

SKRIPSI

ANALISA METODE PENGINGAT *PASSWORD* DENGAN
SECRET SHARING SHAMIR



SAMUEL CHRISTIAN

NPM: 2011730002

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2015

UNDERGRADUATE THESIS

PROTECTING PASSWORD WITH PERSONAL ENTROPY



SAMUEL CHRISTIAN

NPM: 2011730002

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2015**

ABSTRAK

Otentikasi adalah proses untuk menentukan keaslian identitas dari entitas saat akan mengakses sumber daya sebuah sistem. Entitas yang diotentikasi dapat berupa manusia atau pengguna sistem. Sistem yang hendak diakses dapat berupa media sosial, *email*, *electronic banking*, dan sebagainya.

Salah satu metode otentikasi adalah *password*. *Password* digunakan untuk mengakses sumber daya sebuah sistem. Kebutuhan akan sumber daya tidak hanya bergantung pada satu sistem saja. Karena itu, untuk memenuhi kebutuhan akan sumber daya, diperlukan akses ke banyak sistem. Dengan diperlukannya akses ke banyak sistem, maka membutuhkan banyak *password* untuk masing-masing sistem.

Permasalahan akan muncul jika salah satu dari banyak *password* ini hilang, maka otomatis akses kepada sistem tertentu akan hilang juga. Beberapa sistem memiliki mekanisme dengan menyediakan pertanyaan keamanan yang harus dijawab untuk bisa mengembalikan *password*. Pada penelitian ini, akan dikembangkan mekanisme untuk mengembalikan n *password* dengan menyediakan n pertanyaan keamanan. Mekanisme ini akan menggunakan metode *secret sharing* Shamir dengan membagi setiap *password* menjadi beberapa bagian dan membuat pertanyaan keamanan untuk masing-masing *password*.

Untuk mengetahui apakah mekanisme mengembalikan n *password* dengan menyediakan n pertanyaan keamanan dengan menggunakan metode *secret sharing* Shamir akan lebih baik dalam melindungi *password*, maka akan dilakukan pembangunan perangkat lunak yang mengimplementasikan *secret sharing* Shamir dan pengujian terhadap perangkat lunak yang dibangun. Selain itu, akan dilakukan juga pengujian dengan metode survei untuk mengetahui pengaruh dari pertanyaan keamanan terhadap metode *secret sharing* Shamir dalam mengembalikan n *password* ini.

Selain itu, untuk menjaga kerahasiaan *password* dan jawaban dari masing-masing pertanyaan keamanan, metode *secret sharing* Shamir akan dikombinasikan dengan enkripsi dan fungsi *hash*. Teknik enkripsi yang akan digunakan adalah *Data Encryption Standard* dan algoritma fungsi *hash* yang akan digunakan adalah *Secure Hashing Algorithm* 512.

Berdasarkan hasil pengujian, pertanyaan keamanan memiliki pengaruh terhadap mekanisme mengembalikan banyak *password*. Dengan membuat pertanyaan keamanan yang tepat, *password* bisa dengan mudah dikembalikan oleh pemilik *password* dan juga bisa mempersulit pihak selain pemilik *password* untuk mengembalikan *password*.

Kata-kata kunci: Otentikasi, *Password*, Pertanyaan Keamanan, *Secret Sharing* Shamir

ABSTRACT

Authentication is the process of confirming the truth of an entity trying to access system resources. An entity can be human or system user and a system can be a social media system, email system, electronic banking system, and etc.

Password is one of the techniques to authenticate an entity. Password is used before an entity access system resources. The need of access to system resources does not rely on just a certain system only. To fulfill the need of access to system resources, an entity need to have many access to a lot of systems. By this way, an entity must have password for each system he/she has access to.

The problem will appear if one of these passwords is missing. User will lost access to system resources for this certain password. Some of the systems have mechanism that can retrieve the missing password by answering a security questions. In this research, a new mechanism will be developed to retrieve n passwords by creating n security questions. This mechanism will use secret sharing Shamir to divide the each password into shares and creating security questions for each password.

To find out if this developed mechanism that retrieve n passwords by providing n security questions using secret sharing Shamir will perform better in protecting passwords, we will develop a software which implementing secret sharing Shamir. Some tests will also be done to ensure this software development success. Furthermore, some survey tests will also be done to find out security questions effect on secret sharing Shamir in retrieving n passwords.

Moreover, to ensure the secrecy of the passwords and the secrecy of each security question, secret sharing Shamir will be combined with encryption technique and cryptographic hash function. Data Encryption Standard will be used as encryption technique and Secure Hashing Algorithm 512 will be used as cryptographic hash function.

According to the test results, security questions have influence of retrieving many passwords mechanism. By creating appropriate security questions, passwords can be easily retrieved by passwords' owner and can make other parties difficult in retrieving passwords.

Keywords: Authentication, Password, Security Questions, Shamir's Secret Sharing

DAFTAR ISI

| | |
|---|----------|
| DAFTAR ISI | ix |
| DAFTAR GAMBAR | xi |
| DAFTAR TABEL | xiii |
| 1 PENDAHULUAN | 1 |
| 1.1 Latar Belakang | 1 |
| 1.2 Rumusan Masalah | 2 |
| 1.3 Tujuan | 2 |
| 1.4 Batasan Masalah | 2 |
| 1.5 Metodologi Penelitian | 3 |
| 1.6 Sistematika Pembahasan | 3 |
| 2 DASAR TEORI | 5 |
| 2.1 Kriptografi | 5 |
| 2.1.1 Sejarah Kriptografi | 5 |
| 2.1.2 Pengertian Kriptografi | 5 |
| 2.2 Kerahasiaan (<i>Confidentiality</i>) | 6 |
| 2.3 <i>Data Encryption Standard</i> (DES) | 7 |
| 2.3.1 Sejarah | 7 |
| 2.3.2 Struktur DES | 7 |
| 2.3.3 Permutasi Awal | 7 |
| 2.3.4 Pembangunan Kunci Putaran | 8 |
| 2.3.5 Putaran | 11 |
| 2.3.6 Permutasi Akhir | 15 |
| 2.4 Fungsi <i>Hash</i> | 15 |
| 2.5 <i>Secure Hashing Algorithm</i> 512 (SHA-512) | 16 |
| 2.5.1 <i>Message Padding</i> | 16 |
| 2.5.2 Inisialisasi Konstanta Awal | 17 |
| 2.5.3 Ekspansi Blok Message | 17 |
| 2.5.4 Fungsi Kompresi dan Putaran | 18 |
| 2.6 Otentikasi | 22 |
| 2.6.1 <i>Password</i> | 23 |
| 2.7 Eliminasi Gauss-Jordan | 25 |
| 2.7.1 Proses Reduksi Matriks | 26 |
| 2.7.2 Proses Substitusi Balik | 27 |
| 2.8 <i>Secret Sharing</i> Shamir | 28 |
| 2.8.1 Sejarah Singkat | 28 |
| 2.8.2 Pembahasan <i>Secret Sharing</i> Shamir | 28 |
| 2.9 Probabilitas | 30 |
| 2.9.1 Distribusi Binom | 30 |
| 2.10 Entropi | 31 |

