertindak dan dikenai tindakan. Anda menggunakan diagram aktivitas untuk mengilustrasikan pandangan dinamis dari suatu sistem. Diagram aktivitas sangat penting dalam memodelkan fungsi sebuah sistem. Diagram aktivitas menekankan aliran kontrol di antara objek-objek (O'Regan, 2022).



Gambar 2.2 Contoh *activity diagram user login*

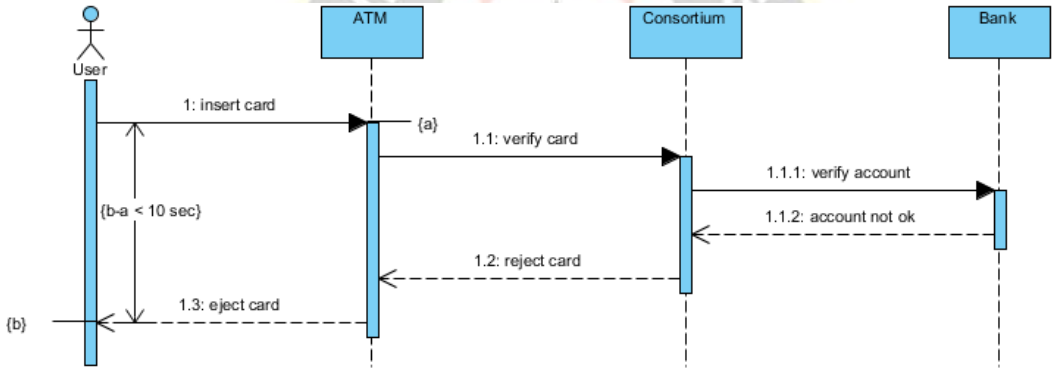
Gambar 2.2 merupakan contoh *activity diagram* yang dialami pengguna saat melakukan *login*, Pengguna diharuskan mengakses halaman *login*, kemudian sistem akan menampilkan halaman *login* yang mengharuskan pengguna mengisi *form login* yang berupa *username* dan *password*, *username* dan *password* akan dicocokan di *database*. Apabila *username* dan *password* benar maka sistem akan menampilkan halaman utama. Namun, apabila *username* dan *password* salah maka

sistem akan mengarahkan pengguna untuk mengisi *username* dan *password* kembali.

2.2.2.3 *Sequential Diagram*

Sequential diagram adalah diagram interaksi yang menggambarkan urutan waktu dari pesan-pesan. Diagram urutan menunjukkan sekumpulan objek dan pesan-pesan yang dikirim dan diterima oleh objek-objek tersebut. Objek-objek ini biasanya merupakan instansiasi dari kelas-kelas yang dinamai atau anonim, tetapi juga bisa mewakili instansiasi dari hal-hal lain, seperti kolaborasi, komponen, dan node (O'Regan, 2022).

Sequential diagram digunakan untuk melakukan pemodelan interaksi antar aktor dengan objek dan antara suatu objek dengan objek yang lain. Contoh *sequential diagram* sistem *reject* pada ATM.

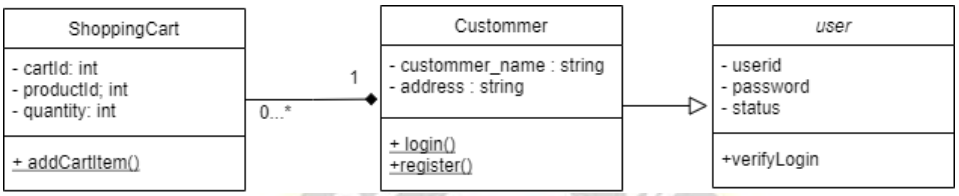


Gambar 2.3 Contoh *sequential diagram* sistem *reject* pada ATM

Gambar 2.3 merupakan contoh *sequential diagram* dalam sistem ATM. Dalam *sequential diagram* dimulai dari *user* melakukan *insert card,* setelah itu ATM akan mengirimkan *verify card* ke *consortium,* dan *consortium* akan mengirim *verify account* ke bank. Apabila bank sudah menerima *verify account* maka bank akan memproses dan akan mengirimkan *feedback* pesan *account not ok* ke *consortium. Consortium* juga akan mengirimkan *feedback reject card* ke ATM yang diikuti pengiriman pesan *eject card* oleh ATM ke *user.*

2.2.2.4 *Class Diagram*

Class diagram merupakan diagram yang digunakan untuk menunjukkan dan menampilkan kelas-kelas di dalam sistem dan juga menunjukkan relasi antar tiap kelasnya (O'Regan, 2022). Diagram kelas berguna membantu pengembang dalam menuliskan kode guna mendapatkan struktur sistem. Diagram kelas juga membantu untuk memastikan sistem merupakan rancangan yang baik. Gambar 2.4 menunjukkan contoh diagram kelas *shopping cart online*.



