

# **TUGAS MATAKULIAH SISTEM KEAMANAN**

## **MENINGKATKAN KEAMANAN PESAN PADA APLIKASI CHATting DENGAN IMPLEMENTASI SHA-3 UNTUK AUTENTIKASI DAN INTEGRITAS DATA**



Diajukan untuk memenuhi sebagian persyaratan lulus  
Mata Kuliah Sistem Keamanan (INF11016)

Arya Winata / 2201020001

Luthfi Kurniawan / 2201020013

M. Zaidan Nugroho / 2201020011

Muhammad Afief Anugrah / 2201020015

Muhammad Gezza Riyan Tri Asmara / 2201020022

**JURUSAN TEKNIK INFORMATIKA  
FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN  
UNIVERSITAS MARITIM RAJA ALI HAJI  
TANJUNGPINANG  
GENAP 2023/2024**

## DAFTAR ISI

DAFTAR ISI .....	2
BAB I PENDAHULUAN .....	3
1.1. Latar Belakang .....	4
1.2. Rumusan Masalah .....	5
1.3. Batasan Masalah.....	5
1.4. Tujuan.....	5
1.5. Manfaat.....	5
BAB II KAJIAN LITERATUR .....	7
2.1. Kajian Pustaka.....	7
2.2. Landasan Teori.....	8
BAB III METODE PENELITIAN .....	<b>Error! Bookmark not defined.</b>
3.1. Alat dan Bahan .....	<b>Error! Bookmark not defined.</b>
3.2. Perancangan Sistem.....	<b>Error! Bookmark not defined.</b>
a. Gambaran Umum Sistem .....	<b>Error! Bookmark not defined.</b>
b. Perhitungan Manual .....	<b>Error! Bookmark not defined.</b>
BAB IV IMPLEMENTASI.....	<b>Error! Bookmark not defined.</b>
BAB V PENUTUP.....	<b>Error! Bookmark not defined.</b>
5.1. Kesimpulan.....	<b>Error! Bookmark not defined.</b>
5.2. Saran.....	<b>Error! Bookmark not defined.</b>
DAFTAR PUSTAKA.....	<b>Error! Bookmark not defined.</b>

## DAFTAR GAMBAR

Gambar 1 Use Case Diagram A .....	11
Gambar 2 Use Case Diagram B .....	12
Gambar 3 Perhitungan Manual A .....	13
Gambar 4 Perhitungan Manual B .....	13
Gambar 5 Perhitungan Manual C .....	14
Gambar 6 Perhitungan Manual D .....	14
Gambar 7 Perhitungan Manual E .....	14
Gambar 8 Perhitungan Manual F .....	14
Gambar 9 Perhitungan Manual G .....	15
Gambar 10 Perhitungan Manual H .....	15
Gambar 11 Perhitungan Manual I .....	15
Gambar 12 Perhitungan Manual J .....	15
Gambar 13 Perhitungan Manual K .....	15
Gambar 14 Perhitungan Manual L .....	15
Gambar 15 Perhitungan Manual M .....	16
Gambar 16 Perhitungan Manual N .....	16
Gambar 17 Inisialisasi Konstanta .....	17
Gambar 18 Inisialisasi Konstanta .....	17
Gambar 19 Inisialisasi Konstanta .....	18
Gambar 20 Hashing SHA-3 (Permutasi Keccak) .....	19
Gambar 21 Hashing SHA-3 (Permutasi Keccak) .....	20
Gambar 22 Hashing SHA-3 (Permutasi Keccak) .....	20
Gambar 23 Hashing SHA-3 (Permutasi Keccak) .....	20
Gambar 24 Hashing SHA-3 (Permutasi Keccak) .....	21
Gambar 25 Hashing SHA-3 (Permutasi Keccak) .....	21
Gambar 26 Hashing SHA-3 (Permutasi Keccak) .....	21
Gambar 27 Hashing SHA-3 (Permutasi Keccak) .....	22
Gambar 28 Implementasi .....	23
Gambar 29 Implementasi .....	23
Gambar 30 Implementasi .....	23
Gambar 31 Implementasi .....	24

# BAB I

## PENDAHULUAN

### 1.1. Latar Belakang

Sebagai makhluk sosial, manusia tidak bisa lepas dari berkomunikasi dengan manusia lainnya. Dengan berkembangnya zaman (Mohammed Banafaaa, 2023), teknologi kini memungkinkan kita untuk berkomunikasi dan berinteraksi dengan seseorang yang bahkan terpisah ribuan kilometer dari kita.

Aplikasi *chatting*, merupakan media teknologi yang memberikan kesempatan untuk kita mengirimkan sebuah pesan secara instan kepada seseorang. Sekarang, dengan kemudahan yang ditawarkan, aplikasi *chatting* kini sudah menjadi media komunikasi utama untuk semua orang (Ajay Kumar Sahu, 2023).

Namun, kemudahan dalam mengirimkan pesan jarak jauh secara *real-time* ini, membuka peluang dalam keamanan siber untuk hal hal seperti, pencurian data, sabotase, pembajakan, penipuan, dan sebagainya. Bahkan tidak jarang kita mendengar kabar mengenai penjualan data pribadi kita, atau tanpa kita inginkan, pesan-pesan kita tersebar ke orang yang tidak bertanggung jawab. Maka dari itu, keamanan sebuah pesan dalam aplikasi *chatting* menjadi semakin penting untuk menjaga pengalaman komunikasi dan interaksi yang baik (Pavankumar Patel, 2023).

Kriptografi menjadi semakin populer dikarenakan masalah tersebut, para peneliti berlomba menciptakan metode metode pengamanan informasi yang lebih mudah di aplikasikan (*low-cost*) namun, tetap menjaga integritasnya. Dalam aplikasi *chatting*, kita dapat mengaplikasikan sebuah algoritma autentikasi dalam setiap pesan yang dikirimkan (Aarti, 2023).