| | | |
|----------|---|-----------|
| 2.10.1 | Sejarah Singkat | 31 |
| 2.10.2 | Pembahasan | 31 |
| 3 | ANALISIS | 33 |
| 3.1 | Studi Kasus | 33 |
| 3.1.1 | Pengenalan Kasus | 33 |
| 3.1.2 | Proses Penyimpanan <i>Password</i> | 33 |
| 3.1.3 | Proses Pengembalian <i>Password</i> | 37 |
| 3.2 | Pemilihan Nilai n dan k | 42 |
| 3.2.1 | Pemilihan Nilai k | 42 |
| 3.2.2 | Pemilihan Nilai n | 43 |
| 3.3 | Analisis Proses | 43 |
| 3.3.1 | Proses Penyimpanan <i>Password</i> | 44 |
| 3.3.2 | Proses Rekonstruksi <i>Password</i> | 45 |
| 3.4 | Diagram | 46 |
| 3.4.1 | Diagram <i>Use Case</i> | 46 |
| 3.4.2 | Diagram Aktivitas | 47 |
| 3.4.3 | Diagram Kelas | 49 |
| 4 | PERANCANGAN | 51 |
| 4.1 | Diagram Kelas Rinci | 51 |
| 4.2 | Deskripsi Kelas dan Fungsi | 51 |
| 4.2.1 | Kelas <i>SHA512</i> | 51 |
| 4.2.2 | Kelas <i>Function</i> | 54 |
| 4.2.3 | Kelas <i>EquationSolver</i> | 54 |
| 4.2.4 | Kelas <i>SecretSharing</i> | 55 |
| 4.2.5 | Kelas <i>DESEncryption</i> | 56 |
| 4.2.6 | Kelas <i>DESDecryption</i> | 60 |
| 4.2.7 | Kelas <i>DataReader</i> | 62 |
| 4.2.8 | Kelas <i>DataWriter</i> | 63 |
| 4.3 | Perancangan Antarmuka | 64 |
| 5 | IMPLEMENTASI DAN PENGUJIAN | 67 |
| 5.1 | Implementasi Perangkat Lunak | 67 |
| 5.1.1 | Tampilan Antarmuka Perangkat Lunak | 67 |
| 5.2 | Pengujian Perangkat Lunak | 71 |
| 5.2.1 | Metode Pengujian | 71 |
| 5.2.2 | Hasil Pengujian Fungsional | 72 |
| 5.2.3 | Analisis Hasil Pengujian Fungsional | 75 |
| 5.2.4 | Analisis dan Hasil Pengujian Survei | 76 |
| 5.2.5 | Kesimpulan Pengujian | 80 |
| 6 | KESIMPULAN DAN SARAN | 83 |
| 6.1 | Kesimpulan | 83 |
| 6.2 | Saran | 84 |
| | DAFTAR REFERENSI | 85 |
| A | THE SOURCE CODE | 87 |

DAFTAR GAMBAR

| | | |
|------|--|----|
| 2.1 | Proses Enkripsi | 7 |
| 2.2 | Proses Permutasi | 8 |
| 2.3 | Proses Pembangunan Kunci Putaran | 9 |
| 2.4 | Putaran dalam DES | 11 |
| 2.5 | Jaringan Feistel | 12 |
| 2.6 | Struktur Putaran dalam SHA-512 | 19 |
| 2.7 | Masukan dan Keluaran dalam 1 Putaran SHA-512 | 20 |
| 2.8 | Fungsi Khusus dalam 1 Putaran SHA-512 | 20 |
| 2.9 | Proses Keseluruhan dari SHA-512 | 22 |
| 2.10 | <i>Username</i> dan <i>Password</i> | 23 |
| 2.11 | <i>Password hashing</i> | 24 |
| 2.12 | <i>Password salting</i> | 25 |
| 3.1 | Proses Penyimpanan <i>Password</i> | 44 |
| 3.2 | Proses Rekonstruksi <i>Password</i> | 45 |
| 3.3 | Diagram <i>use case</i> dari perangkat lunak | 46 |
| 3.4 | Diagram aktivitas untuk menyimpan <i>password</i> | 48 |
| 3.5 | Diagram aktivitas untuk mengembalikan <i>password</i> | 49 |
| 3.6 | Rancangan Diagram Kelas | 49 |
| 4.2 | Kelas SHA512 | 51 |
| 4.3 | Kelas <i>Function</i> | 54 |
| 4.4 | Kelas <i>EquationSolver</i> | 54 |
| 4.5 | Kelas <i>DESEncryption</i> | 55 |
| 4.6 | Kelas <i>DESEncryption</i> | 57 |
| 4.7 | Kelas <i>DESDecryption</i> | 60 |
| 4.8 | Kelas <i>DataReader</i> | 62 |
| 4.9 | Kelas <i>DataWriter</i> | 63 |
| 4.10 | Perancangan Tampilan Awal | 64 |
| 4.11 | Perancangan Tampilan Menyimpan <i>Password</i> | 64 |
| 4.12 | Perancangan Tampilan Mengembalikan <i>Password</i> | 65 |
| 4.13 | Perancangan Tampilan Mengembalikan <i>Password</i> | 65 |
| 4.1 | Diagram Kelas Rinci | 66 |
| 5.1 | Tampilan antarmuka awal | 67 |
| 5.2 | Tampilan antarmuka untuk menyimpan <i>password</i> | 68 |
| 5.3 | Tampilan notifikasi pada antarmuka jika <i>password</i> kurang | 68 |
| 5.4 | Tampilan antarmuka menambah <i>password</i> | 69 |
| 5.5 | Tampilan antarmuka untuk mengembalikan <i>password</i> | 70 |
| 5.6 | Tampilan antarmuka untuk mengembalikan <i>password</i> | 70 |
| 5.7 | Tampilan antarmuka untuk mengembalikan <i>password</i> | 71 |
| 5.8 | Langkah menyimpan <i>password</i> | 72 |
| 5.9 | Tampilan Menjawab Pertanyaan Keamanan | 73 |

| | |
|--|----|
| 5.10 Tampilan Menjawab Pertanyaan Keamanan Kasus Pertama | 74 |
| 5.11 Hasil Pengujian Fungsional Kasus Pertama | 74 |
| 5.12 Tampilan Menjawab Pertanyaan Keamanan Kasus Kedua | 75 |
| 5.13 Hasil Pengujian Fungsional Kasus Kedua | 75 |
| 5.14 Pengujian survei kasus 1 | 77 |
| 5.15 Pengujian survei kasus 2 | 78 |
| 5.16 Pengujian survei kasus 3 | 79 |
| 5.17 Pengujian survei kasus 4 | 80 |

DAFTAR TABEL

| | | |
|------|--|----|
| 2.1 | Matriks Permutasi Awal | 8 |
| 2.2 | Matriks Permutasi untuk <i>Parity Drop</i> | 9 |
| 2.3 | Matriks kompresi <i>P-box</i> | 10 |
| 2.4 | <i>P-box</i> | 12 |
| 2.5 | <i>S-box</i> 1 | 13 |
| 2.6 | <i>S-box</i> 2 | 13 |
| 2.7 | <i>S-box</i> 3 | 13 |
| 2.8 | <i>S-box</i> 4 | 14 |
| 2.9 | <i>S-box</i> 5 | 14 |
| 2.10 | <i>S-box</i> 6 | 14 |
| 2.11 | <i>S-box</i> 7 | 14 |
| 2.12 | <i>S-box</i> 8 | 14 |
| 2.13 | Matriks Permutasi <i>m</i> | 15 |
| 2.14 | Matriks Permutasi Akhir | 15 |
| 2.15 | Konstanta Awal | 17 |
| 3.1 | Nilai x untuk masing-masing $f(x)$ | 35 |
| 3.2 | Hasil Enkripsi setiap <i>Share</i> untuk <i>Password</i> Pertama | 37 |
| 3.3 | Hasil Dekripsi <i>Share</i> | 39 |
| 3.4 | Tabel Pasangan n dan k | 43 |
| 3.5 | Skenario Menyimpan <i>Password</i> | 47 |
| 3.6 | Skenario Menyimpan <i>Password</i> | 47 |

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Otentikasi adalah proses untuk menentukan keaslian identitas dari entitas saat akan mengakses sumber daya sebuah sistem. Entitas yang diotentikasi dapat berupa manusia atau pengguna sistem. Sistem yang hendak diakses dapat berupa media sosial, *email*, *electronic banking*, dan sebagainya. Proses otentikasi menentukan apakah seseorang berhak atau tidak untuk mengakses sumber daya sistem tersebut.

Salah satu dari metode otentikasi adalah dengan menggunakan *password*. *Password* adalah sekumpulan huruf, angka, dan simbol yang sifatnya rahasia. *Password* digunakan untuk mengakses sumber daya sebuah sistem. Saat pengguna sistem hendak mengakses sistem, pengguna akan memasukkan *password* untuk menunjukkan bahwa pengguna memiliki hak untuk mengakses sistem. Hal tersebut yang membuat *password* menjadi sebuah hal yang penting dan harus dijaga kerahasiaannya.

Namun, seorang pengguna tidak hanya membutuhkan sumber daya dari satu sistem saja. Pengguna membutuhkan akses ke banyak sistem. Akses pada sebuah sistem membutuhkan sebuah *password*. Semakin bertambahnya akses ke sistem yang berbeda-beda, semakin bertambah pula *password* yang harus dimiliki.