Gambar 2.4 Contoh *class shopping cart online*

Gambar 2.4 merupakan contoh *class diagram shopping cart online.* Dalam *class diagram* terdapat 3 tabel yang memiliki hubungan satu dengan yang lainnya. Kelas *customer* merupakan sub kelas dari kelas user sehingga kelas *customer* memiliki atribut dan *method* dari kelas user. Kemudian, kelas shoppingCart memiliki hubungan relasi komposisi dengan kelas *customer* dengan multiplisitas *many to one*, dimana customer bisa memiliki lebih dari satu shoppingCart.

2.2.3 Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) adalah metodologi terstruktur yang digunakan dalam pengembangan perangkat lunak untuk memastikan sistem yang dibangun memenuhi kebutuhan pengguna secara efisien dan berkualitas. SDLC menyediakan kerangka kerja yang sistematis untuk merencanakan, membuat, menguji, dan memelihara perangkat lunak (Shokunbi dkk., 2024). Berikut tahapan-tahapan dalam SDLC (Siva dkk., 2023) :

A. Analisis Kebutuhan

Tahap ini bertujuan untuk memahami kebutuhan pengguna dan sistem secara menyeluruh. Pengumpulan data dilakukan melalui wawancara,

observasi, dan studi literatur untuk mengidentifikasi fungsionalitas, batasan, dan aspek keamanan yang diperlukan dalam sistem.

B. Desain Sistem

Berdasarkan kebutuhan yang telah dianalisis, dilakukan perancangan arsitektur sistem. Desain ini mencakup struktur data, alur proses, dan interaksi antar komponen, biasanya divisualisasikan menggunakan diagram UML seperti diagram kelas, aktivitas, dan urutan.

C. Implementasi

Pada tahap ini, desain sistem diubah menjadi kode program menggunakan bahasa pemrograman dan teknologi yang sesuai. Pengembang membangun fitur-fitur sistem sesuai spesifikasi, termasuk integrasi algoritma dan struktur data yang telah dirancang.

D. Pengujian

Sistem diuji untuk memastikan bahwa semua fungsi berjalan sesuai dengan desain. Pengujian dilakukan secara fungsional (*blackbox*), serta pengujian performa dan integritas data menggunakan metode seperti *Big O notation* dan *Merkle Tree*.

E. Deployment

Setelah sistem lolos pengujian, dilakukan proses penempatan ke lingkungan produksi. Ini mencakup konfigurasi server dan migrasi basis data.

F. Pemeliharaan

Tahap ini merupakan proses berkelanjutan untuk memastikan sistem tetap berjalan optimal. Meliputi perbaikan *bug*, pembaruan fitur, dan penyesuaian terhadap kebutuhan baru yang muncul seiring waktu.

2.2.4 Kriptografi

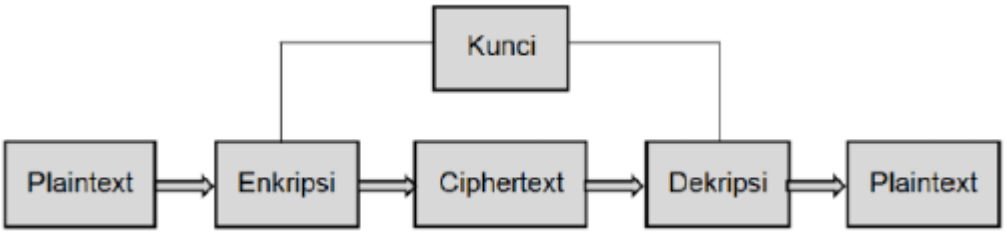
Kriptografi (*cryptography*) berasal dari Bahasa Yunani: "*cryptos*" yang berarti rahasia dan "*graphein*" yang berarti tulisan. Jadi, kriptografi berarti "*secret writing"* yang memiliki arti tulisan rahasia. Kata *graphein* didalam *cryptography* menyiratkan sebuah seni sehingga kriptografi juga berarti dapat didefinisakan

sebagai ilmu dan seni untuk mengubah pesan dengan cara mengubahnya menjadi bentuk yang tidak dapat dipahami artinya, untuk menjaga keamanan pesan dengan menggunakan teknik-teknik matematika tertentu (Jain dkk., 2025).

Kriptografi berdasarkan kunci diklasifikasikan menjadi dua yaitu kriptografi asimetri dan simetri (Ma dkk., 2020).

A. Algortima Simetris

Algoritma ini disebut simetris karena menggunakan *key* atau kunci yang sama dalam proses enkripsi dan dekripsinya. Oleh karena itu, algoritma ini disebut juga algoritma kunci tunggal atau algoritma satu kunci. Kunci dari algoritma ini adalah kunci rahasia atau kunci privat, sehingga algoritma ini disebut juga dengan algoritma kunci privat. Contoh algoritma kriptografi simetris yaitu *Caesar Cipher, Blowfish, Twofish, Hill Cipher, DES, AES*, dan lain-lainnya (Easttom, 2022).