Melalui penelitian ini, kami menawarkan sebuah skema enkripsi menggunakan fungsi *hashing* yaitu SHA-3, Dibandingkan dengan metode *hashing* yang lainnya seperti MD5 atau SHA-1 dan SHA-3, SHA-3 memberikan pendekatan baru yang mampu memberikan setiap pesan sebuah identitas unik dengan ukuran yang pasti (*fixed-size*). Dengan SHA-3 kita dapat mendeterminasi keaslian, keutuhan, dan kepastian dari sebuah pesan yang dikirim. SHA-3 cukup terkenal dengan tingkat keamanan yang diberikan, efesiensi dari algoritmanya, serta fleksibilitas nya yang dapat di aplikasikan kedalam berbagai bentuk.

Dengan demikian, penelitian ini tidak hanya memberikan pandangan baru dalam mengamankan sebuah pesan di aplikasi *chatting*, tetapi juga berkontribusi dalam pengembangan ilmu pengetahuan data dan aplikasi teknologi dalam bidang Kriptografi.

## **1.2. Rumusan Masalah**

Berdasarkan studi masalah yang didapat, peneliti merumuskan sebuah skema enkripsi untuk meningkatkan keamanan pesan dalam aplikasi *chatting*.

## **1.3. Batasan Masalah**

Kami membatasi penelitian ini sampai kepada perancangan dan pengaplikasian konsep kriptografi dalam aplikasi *chatting* dengan:

- 1) Data yang digunakan dalam penelitian ini mencakup periode 2020-2024
- 2) Kami menggunakan metode *Hashing* yaitu SHA-3 dalam skema enkripsi yang kami buat
- 3) 'pesan' dalam aplikasi *chatting* merujuk pada informasi/pesan dalam bentuk teks
- 4) Aplikasi *chatting* yang kami gunakan, adalah aplikasi demo yang hanya digunakan untuk mempresentasikan skema enkripsi yang kami buat.
- 5) Laporan ini secara spesifik untuk membahas implementasi metode *Hashing* SHA-3 dalam melakukan verifikasi pesan di aplikasi *chatting*.

## **1.4. Tujuan**

Penelitian ini bertujuan untuk meningkatkan keamanan pesan dalam aplikasi *chatting* dengan menggunakan fungsi SHA-3. Adapun tujuan penelitian ini meliputi:

- 1) Mengembangkan sebuah model pesan dalam aplikasi *chatting* yang aman dengan menunjukkan keaslian, keutuhan, dan kepastiannya.
- 2) Menganalisis efektivitas dari fungsi SHA-3 kepada sebuah pesan dalam aplikasi *chatting*.
- 3) Mengembangkan skema enkripsi yang aman dan efisien untuk digunakan dalam sebuah aplikasi *chatting*.

## **1.5. Manfaat**

Penelitian ini memiliki manfaat yang meliputi:

- 1) Meningkatkan keamanan pesan dalam aplikasi *chatting*, sehingga dapat melindungi privasi dan informasi sensitif pengguna.
- 2) Memberikan solusi baru dalam mengamankan pesan di aplikasi *chatting* yang lebih aman dan efisien.

- 3) Berkontribusi dalam pengembangan ilmu pengetahuan dan aplikasi teknologi dalam bidang kriptografi.

## BAB II

### KAJIAN LITERATUR

#### 2.1. Kajian Pustaka

Sebagai bahan pertimbangan dalam penelitian ini, berikut beberapa penelitian terdahulu yang berkaitan

Yamamh Alaa, dkk. (2023) Dalam penelitian mereka yang berjudul “A Survey of Parallel Message Authentication and *Hashing* Methods” bertujuan untuk memberikan tinjauan komprehensif tentang metode autentikasi pesan dan *hashing* paralel, menganalisis kelebihan dan kekurangannya, serta membandingkan kinerja dan keamanannya. Penelitian ini melakukan tinjauan literatur terhadap berbagai metode autentikasi pesan dan *hashing* paralel, menganalisisnya berdasarkan kinerja, keamanan, dan kompleksitas implementasi. Hasil penelitian menunjukkan bahwa terdapat beberapa metode autentikasi pesan dan *hashing* paralel, termasuk Message Authentication Code (MAC) dan *hash* paralel. Penelitian ini memiliki beberapa keterbatasan, seperti fokusnya hanya pada metode autentikasi pesan dan *hashing* paralel dan tidak membahas implementasi metodenya.

Burhan ul Islam Khan, dkk. (2022) dalam penelitian mereka yang berjudul "Evolution And Analysis of Secured *Hash* Algoritm (SHA) Family" bertujuan untuk memberikan tinjauan komprehensif tentang evolusi dan analisis keluarga Secure *Hash* Algorithm (SHA), yang merupakan algoritma *hash* kriptografi yang banyak digunakan untuk berbagai aplikasi keamanan. Penelitian ini melakukan tinjauan literatur terhadap berbagai penelitian tentang SHA. Para penulis menganalisis sejarah SHA, termasuk SHA-1, SHA-2, dan SHA-3, serta menganalisis kekuatan dan kelemahannya, dan membandingkannya dengan algoritma *hash* lainnya. Hasil penelitian menunjukkan bahwa SHA telah mengalami evolusi yang signifikan selama bertahun-tahun. SHA-2 dan SHA-3 menawarkan keamanan dan kinerja yang lebih baik daripada SHA-1.