Hal ini dapat menimbulkan masalah jika ada *password* yang hilang atau dilupakan oleh pengguna. Pengguna akan kehilangan akses ke sistem tersebut. Beberapa sistem memang memiliki mekanisme untuk mengembalikan *password* yang hilang dengan menyediakan sebuah pertanyaan keamanan yang harus dijawab oleh pengguna. Namun, mekanisme ini bisa menyulitkan pengguna karena pengguna harus mengingat seluruh jawaban dari pertanyaan keamanan untuk setiap *password*.

Oleh sebab itu, dibutuhkan suatu mekanisme baru untuk bisa mengingat dan mengembalikan banyak *password* ini. Pada penelitian ini, akan dikembangkan mekanisme untuk mengembalikan *n password* dengan membuat *n* pertanyaan. Mekanisme ini akan menggunakan metode *secret sharing*.

Secret sharing adalah metode membagi sebuah pesan atau informasi menjadi beberapa bagian. Bagian-bagian tersebut disebut *share* dan setiap bagian dibagikan kepada beberapa partisipan. Untuk memperoleh kembali informasi, dibutuhkan masing-masing *share*. Terdapat beberapa metode *secret sharing* yang dapat digunakan untuk membagi informasi. Dalam penelitian ini, metode *secret sharing* yang digunakan adalah *secret sharing* Shamir. Metode *secret sharing* Shamir adalah metode *secret sharing* dengan membagi banyak informasi menjadi beberapa *share* dan untuk mengembalikan informasi hanya dibutuhkan beberapa

1 *share* saja.

2 Metode secret sharing Shamir ini akan diaplikasikan untuk membagi *password* menjadi
3 beberapa *share*. Setiap *password* yang akan dibagi diasosiasikan dengan satu pertanyaan
4 keamanan. Dengan menggunakan metode *secret sharing* Shamir, pengguna cukup menjawab
5 sebagian dari pertanyaan keamanan untuk mengembalikan n *password*.

6 Untuk menjaga kerahasiaan *password* dan jawaban dari setiap pertanyaan keamanan,
7 metode secret sharing ini akan dikombinasikan dengan enkripsi dan fungsi *hash*. Pada pe-
8 nelitian ini, teknik enkripsi yang akan digunakan adalah *Data Encryption Standard* dan
9 algoritma fungsi hash yang akan digunakan adalah *Secure Hashing Algorithm* 512.

10 Dalam penelitian ini, akan dibahas mengenai cara kerja metode *secret sharing* Shamir
11 untuk mengembalikan n *password* dengan membuat n pertanyaan keamanan. Selain itu,
12 pertanyaan keamanan yang dibuat akan dianalisis kualitasnya dan pengaruhnya terhadap
13 metode *secret sharing* Shamir dalam mengembalikan banyak *password*.

14 1.2 Rumusan Masalah

15 Berdasarkan latar belakang yang sudah dibuat, maka permasalahan yang akan dibahas da-
16 lam penelitian ini adalah:

- 17 • Bagaimana mengembalikan banyak *password* dengan metode *secret sharing* Shamir?
- 18 • Bagaimana cara membangun perangkat lunak pengingat *password* yang mengimple-
19 mentasikan metode *secret sharing* Shamir?
- 20 • Bagaimana menilai kualitas dari metode *secret sharing* Shamir lewat pertanyaan kea-
21 manan yang dibuat?

22 1.3 Tujuan

23 Berdasarkan rumusan masalah yang sudah ditetapkan, maka tujuan dari penelitian ini ada-
24 lah:

- 25 • Mempelajari bagaimana metode *secret sharing* Shamir dapat mengembalikan banyak
26 *password*.
- 27 • Membangun perangkat lunak pengingat *password* yang mengimplementasikan metode
28 *secret sharing* Shamir.
- 29 • Melakukan pengujian terhadap perangkat lunak pengingat *password* yang dibangun.

30 1.4 Batasan Masalah

31 Batasan masalah pada penelitian ini adalah setiap pertanyaan keamanan dijawab dengan
32 jawaban yang relevan.

1.5 Metodologi Penelitian

Metodologi dalam penelitian ini berupa:

- Melakukan studi literatur untuk mempelajari hal-hal yang diperlukan dalam penggunaan dan implementasi metode *secret sharing* Shamir.
- Membangun perangkat lunak yang mengimplementasikan metode *secret sharing* Shamir.
- Melakukan pengujian pada perangkat lunak yang sudah dibangun.

1.6 Sistematika Pembahasan

Sistematika pembahasan dalam penelitian ini berupa:

- Bab Pendahuluan
Bab 1 berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, metodologi penelitian, dan sistematika pembahasan.
- Bab Dasar Teori
Bab 2 berisi mengenai teori-teori dasar, antara lain kriptografi, algoritma enkripsi, algoritma fungsi *hash*, otentikasi, *secret sharing*, probabilitas, dan entropi.
- Bab Analisis
Bab 3 berisi analisis meliputi studi kasus penerapan metode *secret sharing* Shamir, analisis proses dalam bentuk *flow chart*, dan pemaparan diagram-diagram yang dibutuhkan dalam membangun perangkat lunak.
- Bab Perancangan
Bab 4 berisi tahapan penjelasan rancangan perangkat lunak meliputi diagram kelas rinci, deskripsi dan fungsi setiap kelas yang dibangun, dan rancangan tampilan perangkat lunak.
- Bab Implementasi dan Pengujian
Bab 5 berisi tahapan implementasi pada perangkat lunak meliputi tampilan antarmuka perangkat lunak, pengujian perangkat lunak, dan kesimpulan.
- Bab Kesimpulan dan Saran
Bab 6 berisi kesimpulan serta beberapa saran untuk pengembangan lebih lanjut dari penelitian yang dilakukan dan perangkat lunak yang dibangun.

BAB 2

DASAR TEORI

Pada bab ini akan dibahas dasar-dasar teori yang diperlukan dalam proses penulisan penelitian mengenai perlindungan *password* dengan entropi personal. Terdapat beberapa hal yang dibahas pada bab ini, yaitu mengenai kriptografi, *Data Encryption Standard*, *Secure Hash Algorithm 512*, otentikasi, eliminasi Gauss-Jordan, *secret sharing*, probabilitas, dan entropi.

2.1 Kriptografi

Pada bagian ini akan dijelaskan mengenai kriptografi dimulai dari sejarah kriptografi, dan pengertian kriptografi.

2.1.1 Sejarah Kriptografi

Kriptografi berasal dari bahasa Yunani, terdiri dari dua suku kata yaitu, *kripto* dan *graphia*, *kripto* berarti rahasia dan *graphia* berarti tulisan. Jadi, kriptografi berarti teknik atau metode untuk merahasiakan tulisan.

Kemunculan kriptografi ini diawali karena kebutuhan manusia untuk merahasiakan informasi berupa pesan atau tulisan. Pada zaman dahulu kala, kriptografi digunakan untuk merahasiakan tulisan-tulisan mengenai pesan rahasia, strategi perang, dan masih banyak lagi. Salah satu bentuk penggunaan kriptografi pada zaman dahulu kala adalah alat yang dinamakan *scytale*. *Scytale* digunakan oleh tentara Sparta di Yunani untuk mengirimkan pesan rahasia[1].

2.1.2 Pengertian Kriptografi

Zaman sekarang ini, kerahasiaan informasi menjadi hal yang penting. Informasi yang berharga perlu dirahasiakan sehingga tidak diketahui oleh orang yang tidak berhak. Kriptografi berperan dalam merahasiakan informasi berharga tersebut. Jadi, kriptografi adalah ilmu atau seni untuk menjaga kerahasiaan informasi.

Kriptografi memiliki 4 layanan utama[2]:

1. Kerahasiaan (*confidentiality*)

Layanan ini menjamin bahwa informasi yang dikirimkan tidak diketahui oleh pihak yang tidak berhak melihat atau membacanya.

2. Integritas (*integrity*)

Layanan ini menjamin keaslian dari informasi yang dikirimkan dan menjamin bahwa informasi yang dikirimkan tidak diubah tanpa seijin pengirim informasi.

3. Otentikasi (*authentication*)

Layanan ini menjamin keaslian identitas dari pengirim dan penerima informasi.