Gambar 2.5 Alur proses algoritma simetris

Adapun kelebihan menggunkan algoritma simetris yaitu (Easttom, 2022):

- Kecepatan proses yang dilakukan lebih cepat dibandingkan menggunakan algoritma asimetris.
- Penerapan algoritma simetris ini dapat digunakan pada sistem secara real time dikarenakan kecepatan proses yang lebih cepat.

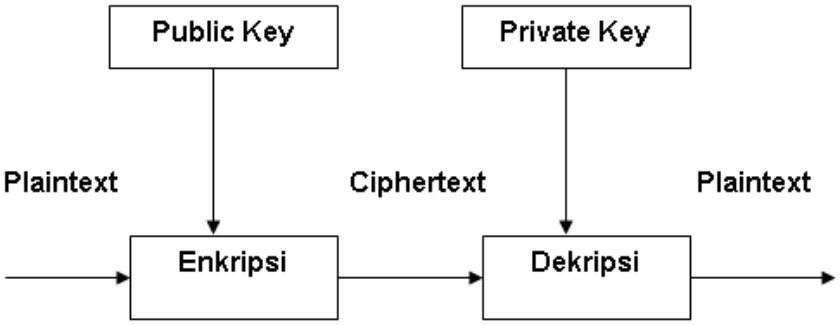
Sedangkan kelemahan menggunkan algoritma simetris yaitu (Easttom, 2022):

- Dalam manajemen kunci sangat penting dalam menentukannya. Dikarenakan setiap pengiriman pesan berbeda kebutuhan kunci yang akan digunakan.

- Dalam pendistribusian kunci akan menjadi masalah tersendiri ketika seseorang menemukannya. Permasalahan ini disebut juga "key distribution problem".

B. Algortima asimetris

Algoritma ini disebut asimetris karena kunci yang digunakan untuk enkripsi berbeda dengan kunci yang digunakan untuk dekripsi. Algoritma ini disebut juga algoritma kunci publik karena kunci yang digunakan untuk enkripsi adalah kunci publik atau *public key*. Sedangkan kunci untuk dekripsi menggunakan kunci pribadi atau *private key*. Contoh algoritma kriptografi simetris yaitu *RSA*, *ECC* dan lain-lainnya (Easttom, 2022).



Gambar 2.6 Alur proses algoritma asimetris

Berdasarkan Gambar 2.6 terdapat dua kunci yang digunakan algoritma asimetris yaitu (Easttom, 2022) :

1. Kunci publik adalah kunci yang diketahui atau dipublikasikan kepada semua orang. Pada proses enkripsi pesan, pengirim menggunakan kunci publik untuk menyampaikan pesan kepada penerima.
2. Kunci pribadi adalah kunci yang digunakan penerima pesan untuk membuka isi pesan yang diterima.

Kelebihan algoritma asimetris (Easttom, 2022):

- Masalah keamanan pada distribusi kunci dapat lebih baik
- Masalah manajemen kunci yang lebih baik karena jumlah kunci yang lebih sedikit

Kelemahan algoritma asimetris (Easttom, 2022):

- Kecepatan yang lebih rendah bila dibandingkan dengan algoritma simetris
- Untuk tingkat keamanan sama, kunci yang digunakan lebih panjang dibandingkan dengan algoritma simetris.

2.2.5 Algoritma *RSA*

Algortima *RSA* merupakan algoritma kunci publik yang paling popular saat ini. *RSA* ditemukan pertama kali pada tahun 1976 oleh 3 orang peneliti dari MIT yaitu Ron Rivest, Adi Shamir dan Leonard Aldeman. Algoritma memiliki keamanan yang terletak pada sulitnya memfaktorkan bilangan besar menjadi faktor-faktor prima. Perhitungan algoritma *RSA* didasarkan pada *teorema euler* (Merahe dkk., 2024).

Fungsi *totient euler*, juga dikenal sebagai fungsi *Euler Phi* dilambangkan dengan Φ(n), adalah jumlah elemen dalam himpunan yang mengurangi modulo sisa n. Dengan kata lain, Φ(n) adalah bilangan positif kurang dari n dan relatif prima terhadap bilangan bulat n (jika n lebih besar dari 1) (Almazari dkk., 2023).

Jika n adalah bilangan prima, maka Φ(n) = n-1. Jika n = pq, dimana p dan q adalah prima, maka Φ(n) = (p-1) (q-1). Berdasarkan sifat $a^k = b^k$ (mod n) untuk k bilangan bulat ≥ 1, maka persamaan dapat ditulis (Dossou-Yovo, Nitaj, & Togbé, 2024) :

$$a^{k\Phi(n)} = 1 \text{ (mod n)} \quad (2.1)$$

bila *a* diganti dengan palinteks *m*, maka persamaan 2.1 dapat ditulis

$$m^{k\Phi(n)} = 1 \text{ (mod n)} \quad (2.2)$$

Berdasarkan sifat *ac=bc (mod n)*, maka persamaan 2.2 dikali dengan m menjadi

$$m^{k\Phi(n)+1} = m \text{ (mod n)} \quad (2.3)$$

Misalkan e dan d dipilih sedemikian sehingga menjadi persamaan 2.4

$$e.d = 1(mod\ \Phi(n)) \quad (2.4)$$

Atau dapat juga ditulis dalam bentuk persamaan 2.5

$$e.d = k\,\Phi(n)+1 \quad (2.5)$$

Masukkan persamaan 2.5 ke dalam persamaan 2.3, sehingga membentuk persamaan 2.6.