Samira Prabhune dan Sonal Sharma (2021) dalam penelitian mereka yang berjudul "End-to-End Encryption for Chat App with Dynamic Encryption Key" mengusulkan skema enkripsi end-to-end untuk meningkatkan keamanan dan privasi komunikasi dalam aplikasi chat. Skema ini menggunakan kunci enkripsi dinamis, yang artinya kunci tersebut dibuat secara acak untuk setiap pesan. Untuk mencapai keamanan yang lebih

baik, kunci enkripsi ini hanya dibagikan antara pengirim dan penerima pesan melalui saluran aman. Para peneliti mengimplementasikan skema ini pada sebuah aplikasi chat prototipe dan melakukan evaluasi terhadap keamanannya. Hasil evaluasi menunjukkan bahwa skema enkripsi *end-to-end* dengan kunci dinamis ini terbukti efektif dan aman. Skema ini menawarkan perlindungan yang kuat terhadap berbagai serangan, seperti intersepsi pesan dan serangan replay. Selain itu, skema ini juga efisien sehingga dapat diimplementasikan pada aplikasi chat praktis.

Debajyoti Halder, dkk. (2022) dalam penelitian mereka yang berjudul "fybrrChat: A Distributed Chat Application for Secure P2P Messaging" mengusulkan aplikasi chat terdistribusi yang aman untuk perpesanan P2P. Aplikasi ini menawarkan beberapa fitur utama, yaitu arsitektur terdistribusi tanpa server pusat yang menjamin privasi dan keamanan data pengguna. Enkripsi *end-to-end* yang mampu melindungi pesan dari intersepsi dan manipulasi. Berbagai file dimana memungkinkan pengguna untuk berbagi file dengan aman. Panggilan suara namun memungkinkan pengguna untuk melakukan panggilan suara dengan aman. Berdasarkan temuan penelitian ini, dapat disimpulkan bahwa fybrrChat merupakan solusi yang efektif untuk perpesanan P2P yang aman.

Fanny Ramadhan, dkk. (2020) dalam paper berjudul "Combination of Hybrid Cryptography In One Time Pad (OTP) Algorithm And Keyed-Hash Message Authentication Code (HMAC) In Securing The Whatsapp Communication Application" mengusulkan metode kriptografi hibrid untuk mengamankan komunikasi *WhatsApp*. Metode ini menggabungkan dua teknik kriptografi, yaitu *One-Time Pad (OTP)* yang secara teori menawarkan kerahasiaan sempurna, tetapi secara praktik tidak cocok untuk aplikasi perpesanan seperti *WhatsApp* karena membutuhkan kunci pra-bagi yang sama panjangnya dengan pesan. Lalu ada tektik *Keyed-Hash Message Authentication Code (HMAC)* yang menjamin integritas pesan dan autentikasi pengirim.

## 2.2. Landasan Teori

### 1) Keamanan siber

Keamanan siber adalah praktik melindungi sistem, jaringan, dan data dari akses, penggunaan, pengungkapan, gangguan, modifikasi, atau penghancuran yang tidak sah (Sharma, 2021). Keamanan siber penting untuk melindungi privasi, integritas, dan ketersediaan informasi.

### 2) Kriptografi



Kriptografi adalah ilmu dan seni untuk menjaga keamanan informasi dengan mengubahnya menjadi bentuk yang tidak terbaca. Algoritma *hashing* adalah salah satu teknik kriptografi yang digunakan untuk menghasilkan nilai *hash* dari suatu data (FIPS PUB 202). Nilai *hash* ini dapat digunakan untuk autentikasi data, integritas data, dan non-repudiation.

### **3) Aplikasi *Chatting***

Aplikasi *chatting* adalah aplikasi perangkat lunak yang memungkinkan pengguna untuk berkomunikasi satu sama lain secara real-time melalui jaringan internet. Aplikasi *chatting* banyak digunakan untuk komunikasi pribadi dan bisnis (Debajyoti Halder, 2022).

## BAB III

### METODE PENELITIAN

#### 3.1. Alat dan Bahan

a. Android Studio

Android Studio adalah IDE (*Integrated Developing Environment*) yang khusus digunakan dalam pengembangan aplikasi berbasis Android. Dengan menggunakan Android Studio, banyak hal penting yang dibutuhkan sebelum sebuah aplikasi Android dapat diprogram, yang dapat langsung tersedia.

b. Android Emulator

Untuk memudahkan proses pengembangan aplikasi Android, sebuah *Emulator* diperlukan untuk dapat menyaksikan perubahan yang *real-time* dan sesuai pada aplikasi kita dengan perangkat yang dituju.

c. Visual Code Studio

Editor kode open-source yang populer dan ringan. Mendukung berbagai bahasa pemrograman, dengan ekstensi yang dapat ditambahkan untuk meningkatkan fungsionalitas. Memiliki fitur seperti *debugger*, *linting*, dan *autocompletion*.

d. Bahasa Pemrograman Dart

Bahasa pemrograman yang digunakan untuk membangun aplikasi Flutter. Dikembangkan oleh Google dan memiliki sintaks yang mirip dengan JavaScript. Diketik secara statis, aman, dan performant.

e. Flutter CLI

Command-line interface (CLI) untuk membangun, menjalankan, dan menguji aplikasi Flutter. Memungkinkan kita untuk membangun sebuah aplikasi Android, menjalankan aplikasi di emulator atau perangkat fisik, dan menguji aplikasi dengan berbagai perintah.

f. Firebase

Platform *back-end-as-a-service* (BaaS) yang menyediakan berbagai layanan untuk aplikasi mobile. Layanan yang ditawarkan termasuk autentikasi, basis data, penyimpanan berkas, dan analitik. Firebase mudah digunakan dan terintegrasi dengan baik dengan aplikasi Flutter.