4. Non-repudiasi (*nonrepudiation*)

Layanan ini menjamin pengirim dan penerima informasi tidak dapat menyangkal aktivitas yang sudah dilakukan.

2.2 Kerahasiaan (*Confidentiality*)

Kerahasiaan adalah layanan yang menjamin bahwa informasi yang dikirimkan tidak dapat dibaca oleh orang atau pihak yang tidak berhak. Dalam kriptografi, informasi yang bisa dibaca dan dimengerti disebut *plaintext*. Informasi yang sudah dirahasiakan sehingga tidak bisa dibaca dan dimengerti disebut *ciphertext*. Untuk merahasiakan *plaintext*, maka *plaintext* harus diubah menjadi *ciphertext*. Kemudian, untuk bisa membaca kembali informasi yang sudah dirahasiakan, *ciphertext* harus diubah kembali menjadi *plaintext*.

Proses untuk mengubah *plaintext* menjadi *ciphertext* dinamakan enkripsi. Sebaliknya, proses untuk mengubah *ciphertext* menjadi *plaintext* dinamakan dekripsi. Proses enkripsi dan dekripsi ini menggunakan kunci. Kunci adalah sekumpulan huruf, angka, atau simbol. Kunci sifatnya rahasia dan hanya boleh diketahui oleh pemilik informasi.

Dalam proses enkripsi, *plaintext* dipetakan dengan fungsi enkripsi E menjadi *ciphertext* menggunakan kunci k , seperti pada persamaan 2.1.

$$E_k(\text{plaintext}) = \text{ciphertext} \quad (2.1)$$

Sementara itu, dalam proses dekripsi, *ciphertext* dipetakan dengan fungsi dekripsi D menjadi *plaintext* menggunakan kunci k seperti pada persamaan 2.2.

$$D_k(\text{ciphertext}) = \text{plaintext} \quad (2.2)$$

Proses enkripsi dan dekripsi ini menggunakan sekumpulan fungsi matematika untuk mengubah *plaintext* menjadi *ciphertext* dan sebaliknya. Sekumpulan fungsi matematika yang digunakan dalam proses enkripsi dan dekripsi dinamakan algoritma kriptografi. Menurut penggunaan kuncinya algoritma kriptografi dibagi menjadi 2 jenis, yaitu algoritma kriptografi kunci simetris dan algoritma kriptografi kunci asimetris.

Algoritma kunci simetris menggunakan kunci yang sama untuk proses enkripsi dan dekripsi. Pemilik informasi melakukan proses enkripsi dan dekripsi dengan kunci yang sama sehingga kunci harus dirahasiakan. Contoh dari algoritma kriptografi kunci simetris antara lain, *Data Encryption Standard* (DES), *Advanced Encryption Standard* (AES), *Twofish*, dan *Blowfish*.

Algoritma kunci asimetris menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsi. Pemilik informasi melakukan proses enkripsi menggunakan kunci yang dinamakan kunci publik dan melakukan proses dekripsi menggunakan kunci yang dinamakan kunci pribadi. Kunci publik sifatnya tidak rahasia dan kunci pribadi sifatnya rahasia. Contoh dari algoritma kriptografi kunci asimetris antara lain, Rivest-Shamir-Adleman (RSA), ElGamal, Diffie-Helman, *Digital Signature Algorithm*, dan *Elliptic Curve Digital Signature Algorithm* (ECDSA).

2.3 Data Encryption Standard (DES)

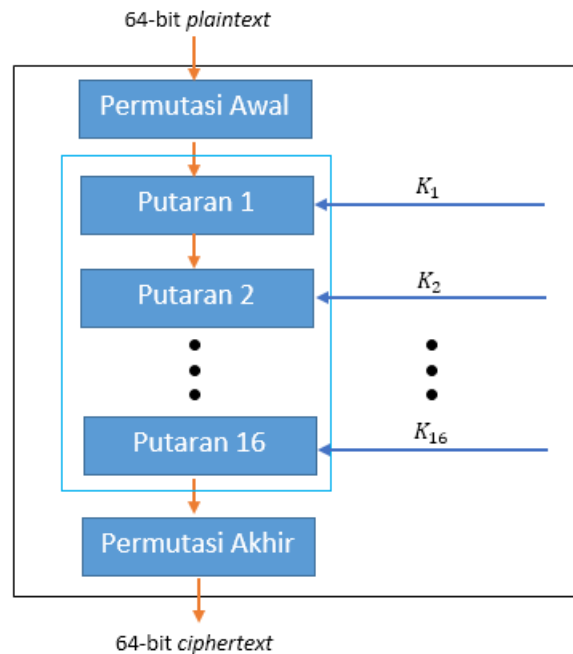
Pada bagian ini akan dijelaskan hal-hal mengenai *data encryption standard* dimulai dari sejarah *data encryption standard*, struktur *data encryption standard*, dan proses enkripsi *data encryption standard*.

2.3.1 Sejarah

Data encryption standard atau disingkat DES adalah algoritma kriptografi kunci simetris. DES pertama kali dipublikasikan oleh *National Institute of Standards and Technology* (NIST) pada tahun 1973. DES merupakan algoritma enkripsi pertama yang disetujui oleh pemerintah Amerika Serikat untuk digunakan secara luas. Pada bulan Maret 1975, NIST memublikasikan DES sebagai standar enkripsi untuk data pemerintahan atau *Federal Information Processing Standard* (FIPS).

2.3.2 Struktur DES

Masukkan dari DES berupa 64-bit *plaintext*. Keluaran dari DES berupa 64-bit *ciphertext*. DES menggunakan kunci yang sama pada proses enkripsi dan dekripsi. Panjang kunci dari DES adalah 64-bit. Proses enkripsi terdiri dari permutasi awal, putaran dan permutasi akhir. Gambar 2.1 menunjukkan proses enkripsi dari DES. Pada bagian selanjutnya akan dijelaskan mengenai setiap bagian dari proses enkripsi.



Gambar 2.1: Proses Enkripsi

2.3.3 Permutasi Awal

Permutasi awal dalam DES menggunakan matriks permutasi mp . Masukan dari matriks permutasi mp adalah *plaintext*. Tabel 2.1 menunjukkan matriks permutasi mp .

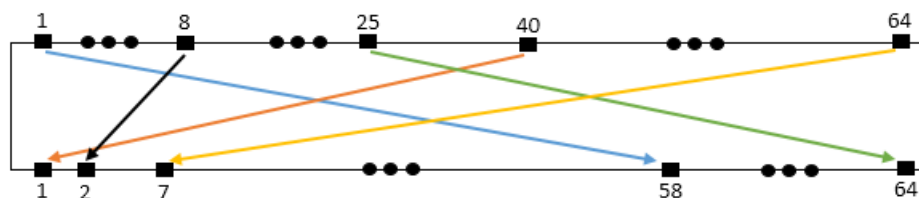
Tabel 2.1: Matriks Permutasi Awal

| | | | | | | | |
|----|----|----|----|----|----|----|---|
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

- 1 Cara kerja dari proses permutasi adalah sebagai berikut. Angka yang ditunjukkan pada
2 posisi ke- i matriks mp merupakan posisi bit dari masukan, sedangkan i menunjukkan posisi
3 bit dari keluaran. Proses permutasi ditunjukkan oleh persamaan 2.3.

$$keluaran_i = masukan_{p_i} \quad (2.3)$$

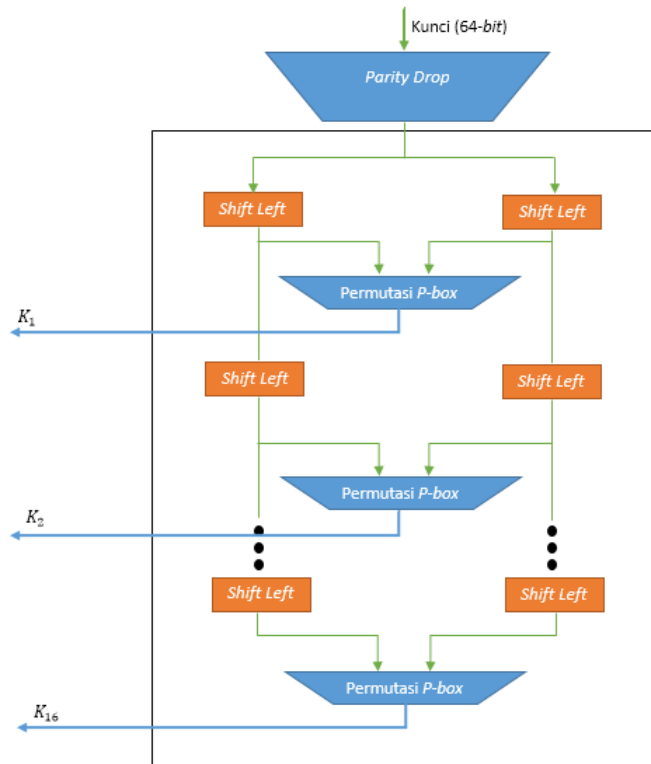
- 4 Sebagai contoh, posisi ke-1 dari matriks mp menunjukkan angka 58. Maka, bit ke-58
5 dari masukan akan menjadi bit ke-1 dari keluaran. Gambar 2.2 menunjukkan ilustrasi dari
6 proses permutasi yang sudah dijelaskan.