$$m^{e.d} = m \text{ (mod n)} \quad (2.6)$$

Atau dapat juga ditulis dalam bentuk persamaan 2.7.

$$(m^e)^d = m \text{ (mod n)} \quad (2.7)$$

Berdasarkan persamaan 2.7, maka fungsi enkripsi (E) dan fungsi dekripsi (D) dirumuskan menjadi persamaan 2.8 dan persamaan 2.9.

$$E_e(m) = c = m^e \text{ mod n} \quad (2.8)$$

$$D_d(m) = m = c^d \text{ mod n} \quad (2.9)$$

Proses pembangkitan kunci *RSA* adalah sebagai berikut (Munir, 2019):

1. Pilih 2 buah bilangan prima p dan q. Untuk tujuan keamanan, bilangan *integer* p dan q dipilih secara *random*, dan harus memiliki panjang bit yang sama. Biasanya panjang bit yang dipilih untukbilangan p dan q ukurannya besar, agar kunci semakin aman.
2. Hitung n, yaitu $n = pq$. Bilangan n akan digunakan sebagai modulus untuk kunci publik
3. Hitung $\Phi(n) = (p-1)(q-1)$, dimana Φ adalah fungsi *totient Euler*.
4. Pilih bilangan *integer* e, dimana $1 < e < \Phi(n)$, fpb$(e, \Phi(n)) = 1$, e *dan* $\Phi(n)$ adalah *coprime* (e relatif prima terhadap $\Phi(n)$).Bilangan e adalah bilangan yang menjadi kunci publik.
5. Hitung kunci dekripsi d, dengan persamaan 2.10 sebagai berikut

$$ed = 1 \text{ (mod } \Phi(n)) \text{ atau } d = e^{-1} \text{ mod } (\Phi(n)) \quad (2.10)$$

Dari persamaan 2.10 didapatkan kunci publik dan kunci privat (Munir, 2019):

- Kunci publik adalah pasangan (e, n)
- Kunci privat adalah pasangan (d, n)

Proses enkripsi dan dekripsi algoritma *RSA* (Munir, 2019):

1. Nyatakan pesan menjadi blok-blok plainteks $m_1, m_2, m_3, \ldots, m_n$, dengan syarat $0 < m_i < n-1$.

2. Hitung blok chiperteks $c_i$untuk blok plainteks p_i dengan persamaan 2.11 sebagai berikut

   $$c = m^e \bmod n \quad (2.11)$$

   Dengan :

   m = blok *plaintext*

   n = p.q

Untuk proses dekripsi dapat menggunakan persamaan 2.12 sebagai berilut:

$$m = c^e \bmod n \quad (2.12)$$

Dengan :