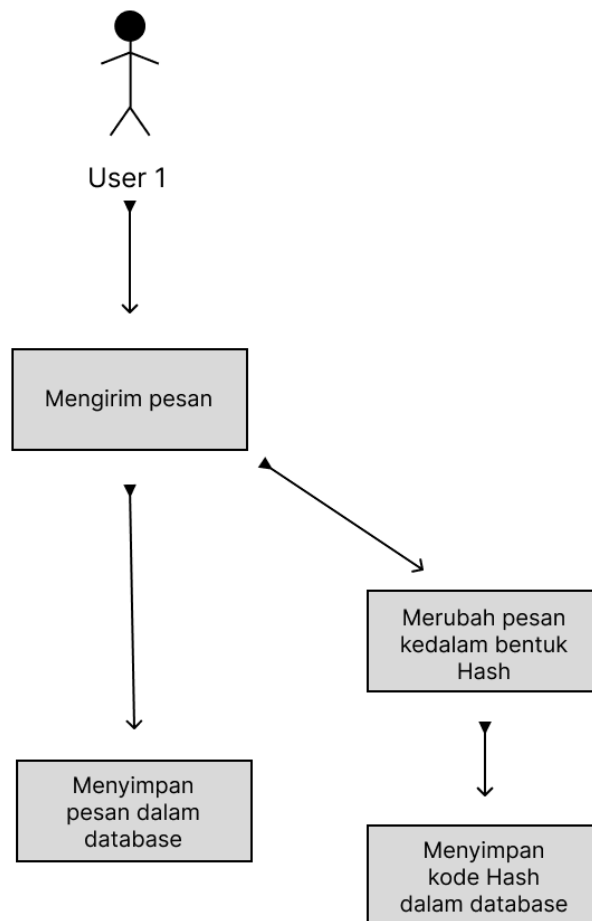
g. Firebase-Firestore

Layanan database *real-time* NoSQL yang disediakan oleh Firebase. Menyimpan data dalam format dokumen JSON dan memungkinkan sinkronisasi data real-time antar perangkat. Memiliki kueri data yang fleksibel dan mudah digunakan.

### 3.2. Perancangan Sistem

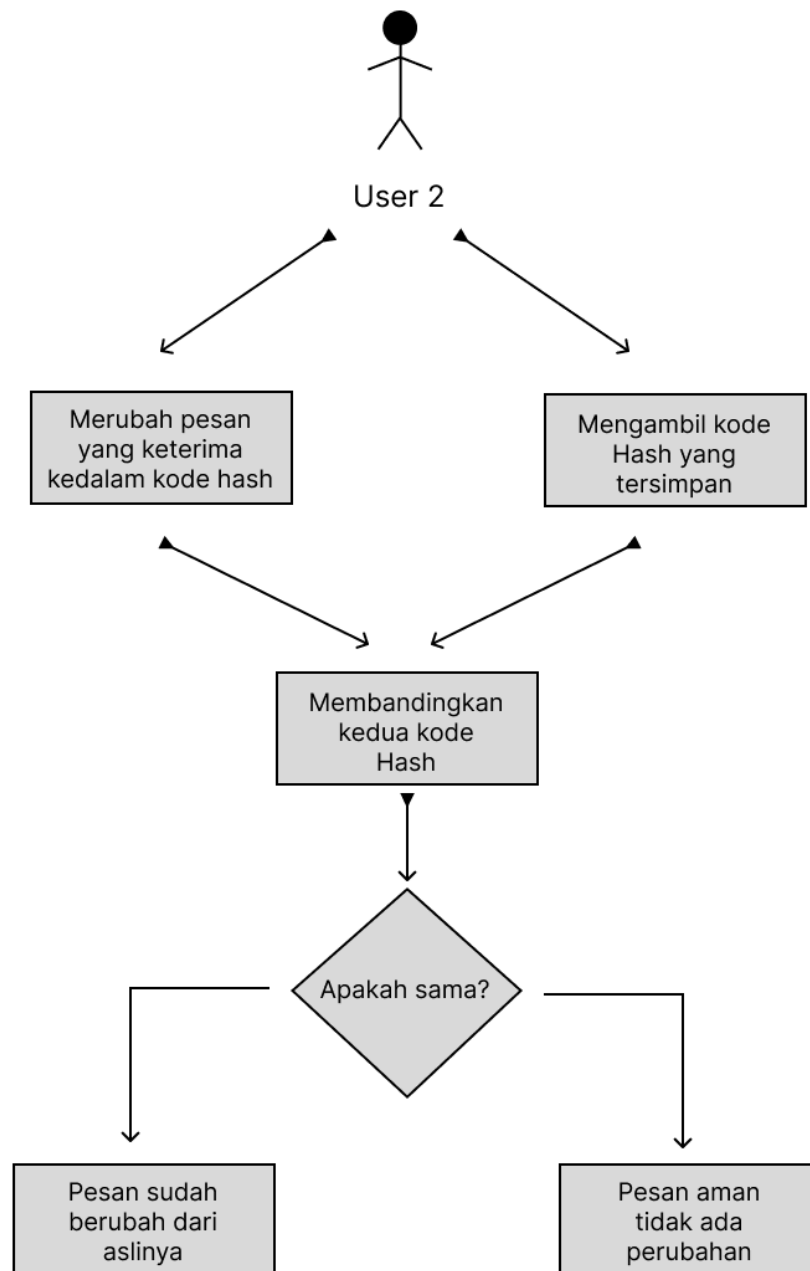
#### a. Gambaran Umum Sistem

Dalam sebuah aplikasi *chatting*, kegiatan mengirim dan menerima pesan antar pengguna menjadi proses yang paling penting, maka dari itu kami menitik beratkan sistem keamanan yang menggunakan fungsi *hashing* pada proses ini.



Gambar 1 Use Case Diagram A

Saat pengguna pertama mengirimkan sebuah pesan melalui aplikasi *chatting*, pesan tersebut akan di ambil terlebih dahulu kode *hash* nya (kode *hash* awal), kemudian kode *hash* tersebut akan disimpan secara terpisah.