Gambar 2.2: Proses Permutasi

7 2.3.4 Pembangunan Kunci Putaran

- 8 DES menggunakan kunci dengan panjang 64- bit . Kunci ini perlu diubah menjadi kunci un-
9 tuk setiap putaran DES dengan panjang masing-masing 48- bit . Proses pembangunan kunci
10 putaran terdiri dari *parity drop*, *shift left*, dan permutasi P -*box*. Gambar 2.3 menunjukk-
11 an keseluruhan proses dari pembangunan kunci putaran. Pada bagian ini akan dijelaskan
12 masing-masing proses dari pembangunan kunci putaran.



Gambar 2.3: Proses Pembangunan Kunci Putaran

1 *Parity Drop*

2 Pada proses ini, *parity bit* akan dihilangkan dari kunci masukan. *Bit* yang dihilangkan
 3 adalah *bit* posisi kelipatan 8, yaitu posisi ke-8, posisi ke-16, posisi ke-24, dan seterusnya
 4 sampai posisi ke-64. Proses penghilangan *parity bit* ini menggunakan matriks permutasi p
 5 seperti ditunjukkan pada Tabel 2.2. Cara kerja proses permutasi sama dengan cara kerja
 6 proses permutasi pada tahap permutasi awal (Subbab 2.3.3).

Tabel 2.2: Matriks Permutasi untuk *Parity Drop*

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 60 | 52 | 44 | 36 | 63 | 55 | 47 | 39 |
| 31 | 23 | 15 | 7 | 62 | 54 | 46 | 38 |
| 30 | 22 | 14 | 6 | 61 | 53 | 45 | 37 |
| 29 | 21 | 13 | 5 | 28 | 20 | 12 | 4 |

7 Hasil akhir dari proses ini kunci dengan panjang 56-bit.

8 *Shift Left*

9 Pada proses ini, kunci hasil proses *parity drop* dibagi menjadi 2 bagian dengan panjang
 10 masing-masing 28-bit, yaitu bagian kiri (L) dan bagian kanan (R). L dan R akan digeser ke

1 arah kiri secara sirkular sebanyak 1 atau 2 *bit* tergantung dari urutan putaran. Ketentuan
2 banyak *bit* yang digeser adalah sebagai berikut.

3 • Untuk putaran ke-1, 2, 9, dan 16 maka L dan R akan digeser ke arah kiri secara
4 sirkular sebanyak 1 *bit*.

5 • Untuk putaran ke-3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, dan 15, L dan R akan digeser ke
6 arah kiri secara sirkular sebanyak 2 *bit*.

7 Sebagai contoh, diasumsikan L dan R pada persamaan 2.4 dan 2.5.

$$L = 1001\ 1010\ 1000\ 0110\ 0110\ 1111\ 1101 \quad (2.4)$$

$$R = 0001\ 0100\ 0111\ 1110\ 1010\ 0101\ 1011 \quad (2.5)$$

8 Untuk putaran ke-1, 2, 9, dan 16 maka hasil dari L dan R akan seperti yang ditunjukkan
9 pada persamaan 2.6 dan 2.7.

$$L = 0011\ 0101\ 0000\ 1100\ 1101\ 1111\ 1011 \quad (2.6)$$

$$R = 0010\ 1000\ 1111\ 1101\ 0100\ 1011\ 0110 \quad (2.7)$$

10 Sementara itu, jika untuk putaran ke-3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, dan 15 akan
11 seperti yang ditunjukkan pada persamaan 2.8 dan 2.9.

$$L = 0110\ 1010\ 0001\ 1001\ 1011\ 1111\ 0110 \quad (2.8)$$

$$R = 0101\ 0001\ 1111\ 1010\ 1001\ 0110\ 1100 \quad (2.9)$$

12 Kemudian, L dan R akan disatukan kembali sehingga panjangnya menjadi 56-*bit*.

13 Permutasi P -box

14 Tahap ini adalah proses permutasi untuk mengubah kunci dari proses *Shift Left* dengan pan-
15 jang 56-*bit* menjadi kunci putaran dengan panjang 48-*bit*. Tabel 2.3 menunjukkan matriks
16 permutasi P -box yang digunakan untuk proses ini.

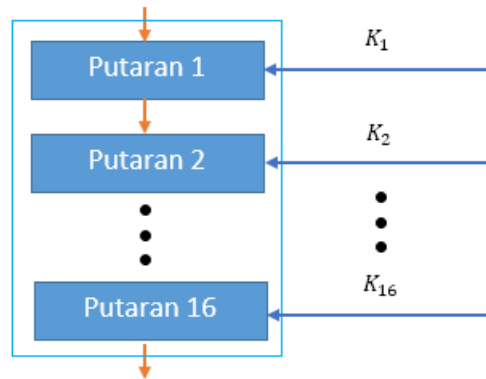
Tabel 2.3: Matriks kompresi P -box

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 14 | 17 | 11 | 24 | 1 | 5 | 3 | 28 |
| 15 | 6 | 21 | 10 | 23 | 19 | 12 | 4 |
| 26 | 8 | 16 | 7 | 27 | 20 | 13 | 2 |
| 41 | 52 | 31 | 37 | 47 | 55 | 30 | 40 |
| 51 | 45 | 33 | 48 | 44 | 49 | 39 | 56 |
| 32 | 29 | 36 | 50 | 42 | 46 | 53 | 34 |

17 Hasil keluaran dari proses ini adalah kunci putaran dengan panjang 48-*bit* dan siap
18 dipakai untuk masing-masing putaran.

2.3.5 Putaran

DES terdiri dari 16 putaran. Setiap putaran adalah jaringan Feistel yang akan dijelaskan pada bagian selanjutnya. Gambar 2.4 menunjukkan ilustrasi dari 16 putaran dari DES.



Gambar 2.4: Putaran dalam DES

Jaringan Feistel

Pada bagian ini akan dijelaskan mengenai sejarah singkat dari jaringan Feistel dan pembahasan jaringan Feistel.

Sejarah Singkat

Jaringan Feistel diciptakan oleh ilmuwan asal Jerman bernama Horst Feistel. Horst Feistel mempublikasikan jaringan ini pada tahun 1973. Jaringan Feistel banyak digunakan dalam berbagai skema enkripsi khususnya digunakan dalam DES.

Pembahasan

Masukan dari jaringan Feistel adalah *plaintext* dengan panjang 64-bit dan keluaran dari jaringan Feistel adalah *ciphertext* dengan panjang 64-bit. Jaringan Feistel menggunakan kunci K dan fungsi enkripsi f dalam pemrosesan *plaintext*. Selanjutnya akan dijelaskan langkah-langkah pemrosesan *plaintext* pada jaringan Feistel.