m = blok *plaintext*

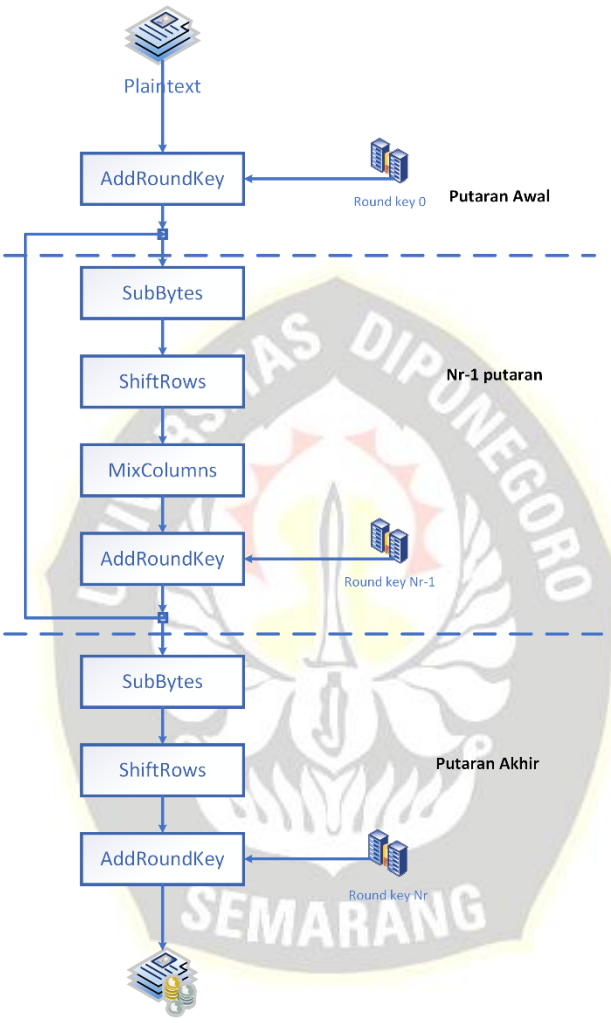
n = p.q

2.2.6 Algoritma *AES*

Algoritma Advanced Encryption Standard (*AES*) adalah algoritma simetris enkripsi yang diterbitkan oleh *US National Institute of Standards and Technology* (NIST) pada tahun 2000. Tujuan utama dari algoritma ini adalah untuk menggantikan algoritma DES setelah ditemukan memiliki kerentanan keamanan sebagai komputer menjadi lebih cepat. NIST mengundang para ahli yang bekerja di bidang enkripsi dan keamanan data dari seluruh dunia untuk mengembangkan algoritma *block cipher* yang mengenkripsi dan mendekripsi data lebih baik daripada algoritma DES (Waybhase & Adakane, 2022).

Setelah melalui beberapa rangkaian kriteria keamanan dan evaluasi parameter, NIST memilih salah satu algoritma kriptografi yang diusulkan oleh dua kriptografer Belgia, yaitu Joan Daeman dan Vincent Rijmen. Nama asli *AES* adalah algoritma Rijndael. Namun, nama ini tidak terlalu umum karena algoritma ini dikenal di seluruh dunia sebagai *Advanced Encryption Standard* (*AES*) (Abdullah, 2022).

Algoritma enkripsi *AES* membantu mengacak data sensitif, ke dalam format yang tidak dapat dibaca manusia tetapi dapat didekripsi. Proses dekripsi data hanya dapat dilakukan jika memiliki kunci yang benar (Scripcariu, 2018).



Gambar 2.7 Proses enkripsi *AES* (Munir, 2019)

Berdasarkan Gambar 2.7 proses enkripsi dapat diperinci sebagai berikut (Munir, 2019):

- Siapkan *array* berukuran 4x4 bernama Kunci.
- Siapkan *array* berukuran 4x4 bernama State.
- Masukkan kunci yang digunakan untuk enkripsi.
- Masukkan *plaintext* yang akan dienkripsi.

- Konversikan *plaintext* dan kunci enkripsi dari kode *ASCII* ke heksadesimal.
- Kelompokkan *bit-bit* teks tersebut menjadi 128-*bit* tiap bagiannya.
- Ambil 128-*bit* (16 karakter) *plaintext* pertama untuk diproses.
- Kelompokkan *bit* teks tersebut menjadi 16 bagian dengan 8-*bit* tiap bagiannya.
- Masukkan tiap-tiap bagian teks tersebut ke dalam tiap-tiap sel pada matriks berukuran 4x4.
- Konversikan *bit* ke dalam heksadesimal.
- *AddRoundKey*.

  Pada tahap ini, dilakukan operasi XOR antara *state matrix* dengan *key matrix* sehingga didapat *state matrix* yang baru. Pada akhir setiap *round*, juga dilakukan *AddRoundKey*.
- *SubBytes*.

  Merupakan tahapan dimana dilakukan substitusi *byte* menggunakan ketetapan dari tabel S-Box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

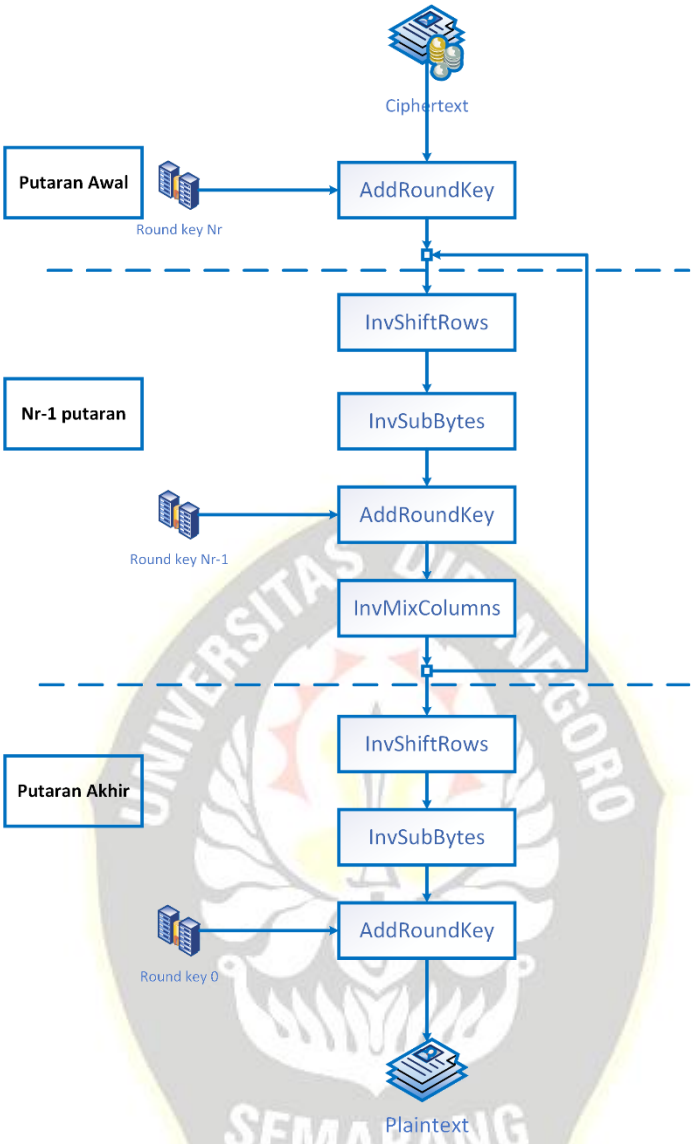
Gambar 2.8 *S-Box table* (Munir, 2019)