Gambar 2 Use Case Diagram B

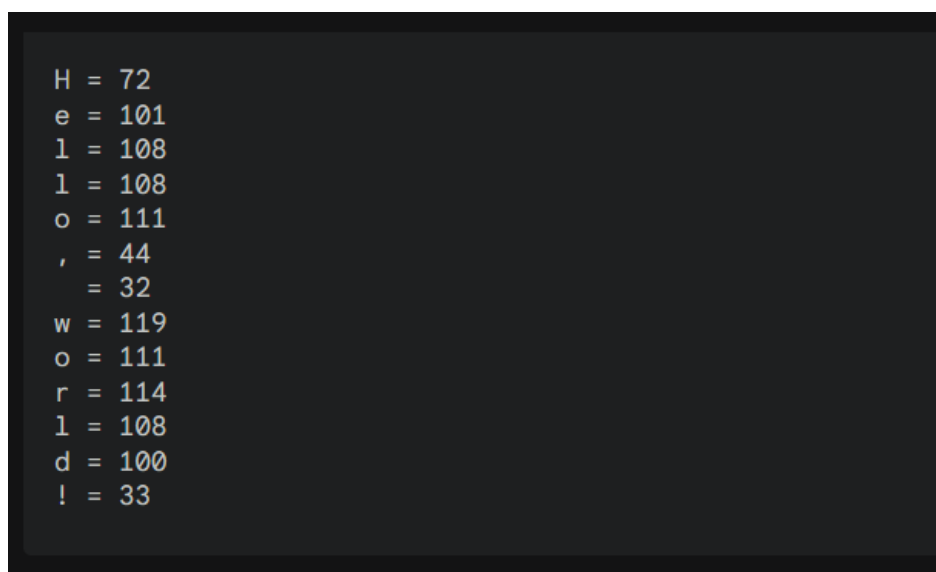
Kemudian dari pihak pengguna kedua sebagai penerima, pesan yang diterima akan di ambil kembali kode *hash* nya (kode *hash* akhir). Untuk dapat memastika pesan yang diterima aman dan tidak mengalami perubahan, kode *hash* awal yang sudah tersimpan akan di bandingkan dengan kode *hash* akhir.

## b. Perhitungan Manual

Berikut adalah perhitungan manual untuk *hashing* SHA-3 menggunakan algoritma Keccak dengan panjang *hash* 256 bit (SHA3-256) pada string "Hello, world!", Langkah-langkah nya sebagai berikut:

### 1. Persiapan:

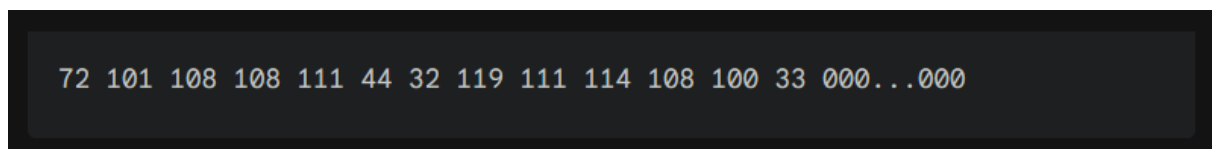
- Konversi string "Hello, world!" ke ASCII:



```
H = 72
e = 101
l = 108
l = 108
o = 111
, = 44
  = 32
w = 119
o = 111
r = 114
l = 108
d = 100
! = 33
```

Gambar 3 Perhitungan Manual A

- Tambahkan padding bit 1 ke akhir string, diikuti dengan 0 sebanyak diperlukan untuk membuat panjang total menjadi kelipatan 512 bit. Dalam kasus ini, tambahkan 1 bit dan 447 bit 0:



```
72 101 108 108 111 44 32 119 111 114 108 100 33 000...000
```

Gambar 4 Perhitungan Manual B

- Bagi string yang di-padding menjadi blok-blok 512 bit. Dalam kasus ini, hanya ada satu blok:

```
72 101 108 108 111 44 32 119 111 114 108 100 33 000...000
```

Gambar 5 Perhitungan Manual C

2. Inisialisasi Keadaan:

- Definisikan konstanta untuk nilai awal state (Keccak-f):

```
a0 = 0x67452301  
a1 = 0xEFCDAB89  
a2 = 0x98BADCFE  
a3 = 0x10325476  
a4 = 0xC3D2E1F0  
a5 = 0x76543210  
a6 = 0xFEDCBA98  
a7 = 0x89ABCDEF
```

Gambar 6 Perhitungan Manual D

- Buat array state 25 kata (5 kata per baris):

```
[a0, a1, a2, a3, a4]  
[a5, a6, a7, ..., ...]
```

Gambar 7 Perhitungan Manual E

3. Proses Kompresi:

- Ulangi langkah-langkah berikut untuk setiap blok 512 bit:
  - Bagi blok menjadi 16 kata 32 bit:

```
b0, b1, b2, ..., b15
```

Gambar 8 Perhitungan Manual F

- Lakukan operasi *XOR* pada state dan blok:

```
a0 = a0 ^ b0
a1 = a1 ^ b1
...
```

Gambar 9 Perhitungan Manual G

- Lakukan operasi round Keccak pada state:

```
round(a0, a1, a2, ..., a24)
```

Gambar 10 Perhitungan Manual H

- Operasi round terdiri dari beberapa langkah:

- Theta:

```
c = a0 ^ a5 ^ a10 ^ a15 ^ a20
d = a1 ^ a6 ^ a11 ^ a16 ^ a21
...
```

Gambar 11 Perhitungan Manual I

- Rho:

```
a0 = rotate(a0, 1)
a1 = rotate(a1, 3)
...
```

Gambar 12 Perhitungan Manual J

- Pi:

```
a0 = a0 ^ a1 ^ a2 ^ ... ^ a24
```

Gambar 13 Perhitungan Manual K

- Chi:

```
a0 = a0 ^ ((a1 & a2) | (~a1 & a3))
a1 = a1 ^ ((a2 & a3) | (~a2 & a4))
...
```