1. *Plaintext* dibagi menjadi 2 bagian sama panjang, yaitu bagian kiri (L_{i-1}) dan bagian kanan (R_{i-1}). Huruf i menunjukkan urutan dari putaran. Panjang masing-masing bagian adalah 32-bit.
2. Bagian kanan (R_{i-1}) pada *plaintext* akan menjadi bagian kiri (L_i) dari *ciphertext*. Persamaan 2.10 menunjukkan langkah yang sudah dijelaskan.

$$L_i = R_{i-1} \quad (2.10)$$

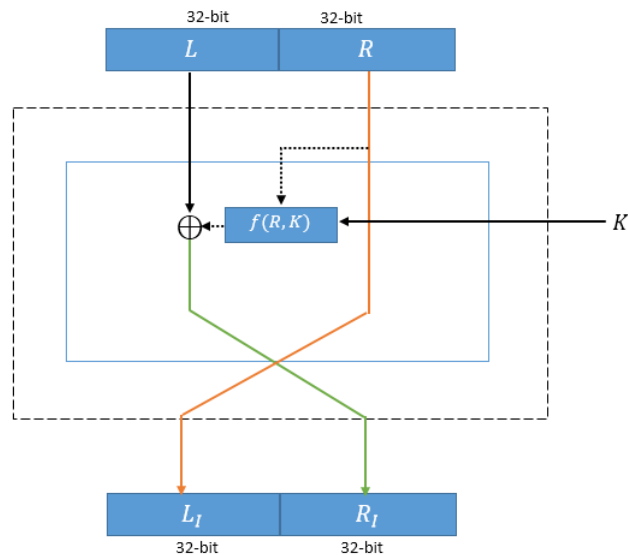
3. Untuk memperoleh bagian kanan dari *ciphertext* (R_i), bagian kanan dari *plaintext* (R_{i-1}) dan kunci putaran K_i dipetakan dengan fungsi f . Kemudian, hasil pemetaan dengan fungsi f akan di *exclusive-or* (XOR) dengan bagian kiri dari *plaintext* (L_{i-1}).

Persamaan 2.11 menunjukkan langkah yang sudah dijelaskan.

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \quad (2.11)$$

4. Hasil akhirnya berupa *ciphertext* dengan 2 bagian sama panjang, yaitu bagian kiri (L_i) dan bagian kanan (R_i).

Gambar 2.5 menunjukkan ilustrasi dari langkah-langkah yang sudah dijelaskan.



Gambar 2.5: Jaringan Feistel

4 Fungsi DES

Fungsi DES adalah fungsi f yang digunakan dalam jaringan Feistel pada Gambar 2.5. Fungsi DES terdiri dari 4 bagian, yaitu ekspansi *P-box*, operasi XOR, substitusi *S-box*, dan permutasi. Pada bagian selanjutnya akan dijelaskan masing-masing bagian dari fungsi DES.

8 Ekspansi *P-box*

Pada bagian ini, masukan berupa blok bagian kanan dari *plaintext* (R) dengan panjang 32-bit. Ekspansi *P-box* menggunakan matriks permutasi p yang ditunjukkan pada tabel 2.4.

Tabel 2.4: *P-box*

| | | | | | |
|----|----|----|----|----|----|
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

Hasil keluaran dari ekspansi *P-box* adalah blok dengan panjang 48-bit.

1 Operasi XOR

- 2 Setelah ekspansi $P\text{-box}$, dilakukan operasi XOR antara R dengan kunci putaran ke- i , K_i .
- 3 Kunci putaran hanya digunakan pada bagian ini saja.

4 Substitusi $S\text{-box}$

- 5 Pada bagian ini, akan dilakukan substitusi pada R dengan menggunakan $S\text{-box}$. Masukan
- 6 dari $S\text{-box}$ adalah R dengan panjang 48-bit dan keluarannya adalah R dengan panjang 32-bit.
- 7 R akan dibagi menjadi 8 blok dengan panjang masing-masing 6-bit. Setiap blok memiliki $S\text{-box}$
- 8 masing-masing. Blok pertama menggunakan $S\text{-box}$ pertama, blok kedua menggunakan
- 9 $S\text{-box}$ kedua, dan seterusnya. Berikut masing-masing dari $S\text{-box}$ ditunjukkan pada Tabel 2.5
- 10 sampai Tabel 2.12.

Tabel 2.5: $S\text{-box}$ 1

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|----|----|----|---|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 10 | 3 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

Tabel 2.6: $S\text{-box}$ 2

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|----|----|----|----|----|----|----|----|----|---|----|----|----|----|----|----|
| 0 | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 1 | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 12 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 2 | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 3 | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |

Tabel 2.7: $S\text{-box}$ 3

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|----|----|----|----|---|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 1 | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 2 | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 3 | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |

Tabel 2.8: *S-box 4*

| | | | | | | | | | | | | | | | | |
|---|----|----|----|---|----|----|----|----|----|---|----|----|----|----|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 1 | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 2 | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |

Tabel 2.9: *S-box 5*

| | | | | | | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 2 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 3 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

Tabel 2.10: *S-box 6*

| | | | | | | | | | | | | | | | | |
|---|----|----|----|----|---|----|----|----|----|----|----|----|----|----|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 2 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 3 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |

Tabel 2.11: *S-box 7*

| | | | | | | | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|----|----|----|----|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 1 | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 2 | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 3 | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |

Tabel 2.12: *S-box 8*

| | | | | | | | | | | | | | | | | |
|---|----|----|----|---|----|----|----|----|----|----|----|----|----|----|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 2 | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 3 | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

- 1 Proses substitusi terjadi sebagai berikut. Kombinasi *bit* ke-1 dan *bit* ke-6 pada blok
- 2 akan menunjukkan posisi baris pada *S-box*. Kemudian, kombinasi dari *bit* ke-2 sampai ke-5

menunjukkan posisi kolom pada *S-box*. Setelah itu, angka yang ditunjuk oleh baris dan kolom pada *S-box* ini akan menjadi blok keluaran.

Sebagai contoh, diasumsikan masukan dari *S-box* pertama adalah 110011. Maka, kombinasi *bit*nya adalah 11 untuk baris dan 1001 untuk kolom. Jadi, baris yang dipilih adalah baris ke-3 dan kolom yang dipilih adalah kolom ke-9. Angka yang ditunjuk oleh *S-box* pertama pada baris ke-3 dan kolom ke-9 adalah 11. Maka, blok keluaran untuk *S-box* pertama adalah 1011. Lalu, setelah seluruh blok masukan diproses dengan *S-box* masing-masing, seluruh blok keluaran digabungkan menjadi blok dengan panjang 32-*bit*.

9 Permutasi

Bagian ini adalah bagian terakhir dari fungsi DES. Masukan dari bagian ini adalah blok keluaran dari proses substitusi *S-box*, yaitu blok dengan panjang 32-*bit*. Proses permutasi dilakukan dengan menggunakan matriks m yang ditunjukkan oleh Tabel 2.13. Hasil keluaran dari bagian ini adalah blok dengan panjang 32-*bit*.

Tabel 2.13: Matriks Permutasi m

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 |
| 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 |
| 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

Setelah proses permutasi ini, hasil dari proses permutasi akan di exclusive-or (XOR) dengan L_{i-1} seperti yang sudah dijelaskan pada bagian Jaringan Feistel. Hasil XOR adalah bagian kanan dari ciphertext (R_i). Setelah itu, L_i dan R_i akan digabungkan kemudian dijadikan sebagai masukan untuk putaran selanjutnya.

18 2.3.6 Permutasi Akhir

Setelah dilakukan 16 putaran, tahap terakhir dari enkripsi DES adalah permutasi akhir. Proses permutasi akhir menggunakan matriks yang ditunjukkan pada Tabel 2.14. Hasil dari proses permutasi akhir adalah 64-bit *ciphertext*.

Tabel 2.14: Matriks Permutasi Akhir

| | | | | | | | |
|----|---|----|----|----|----|----|----|
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 |
| 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

22 2.4 Fungsi *Hash*

Fungsi *hash* adalah fungsi yang memiliki masukan berupa *string* dengan panjang sembarang dan menghasilkan keluaran berupa *string* dengan panjang yang tetap. Masukan dari fungsi

1 *hash* dinamakan *message*. Hasil keluaran dari fungsi *hash* dinamakan *digest*. *Message* m
 2 akan dipetakan dengan fungsi *hash* H menghasilkan *digest* h . Persamaan 2.12 menunjukkan
 3 pemetaan m dengan H yang menghasilkan h .

$$h = H(m) \quad (2.12)$$

4 Fungsi *hash* harus memiliki 3 kriteria sebagai berikut[2].

5 1. Preimage Resistance

6 Untuk setiap $h = H(m)$ yang dihasilkan, tidak mungkin dikembalikan m sedemikian
 7 rupa sehingga $H(m) = h$. Dalam proses pembuatan *digest*, fungsi *hash* menghilangkan
 8 beberapa bagian dari m (*lossy*). Maka dari itu, *digest* tidak bisa dikembalikan menjadi
 9 *message*. Itulah sebabnya fungsi *hash* disebut fungsi satu arah.