- ShiftRows

  ShiftRows dimana baris pertama tidak digeser, baris kedua digeser ke kiri 1 *byte*, baris ketiga digeser ke kiri 2 *byte*, dan baris keempat digeser ke kiri 3 *byte*.
- MixColumns

  Proses ini terdiri dari penyebaran (penggabungan) *byte* dari setiap kolom *array state*. Setiap kolom dipandang sebagai polinomial dalam persamaan *Galois Field* GF(2^8).
- AddRoundKey

  Langkah ini sama dengan langkah *addroundkey* sebelumnya. Perbedaannya ada pada matriks kunci. Setiap putaran memiliki matriks kunci yang berbeda, matriks kunci dibangkitkan melalui proses ekspansi kunci yang menghasilkan kunci untuk 10 putaran.
- Ulangi langkah *SubBytes. ShiftRows, MixColumn dan AddRoundKey* sebanyak 9 round.
- Pada round 10 hanya lakukan langkah *SubBytes, Shif Rows, dan AddRound Key*
- Setelah 10 putaran, maka didapatkan *ciphertext.*



Gambar 2.9 Proses dektipsi *AES* (Munir, 2019)

Berdasarkan Gambar 2.9 proses dekripsi dapat diperinci sebagai berikut (Munir, 2019):

- Siapkan *array* berukuran 4x4 bernama Kunci.
- Siapkan *array* berukuran 4x4 bernama State.
- Masukkan kunci yang digunakan untuk dekripsi.
- Masukkan *ciphertext* yang akan didekripsi.
- Konversikan *ciphertext* dan kunci dekripsi dari kode *ASCII* ke heksadesimal.

- Kelompokkan bit-bit teks tersebut menjadi 128-bit tiap bagiannya.
- Ambil 128-bit (16 karakter) *ciphertext* pertama untuk diproses.
- Kelompokkan bit teks tersebut menjadi 16 bagian dengan 8-bit tiap bagiannya.
- Masukkan tiap-tiap bagian teks tersebut ke dalam tiap-tiap sel pada matriks berukuran 4x4.
- Konversikan bit ke dalam heksadesimal.
- AddRoundKey.

  Pada tahap ini, dilakukan operasi XOR antara *state* matrix dengan key matrix sehingga didapat *state* matrix yang baru.
- InvShiftRows

  InvShiftRows mengembalikan matriks *state* dengan syarat baris pertama tidak digeser, baris kedua digeser ke kanan sebesar 1 *byte*, baris ketiga digeser ke kanan sebesar 2 *byte*, dan baris keempat digeser ke kanan sebesar 3 *byte*.
- *InvSubBytes*

  Merupakan tahapan dimana dilakukan substitusi *byte* menggunakan ketetapan dari tabel Reverse S-Box.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Gambar 2.10 *Reverse S-Box* (Munir, 2019)

- *AddRoundKey*

  Langkah ini sama dengan langkah addroundkey sebelumnya. Perbedaannya ada pada matriks kunci. Setiap putaran memiliki matriks kunci yang berbeda, matriks kunci dibangkitkan melalui proses ekspansi kunci yang menghasilkan kunci untuk 10 putaran.
- InvMixColumns

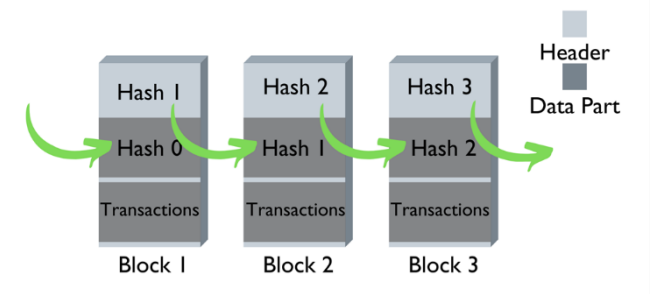
  Invmixcolumns merupakan proses mengembalikan nilai awal yang sebelumnya telah dilakukan dalam proses mixcolumns pada saat enkripsi.
- Ulangi langkah 12-15 sebanyak 9 round (putaran)
- Pada round 10 hanya lakukan langkah *InvShiftRows, InvSubBytes*, dan *AddRound Key.*
- Setelah 10 putaran, maka didapatkan *plaintext.*