Gambar 14 Perhitungan Manual L

4. Finalisasi:

- Hitung nilai *hash* dari state:

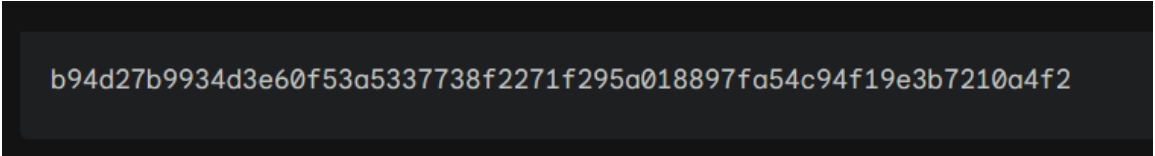


```
hash = a0 || a1 || a2 || ... || a24
```

Gambar 15 Perhitungan Manual M

- Hasil:

*Hash* SHA3-256 dari string "Hello, world!" adalah:



```
b94d27b9934d3e60f53a5337738f2271f295a018897fa54c94f19e3b7210a4f2
```

Gambar 16 Perhitungan Manual N



## BAB IV

### IMPLEMENTASI

#### A. Inisialisasi Konstanta

```
static const int _blockSize = 136;
```

Gambar 17 Inisialisasi Konstanta

Dalam algoritma SHA-3, pesan dibagi menjadi blok-blok yang kemudian diolah secara terpisah. Ukuran blok untuk SHA-3-256 adalah 136 byte.

```
static const List<int> _roundConstants = [  
    0x0000000000000001,  
    0x0000000000000082,  
    0x800000000000008a,  
    0x8000000080008000,  
    0x000000000000808b,  
    0x0000000080000001,  
    0x8000000080008081,  
    0x8000000000008009,  
    0x000000000000008a,  
    0x0000000000000088,  
    0x0000000080008009,  
    0x000000008000000a,  
    0x000000008000808b,  
    0x800000000000008b,  
    0x8000000000008089,  
    0x8000000000008003,  
    0x8000000000008002,  
    0x8000000000000080,  
    0x000000000000800a,  
    0x800000008000000a,  
    0x8000000080008081,  
    0x8000000000008080,  
    0x0000000080000001,  
    0x8000000080008008  
];
```

Gambar 18 Inisialisasi Konstanta

Dalam algoritma Keccak, konstanta putaran dipilih dengan hati-hati untuk mencapai sifat keamanan yang diinginkan serta untuk memastikan kecepatan dan efisiensi operasi. Nilai-nilai ini telah dipilih berdasarkan analisis kriptografis yang mendalam dan pengujian.

Penggunaan nilai konstan tersebut adalah bagian integral dari desain algoritma, dan menggantinya dengan nilai yang berbeda tanpa pemahaman yang mendalam tentang desain algoritma dapat menghasilkan hasil yang tidak diinginkan dan bahkan bisa mengorbankan keamanan.

Dengan demikian, penting untuk menggunakan nilai konstan yang telah ditentukan dalam implementasi SHA-3.

Offset putaran (rotation offsets) dalam fungsi *hashing* SHA-3 adalah daftar nilai yang menentukan jumlah pergeseran bit untuk setiap nilai dalam array selama tahap rotasi ( $\rho$ ). Pada setiap putaran, nilai-nilai dalam array keadaan dirotasi ke kiri sesuai dengan offset yang ditentukan dalam daftar ini.

Dalam implementasi SHA-3, terdapat 25 nilai offset rotasi yang digunakan dalam proses perhitungan *hash*. Offset ini menentukan bagaimana setiap nilai dalam array keadaan akan digeser selama tahap rotasi dalam fungsi keccakF.

```
static const List<int> _rotationOffsets = [  
    0,  
    36,  
    3,  
    41,  
    18,  
    1,  
    44,  
    10,  
    45,  
    2,  
    62,  
    6,  
    43,  
    15,  
    61,  
    28,  
    55,  
    25,  
    21,  
    56,  
    27,  
    20,  
    39,  
    8,  
    14  
];
```

Gambar 19 Inisialisasi Konstanta

Dalam kode ini, setiap nilai dalam daftar menunjukkan jumlah pergeseran bit yang diperlukan untuk nilai yang sesuai dalam array keadaan selama putaran tertentu. Misalnya, nilai pertama (0) menunjukkan bahwa tidak ada pergeseran yang diperlukan untuk nilai pertama dalam array keadaan, sedangkan nilai kedua (36) menunjukkan bahwa nilai kedua akan digeser ke kiri sebanyak 36 bit.

Dengan menggunakan offset rotasi ini, fungsi keccakF mengubah keadaan internal dalam algoritma *hashing* SHA-3 setiap kali putaran dilakukan, sehingga menghasilkan pencampuran (diffusion) yang diperlukan untuk mencapai keamanan kriptografi yang diinginkan.

## B. *Hashing* SHA-3 (Permutasi Keccak)

Dalam proses penghitungan *hash* menggunakan fungsi keccakF, terdapat beberapa langkah yang dijalankan secara berurutan. Setiap langkah tersebut memiliki peran masing-masing dalam mengubah keadaan internal dari nilai-nilai yang diolah.

```
static List<int> _keccakF(List<int> state) {  
    final List<int> output = List<int>.filled(state.length, 0);  
    final List<int> lanes = List<int>.filled(25, 0);  
  
    for (var round = 0; round < 24; round++) {  
        _theta(state, lanes);  
        _rho(state, lanes);  
        _pi(state, lanes);  
        _chi(state, lanes);  
        _iota(state, round);  
    }  
  
    return state;  
}
```

Gambar 20 Hashing SHA-3 (Permutasi Keccak)

Dalam konteks fungsi *hashing* SHA-3, "lane" merujuk pada nilai-nilai yang disimpan dalam array keadaan (state array) yang digunakan selama proses *hashing*. Dalam implementasi, lane direpresentasikan oleh bilangan bulat (integer), dan sejumlah lane membentuk array keadaan. Misalnya, dalam implementasi SHA-3 ini, terdapat 25 lane yang membentuk array keadaan.