10 2. Second Preimage Resistance

11 Untuk setiap m yang diberikan, tidak mungkin mencari $m' \neq m$ sedemikian rupa
 12 sehingga $H(m') = H(m)$.

13 3. Collision Resistance

14 Tidak mungkin mencari pasangan m dan m' sedemikian rupa sehingga $h = H(m)$
 15 sama dengan $h' = H(m')$. Untuk 2 *message* yang berbeda tidak mungkin menghasilkan
 16 *digest* yang sama.

17 Contoh fungsi *hash* antara lain MD-2, MD-4, MD-5, SHA-0, SHA-1, SHA-256, dan
 18 SHA-512.

19 2.5 Secure Hashing Algorithm 512 (SHA-512)

20 *Secure hashing algorithm* 512 atau SHA-512 adalah algoritma fungsi *hash* yang menghasilkan
 21 *digest* dengan panjang 512-bit. Proses dari SHA-512 terdiri dari *message padding*, inisialisasi
 22 konstanta awal, ekspansi blok *message*, fungsi kompresi, dan putaran. Bagian selanjutnya
 23 akan menjelaskan masing-masing proses dari SHA-512.

24 2.5.1 Message Padding

25 Sebelum *digest* dibuat, *message* akan dipadding terlebih dahulu. Pertama-tama, blok *mes-*
 26 *sage* M akan dipadding dengan blok L . Blok L berisi informasi mengenai panjang dari M .
 27 Panjang dari blok L adalah 128-bit. Kemudian, gabungan dari blok M dan L akan dipadding
 28 lagi dengan blok *padding* P sampai panjang dari gabungan blok M , L , dan P mencapai ke-
 29 lipatan 1024-bit. Panjang dari blok *padding* P bervariasi. Persamaan 2.13 menunjukkan
 30 rumus untuk menghitung panjang dari blok *padding* P .

$$(M + P + 128) = 0 \text{ mod } 1024 \quad \Rightarrow \quad P = (-M - 128) \text{ mod } 1024 \quad (2.13)$$

31 Isi dari blok *padding* P adalah angka 1 diikuti dengan angka 0. Sebagai contoh, jika
 32 panjang dari *message* (M) adalah 2590 bit, maka panjang dari blok *padding* P ditunjukkan
 33 pada persamaan 2.14.

$$\begin{aligned}
P &= (-2590 - 128) \bmod 1024 \\
&= -2718 \bmod 1024 \\
&= 354
\end{aligned} \tag{2.14}$$

Maka, dari persamaan 2.14, panjang dari blok P adalah 354 *bit*. Isi dari blok P adalah 1 *bit* angka 1 diikuti dengan 353 *bit* angka 0.

2.5.2 Inisialisasi Konstanta Awal

Setelah proses *message padding*, proses selanjutnya adalah inisialisasi konstanta awal. Ada 8 konstanta awal yang akan dibentuk. Delapan konstanta awal ini akan diberi nama $A_0, B_0, C_0, D_0, E_0, F_0, G_0$, dan H_0 . Panjang masing-masing konstanta awal ini adalah 64-*bit*. Setiap nilai konstanta awal diperoleh dari nilai di belakang koma dari akar kuadrat bilangan prima. Kemudian, nilai di belakang koma ini akan diubah menjadi heksadesimal. Bilangan prima yang digunakan untuk masing-masing konstanta awal adalah bilangan prima awal secara berurutan, yaitu 2, 3, 5, 7, 11, 13, 17, dan 19.

Sebagai contoh, misalkan akan dicari nilai untuk A_0 . A_0 merupakan konstanta awal pertama maka bilangan prima yang digunakan adalah bilangan prima urutan pertama, yaitu 2. Setelah itu, akan dihitung akar kuadrat dari 2. Kemudian, angka di belakang koma dari akar kuadrat 2 akan diubah menjadi heksadesimal. Nilai heksadesimal inilah yang menjadi nilai dari A_0 . Persamaan 2.15 menunjukkan langkah yang sudah dijelaskan.

$$\begin{aligned}
A_0 &= \sqrt{2} \\
&= 1.4142135623730950 \\
&= (1.6A09E667F3BCC908)_{16} \\
&= 6A09E667F3BCC908
\end{aligned} \tag{2.15}$$

Tabel 2.15 menunjukkan nilai masing-masing konstanta.

Tabel 2.15: Konstanta Awal

| <i>Konstanta</i> | <i>Nilai</i> | <i>Konstanta</i> | <i>Nilai</i> |
|------------------|------------------|------------------|------------------|
| A_0 | 6A09E667F3BCC908 | E_0 | 510E527FADE682D1 |
| B_0 | BB67AE8584CAA73B | F_0 | 9B05688C2B3E6C1F |
| C_0 | 3C6EF372FE94F828 | G_0 | 1F83D9ABFB41BD6B |
| D_0 | A54FF53A5F1D36F1 | H_0 | 5BE0CD19137E2179 |

2.5.3 Ekspansi Blok Message

Setelah inisialisasi konstanta awal, proses berikutnya adalah ekspansi blok *message*. Sesudah blok *message* dipadding, blok *message* akan dibagi menjadi beberapa blok yang panjangnya masing-masing 1024-*bit*. Kemudian, setelah dibagi menjadi beberapa blok 1024-*bit*, masing-masing dari 1024-*bit* akan dibagi lagi menjadi blok-blok dengan panjang 64-*bit*. Blok dengan panjang 64-*bit* ini dinamakan *word*.

Satu blok 1024-bit terdiri dari 16 word. Proses ekspansi blok *message* akan mengekspansi dari 16 word dari 1 blok 1024-bit menjadi 80 word. Masing-masing word ini akan diberi nama W_0 sampai W_{79} . Untuk W_0 sampai W_{15} berisi dari 16 word pertama dari blok 1024-bit. Sementara itu, W_{16} sampai W_{79} diperoleh dengan rumus dasar yang ditunjukkan oleh persamaan 2.16.

$$W_i = W_{i-16} \oplus RotShift_{1-8-7}(W_{i-15}) \oplus W_{i-7} \oplus RotShift_{19-61-6}(W_{i-2}) \quad (2.16)$$

Sebagai contoh untuk memperoleh nilai dari W_{60} , maka rumus dasarnya adalah seperti yang ditunjukkan pada persamaan 2.17.

$$W_{60} = W_{44} \oplus RotShift_{1-8-7}(W_{45}) \oplus W_{53} \oplus RotShift_{19-61-6}(W_{58}) \quad (2.17)$$

$RotShift$ pada persamaan 2.16 dan 2.17 adalah hasil *exclusive-or* (XOR) dari operasi rotasi ke kanan dan *shift left*. Rumus untuk rotasi ke kanan dan *shift left* ditunjukkan pada persamaan 2.18.

$$RotShift_{l-m-n}(x) = RotR_l(x) \oplus RotR_m(x) \oplus ShL_n(x) \quad (2.18)$$

$RotR_i(x)$ pada persamaan 2.18 adalah rotasi ke kanan x sebanyak i bit. Sebagai contoh, diasumsikan $i = 2$ dan $x = 1001$, maka hasil dari $RotR_2(1001)$ ditunjukkan pada persamaan 2.19.

$$\begin{aligned} i = 1 & \Rightarrow x = 1100 \\ i = 2 & \Rightarrow x = 0110 \\ RotR_2(1001) &= 0110 \end{aligned} \quad (2.19)$$

Sementara itu, $ShL_i(x)$ pada persamaan 2.18 adalah operasi *shift left* x sebanyak i bit dipadding dengan angka 0. Sebagai contoh, diasumsikan $i = 2$ dan $x = 1011$, maka hasil dari $ShL_2(1011)$ ditunjukkan pada persamaan 2.20.

$$\begin{aligned} i = 1 & \Rightarrow x = 0110 \\ i = 2 & \Rightarrow x = 1100 \\ ShL_2(1011) &= 1100 \end{aligned} \quad (2.20)$$

Setelah ekspansi blok *message* menjadi 80 word untuk setiap blok *message*, proses selanjutnya adalah putaran dari SHA-512. Proses putaran SHA-512 akan dijelaskan pada bagian selanjutnya.