2.2.7 Hashchain

Hashchain adalah teknik kriptografi untuk mengamankan data dengan menghasilkan serangkaian nilai *hash* terkait yang dihasilkan secara berurutan. Hash chain digunakan untuk melindungi data yang dikirimkan melalui jaringan komunikasi dengan mengenkripsi data menggunakan nilai *hash* sebagai kunci enkripsi (Qiuyuan Huang dkk., 2011). Fungsi *hash* H adalah sebuah algoritma yang menerima input sebuah pesan M, sebuah kunci dengan panjang tetap K dan mengeluarkan sebuah pesan dengan panjang tetap yang disebut D (intisari pesan) (Tiwari dkk., 2020). *Hahchain* dapat dibentuk dengan mengaplikasikan *one way hash function* h(.) sebagai *initial hash seed* secara rekursif seperti pada persamaan 2.13 (Mei, 2017).

$$h^N(s) = h(h(h \ldots h(s))) \text{ (N kali)} \quad (2.13)$$

Dalam persamaan 2.13 apabila nilai $h^N(s)$ diketahui tetapi nilai *hash seed* tidak diketahui maka penyerang tidak dapat mengetahui nilai $h^{N-1}(s)$ (Mei, 2017).



Gambar 2.11 Struktur *hashchain*

Dalam Gambar 2.11 ditunjukan struktur dari *hashchain*, nilai *hash* yang dihasilkan selama proses enkripsi digunakan sebagai kunci untuk proses enkripsi berikutnya begitu seterusnya hingga blok selesai. Penggunaan rantai *hash* diharapkan dapat menjaga kerahasiaan data yang dikirimkan meskipun jaringan komunikasi yang digunakan tidak aman (Qiuyuan Huang dkk., 2011).

2.2.8 Big O Notation

Big O notation merupakan notasi yang ditemukan Bachman pada tahun 1892. *Big O notation* digunakan untuk menggambarkan kompleksitas waktu atau ruang dari sebuah algoritma. *Big O notation* memiliki tujuan untuk memberikan perkiraan atas seberapa efisien algoritma tersebut dalam menangani input yang semakin besar. Dalam *Big O notation*, "O" mengacu pada orde pertumbuhan atau kompleksitas algoritma dalam hubungannya dengan ukuran input (Phalke dkk., 2022). Notasi ini menyatakan batas atas dari fungsi yang menggambarkan jumlah operasi atau memori yang diperlukan oleh algoritma, tergantung pada ukuran input (Chivers & Sleightholme, 2015).

Big O Notation mengkonversi total langkah algoritma dalam bentuk aljabar, dengan mengabaikan konstanta dan koefisien yang tidak memiliki dampak terhadap kompleksitas permasalahan yang diselesaikan oleh algoritma. Secara umum *Big O notation* diklasifikasikan sebagai berikut (Bae, 2019).

- **O(1) - Konstan:** Algoritma dengan kompleksitas **O(1)** akan eksekusi dalam waktu yang sama tidak peduli berapa banyak data yang diinputkan. Contoh: Mengakses elemen *array* berdasarkan indeks.

- **O(log n) - Logaritmik:** Waktu eksekusi meningkat logaritmik dengan penambahan jumlah input. Kompleksitas ini efektif untuk algoritma seperti pencarian biner.
- **O(n) - Linier:** Algoritma dengan kompleksitas **O(n)** akan meningkat eksekusinya secara linier berdasarkan ukuran input. Contoh: *Loop* sederhana melalui setiap elemen dalam *array*.
- **O(n log n) - Linier Logaritmik:** Algoritma *sorting* bekerja efektif dengan kompleksitas ini, seperti *mergesort* dan *heapsort*.
- **O(n^2) - Kuadratik:** Waktu yang dibutuhkan meningkat secara kuadratik dengan penambahan *input*. Contoh: *Nested loops* untuk setiap elemen *array*.
- **O(2^n) - Eksponensial:** Algoritma dengan kompleksitas ini memiliki peningkatan waktu eksekusi yang sangat tinggi bahkan dengan penambahan kecil pada ukuran *input*. Contoh: Algoritma pencarian rekursif pada kombinasi tertentu.
- **O(n!) - Faktorial:** Kompleksitas meningkat secara faktorial terhadap penambahan input. Contoh: Algoritma untuk menyelesaikan masalah *traveling salesperson* melalui *brute force*.