State Array: Ini adalah struktur data yang digunakan untuk menyimpan nilai-nilai dalam proses *hashing* SHA-3. Array keadaan terdiri dari sejumlah lane, dan setiap lane menyimpan nilai yang merupakan bagian dari keadaan internal yang sedang diproses. Dalam implementasi SHA-3, array keadaan memiliki ukuran tetap dan disusun sebagai matriks 5x5, di mana setiap "sel" dalam matriks tersebut adalah satu lane.

```

static void _theta(List<int> state, List<int> lanes) {
    for (var i = 0; i < 5; i++) {
        lanes[i] = state[i] ^
            state[i + 5] ^
            state[i + 10] ^
            state[i + 15] ^
            state[i + 20];
    }

    for (var i = 0; i < 5; i++) {
        final T = lanes[(i + 4) % 5] ^ _rotateLeft(lanes[(i + 1) % 5], 1);
        for (var j = 0; j < 25; j += 5) {
            state[j + i] ^= T;
        }
    }
}

```

Gambar 21 Hashing SHA-3 (Permutasi Keccak)

Theta (\_theta): Langkah ini bertujuan untuk menghasilkan efek difusi dengan menggabungkan nilai-nilai dari beberapa lane. Ini dilakukan dengan melakukan operasi XOR pada nilai-nilai yang berbeda dari beberapa lane.

```

static void _rho(List<int> state, List<int> lanes) {
    for (var i = 0; i < 25; i++) {
        lanes[i] = _rotateLeft(state[i], _rotationOffsets[i]);
    }
}

```

Gambar 22 Hashing SHA-3 (Permutasi Keccak)

Rho (\_rho): Langkah ini bertanggung jawab untuk melakukan rotasi pada nilai-nilai dalam array keadaan berdasarkan offset rotasi yang ditentukan. Rotasi dilakukan untuk menyebarkan informasi ke seluruh array keadaan dan memperkenalkan tingkat keacakan yang diperlukan.

```

static void _pi(List<int> state, List<int> lanes) {
    for (var i = 0; i < 25; i++) {
        state[i] = lanes[_piMapping[i]];
    }
}

```

Gambar 23 Hashing SHA-3 (Permutasi Keccak)

Pi (\_pi): Langkah ini melakukan permutasi (pemetaan) pada posisi-nilai dalam array keadaan. Hal ini membantu dalam memperkenalkan ketidakdugaan dalam perubahan nilai-nilai dalam array keadaan.

```
static void _chi(List<int> state, List<int> lanes) {
    for (var i = 0; i < 25; i += 5) {
        for (var j = 0; j < 5; j++) {
            lanes[i + j] =
                state[i + j] ^ ((state[i + j + 1] ^ 1) & state[i + j + 2]);
        }
    }
}
```

Gambar 24 Hashing SHA-3 (Permutasi Keccak)

Chi (\_chi): Langkah ini bertujuan untuk menghasilkan efek difusi lebih lanjut dengan melakukan operasi *XOR* yang kompleks pada nilai-nilai dalam setiap lane.

```
static void _iota(List<int> state, int round) {
    state[0] ^= _roundConstants[round];
}
```

Gambar 25 Hashing SHA-3 (Permutasi Keccak)

Iota (\_iota): Langkah ini menambahkan konstanta putaran ke nilai spesifik dalam array keadaan. Ini bertujuan untuk memperkenalkan keacakan yang diperlukan pada setiap putaran.

```
static List<int> _pad(UInt8List message) {
    final blockSize = blockSize ~/ 8;
    final messageLength = message.length;
    final remainder = messageLength % blockSize;
    final padLength = remainder == 0 ? blockSize : blockSize - remainder;
    final padded = UInt8List(messageLength + padLength);
    padded.setRange(0, messageLength, message);

    // padding
    padded[messageLength] = 0x06; // append 0110 0000
    padded[padded.length - 1] |= 0x80; // append 1000 0000
    return padded;
}
```

Gambar 26 Hashing SHA-3 (Permutasi Keccak)

Padding adalah proses menambahkan bit-bit tambahan ke blok data agar ukurannya sesuai dengan ukuran blok yang diharapkan oleh algoritma *hashing*. Dalam konteks SHA-3, padding dilakukan untuk memastikan bahwa panjang blok data yang diinputkan memenuhi kebutuhan algoritma (misalnya, ukuran blok 136 byte). Padding ini biasanya dilakukan dengan menambahkan bit-bit tertentu pada akhir blok data.

```

static List<int> _digest(String message) {
    final paddedMessage = _pad(UInt8List.fromList(message.codeUnits));
    final state = List<int>.filled(200, 0);

    for (var offset = 0; offset < paddedMessage.length; offset += 136) {
        final blockEnd = offset + 136 <= paddedMessage.length
            ? offset + 136
            : paddedMessage.length;
        final block = paddedMessage.sublist(offset, blockEnd);

        for (var i = 0; i < block.length; i++) {
            state[i] ^= block[i]; // Use 'i' directly, not 'i ~/ 8'
        }

        _keccakF(state);
    }

    return state.sublist(0, 32);
}

```

Gambar 27 Hashing SHA-3 (Permutasi Keccak)

fungsi `_digest` adalah fungsi utama dalam implementasi fungsi *hashing* SHA-3 yang bertanggung jawab atas pengolahan blok data dan menghasilkan nilai *hash* akhir. Fungsi ini memanggil langkah-langkah utama dari algoritma *hashing* SHA-3, termasuk *theta*, *rho*, *pi*, *chi*, dan *iota*, untuk setiap blok data yang diberikan.