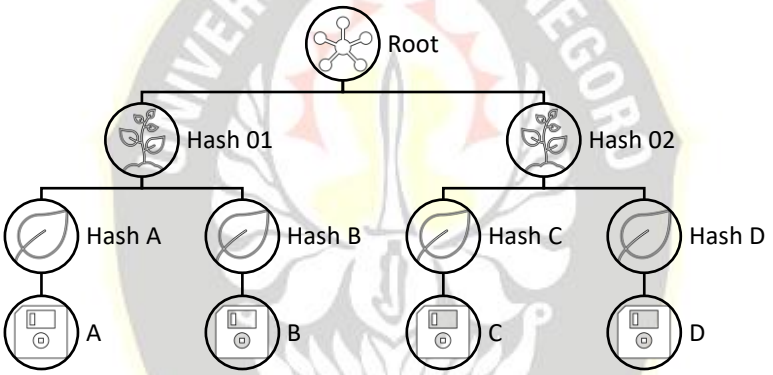
2.2.9. Merkle Tree

Merkle tree pertama kali diajukan oleh Merkle pada tahun 1979. Metode *merkle tree* membagi data menjadi blok-blok, melakukan penghitungan nilai hash untuk setiap blok, memasukkannya ke dalam *leaf node*, dan mengulangi peng-*hash*-an dalam bentuk pohon hash biner sampai nilai hash tunggal tercipta yang disebut *hash root*. Dalam hal ini, nilai *hash root* dapat digunakan sebagai tanda tangan. Dalam teknologi *blockchain*, metode *merkle tree* ini biasa digunakan untuk memeriksa integritas data blok (Lee & Park, 2021).

Merkle tree dapat digunakan untuk memeriksa integritas data besar. Artinya, *Merkle tree* dapat dibangun dengan membagi sejumlah besar data menjadi blok-blok dengan ukuran tertentu dan meletakkan blok-blok tersebut ke dalam *leaf node* yang sesuai. Jika data akan disinkronkan ke node lain, *Root merkle tree* akan diekstraksi dari data setelah transmisi data, dan integritas data yang ditransmisikan

dapat diperiksa dengan membandingkan nilai-nilainya. Jika terjadi kesalahan sekalipun hanya satu *byte* data karena kesalahan dari proses transmisi data, kesalahan tersebut dapat segera terdeteksi melalui perbandingan *Root merkle tree*. Dalam kasus ini, nilai hash dari semua data blok dibandingkan satu sama lain. Oleh karena itu, *merkle tree* digunakan untuk memeriksa integritas data dalam lingkungan *blockchain* umum (Lee & Park, 2021).

Merkle tree merupakan bagian esensial dari teknologi *blockchain*, dan merupakan struktur yang memungkinkan verifikasi yang aman terhadap stabilitas dan konten data. *Merkle tree* merupakan bagian yang termasuk dalam arsitektur *blockchain* dan tahapan *blockchain* untuk berbagi data yang aman dari satu pengguna ke pengguna lainnya (Rathee & Singh, 2021).



Gambar 2.12 Diagram *merkle tree*

Berdasarkan Gambar 2.12 dijelaskan bahwa:

- **Hash A**: adalah hash dari data A.
- **Hash B**: adalah hash dari data B.
- **Hash C**: adalah hash dari data C.
- **Hash D**: adalah hash dari data D.
- **Hash 01**: adalah hash dari penggabungan Hash A + Hash B.
- **Hash 02**: adalah hash dari penggabungan Hash C + Hash D.
- **Root**: adalah hash dari penggabungan Hash 01 + Hash 02.

Dari Gambar 2.12, *merkle tree* memudahkan perluasan data dan verifikasi keaslian data dengan efisien. Setiap kali ada perubahan, *merkle tree* dapat segera mendeteksi perubahan pada *merkle root*. *Merkle tree* sangat berguna pada aplikasi *blockchain*, di mana integritas data dan kemampuan audit adalah prioritas.

2.2.10. Pbkdf2

Pbkdf2 (Password-Based Key Derivation Function 2) merupakan salah satu algoritma derivasi kunci yang dirancang untuk mengubah kata sandi (*password*) menjadi kunci kriptografi yang aman. Algoritma ini termasuk dalam standar kriptografi yang ditetapkan oleh *RSA Laboratories* dalam dokumen *PKCS #5 v2.0* dan telah diadopsi secara luas dalam berbagai aplikasi keamanan informasi, termasuk penyimpanan kata sandi, enkripsi data, dan autentikasi (Visconti dkk., 2015).

Pbkdf2 bekerja dengan mengubah kata sandi yang dimasukkan oleh pengguna menjadi kunci kriptografi yang kuat melalui proses derivasi berbasis fungsi hash kriptografis. Proses ini dimulai dengan menggabungkan kata sandi dan salt, yaitu nilai acak yang ditambahkan. Kombinasi tersebut kemudian diproses menggunakan fungsi *pseudorandom* seperti *HMAC (Hash-based Message Authentication Code)* (Choi & Seo, 2021). Hasil dari proses ini diulang sebanyak jumlah iterasi yang telah ditentukan, di mana setiap iterasi bertujuan untuk meningkatkan kompleksitas komputasi dan memperlambat upaya *brute-force*. Setelah seluruh iterasi selesai, keluaran akhir berupa kunci derivatif dengan panjang tertentu yang dapat digunakan untuk keperluan enkripsi atau autentikasi (Iuorio dkk., 2019).