1. Pertama, blok pesan (message block) di-pad agar sesuai dengan ukuran blok yang diharapkan oleh algoritma *hashing* SHA-3. Ini dilakukan dengan memanggil fungsi `_pad` yang menambahkan padding pada akhir pesan.
2. Selanjutnya, sebuah array keadaan (state array) dibuat dengan ukuran yang sesuai untuk menyimpan nilai-nilai keadaan internal selama proses *hashing*. Dalam implementasi ini, array keadaan memiliki panjang 200, karena panjang internal array state setiap 64-bit lane sebesar 25, maka totalnya menjadi 200.
3. Blok-blok pesan yang di-pad kemudian diproses satu per satu dalam loop. Setiap blok pesan di-*XOR*-kan dengan nilai-nilai dalam array keadaan menggunakan operasi *XOR* bit demi bit.
4. Setiap blok pesan yang di-*XOR* -kan kemudian diolah melalui fungsi `_keccakF`, yang merupakan implementasi dari langkah-langkah *theta*, *rho*, *pi*, *chi*, dan *iota* dari algoritma *hashing* SHA-3.
5. Setelah semua blok pesan diproses, nilai *hash* akhir diambil dari array keadaan dan dikembalikan sebagai hasil fungsi.

### C. Implementasi

Setelah itu semua, fungsi *hashing* SHA-3 dapat diimplementasikan ke dalam aplikasi *chatting*, dengan menggunakan pesan yang dikirim oleh pengguna sebagai masukan pada fungsi `_digest`, kita akan mendapatkan nilai *hash* dari pesan tersebut.

```
void sendMessage() async {  
    // send message only if the message not empty  
    if (_messageController.text.isNotEmpty) {  
        String message = _messageController.text;  
        String hash = SHA3.hash(message);  
    }  
}
```

Gambar 28 Implementasi

Kemudian setelah itu, pesan akan dikirimkan beserta kode hash dari pesan tersebut ke penerima.

```
await _chatService.sendMessage(widget.receiverUserID, message, hash);
```

Gambar 29 Implementasi

Jika sudah seperti itu, langkah selanjutnya kami hanya perlu untuk melakukan verifikasi atas pesan yang dikirim dan pesan yang diterima akhirnya. Hal ini dilakukan dengan membandingkan kode *hash* yang tersimpan, dengan kode *hash* dari pesan yang diterima.

```
bool verifikasi(String receivedMessage, String receivedHash) {  
    String calculatedHash =  
        hashReceived(receivedMessage); // Hitung hash pesan yang diterima  
    return calculatedHash ==  
        receivedHash; // Periksa apakah hash yang diterima sama dengan hash yang dikirimkan  
}  
  
String hashReceived(String receivedMessage) {  
    return SHA3.hash(receivedMessage); // Menghitung hash pesan yang diterima  
}
```

Gambar 30 Implementasi

Untuk melakukan uji coba atas sistem enkripsi yang kami buat, kami membuat suatu logika, dimana pesan akan memiliki kemungkinan untuk berubah nilainya saat dikirimkan. Nantinya pesan yang ter-modifikasi, akan di tandai dengan blok pesan berwarna kuning.



**Gambar 31 Implementasi**



## **BAB V**

### **PENUTUP**

#### **5.1. Kesimpulan**

*Hashing* merupakan metode enkripsi satu arah, yang memungkinkan kita untuk memiliki sebuah kode yang sulit sekali untuk dipecahkan. Ada banyak sekali implementasi dari metode *hashing*, contohnya seperti melakukan verifikasi pesan dalam aplikasi *chatting*, dengan begitu kita akan memiliki sebuah standar model pesan yang lebih aman.

Dalam penelitian ini kami menggunakan SHA-3, yang mana merupakan standarisasi algoritma *hashing* yang sangat efektif, cepat, dan aman. Algoritma ini melibatkan banyak sekali proses yang bertujuan untuk melakukan difusi, pemetaan, pengacakan, permutasi, dan sebagainya.

#### **5.2. Saran**

Saran kami untuk penelitian selanjutnya, Ada banyak sekali letak pengaplikasian metode *hashing* SHA-3 dalam aplikasi *chatting*. Metode ini sangat efektif, tetapi disatu sisi juga membutuhkan ketelitian saat menyusun algoritma *Keccak*.

Metode *hashing* SHA-3 dapat meningkatkan kualitas keamanan yang ada dimasyarakat, contohnya seperti yang kami lakukan pada penelitian ini, aplikasi *chatting* menjadi media utama bagi masyarakat saat ini dalam berbagi/bertukar informasi.

## DAFTAR PUSTAKA

- Aarti, U. C. (2023). VAARTA: A Secure Chatting Application Using Firebase. *Springger Link*.
- Ajay Kumar Sahu, V. V. (2023). WEB- BASED CHAT APPLICATION USING REACT. *SSRN*.
- Burhan ul Islam Khan, R. F. (2022). EVOLUTION AND ANALYSIS OF SECURE HASH ALGORITHM (SHA) FAMILY. *Malaysian Journal of Computer Science*.
- Debajyoti Halder, S. B. (2022). fybrChat: A Distributed Chat Application for Secure P2P Messaging. *arXiv*.
- Mohammed Banafaaa, I. S. (2023). 6G Mobile Communication Technology: Requirements, Targets, Applications, Challenges, Advantages, and Opportunities. *ScienceDirect*.
- Pavankumar Patel, I. B. (2023). ChatApp with Encryption using Firebase. *Authorea Preprint*.
- Sharma, S. P. (2021). End-to-End Encryption for Chat App with Dynamic Encryption Key. *IEEE*.
- Yamamh Alaa, A. F. (2023). A Survey of Parallel Message Authentication and Hashing Method. *JUBPAS*.