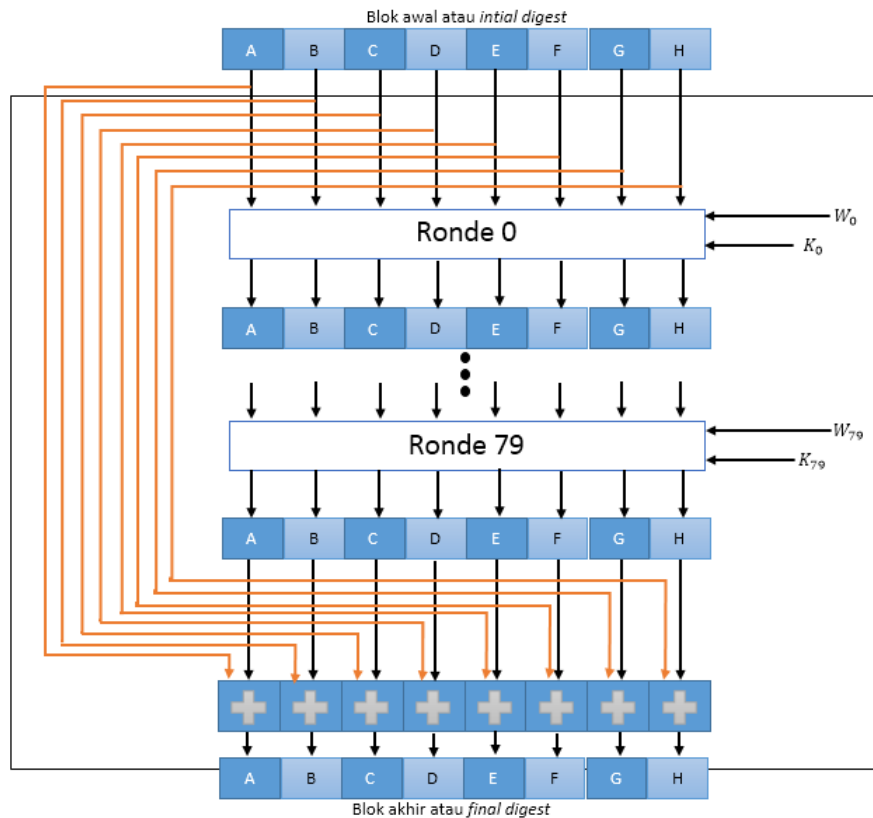
2.5.4 Fungsi Kompresi dan Putaran

Fungsi kompresi adalah proses yang mengkompresi blok 512-bit dan blok *message* yang berukuran 1024-bit menjadi blok keluaran dengan panjang 512-bit. Fungsi kompresi ini

1 terdiri dari 80 putaran SHA-512.

2 Struktur Putaran

3 Masukan dari putaran SHA-512 adalah blok dengan panjang 512-bit terdiri dari 8 word (A ,
4 B , C , D , E , F , G , dan H). Untuk putaran pertama, blok 512-bit diperoleh dari konstanta
5 awal (A_0 sampai H_0) sedangkan untuk putaran kedua dan selanjutnya blok 512-bit diperoleh
6 dari hasil dari putaran sebelumnya. Gambar 2.6 menunjukkan ilustrasi proses yang sudah
7 dijelaskan.

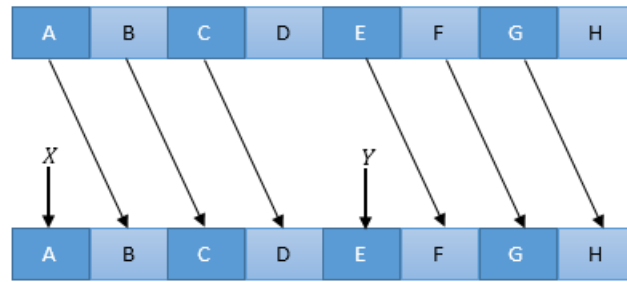


Gambar 2.6: Struktur Putaran dalam SHA-512

8 Dalam 1 putaran SHA-512, *word* keluaran diperoleh dari salinan *word masukan*, berikut
9 menunjukkan masukan dan keluaran dari masing-masing *word*.

- 10 • *Word* keluaran B diperoleh dari *word* masukan A
- 11 • *Word* keluaran C diperoleh dari *word* masukan B
- 12 • *Word* keluaran D diperoleh dari *word* masukan C
- 13 • *Word* keluaran F diperoleh dari *word* masukan E
- 14 • *Word* keluaran G diperoleh dari *word* masukan F
- 15 • *Word* keluaran H diperoleh dari *word* masukan G

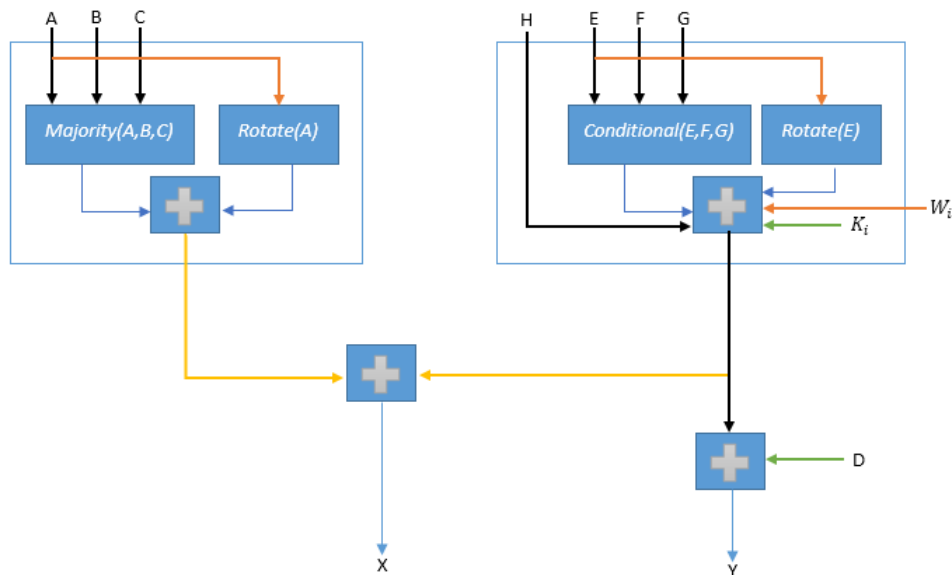
16 Gambar 2.7 menunjukkan ilustrasi dari masukan dan keluaran dalam 1 putaran SHA-512
17 untuk setiap *word*.



Gambar 2.7: Masukan dan Keluaran dalam 1 Putaran SHA-512

- 1 Untuk nilai *word* keluaran *A* dan *E* diperoleh dari *word* *X* dan *Y*. *Word* *X* dan *Y* ini
- 2 diperoleh dari sebuah fungsi khusus. Gambar 2.8 menunjukkan struktur dari fungsi khusus.
- 3 Berikut akan dijelaskan struktur dari fungsi khusus.

4 Struktur Fungsi Khusus



Gambar 2.8: Fungsi Khusus dalam 1 Putaran SHA-512

- 5 *Word* *Y* pada gambar 2.8 diperoleh dari proses persamaan 2.21.

$$Y = D + (Conditional(E, F, G) + Rotate(E) + W_i + K_i + H) \quad (2.21)$$

- 6 Nilai W_i diperoleh dari proses Ekspansi Blok *Message* (Subbab 2.5.3), dimana i menun-
- 7 jukkan urutan dari putaran. Nilai K_i pada persamaan 2.21 diperoleh dari nilai belakang
- 8 koma akar kubik bilangan prima ke- $(i + 1)$. Kemudian, nilai belakang koma ini akan dikon-
- 9 versi menjadi heksadesimal.

- 10 Bilangan prima yang digunakan untuk menghitung nilai K_i dimulai dari 2 untuk K_0 ,
- 11 3 untuk K_1 , dan seterusnya secara berurutan sampai 409 untuk K_{79} . Persamaan 2.22
- 12 menunjukkan cara untuk menghitung salah satu dari nilai K_i .

$$\begin{aligned}
K_{79} &= \sqrt[3]{409} \\
&= 7.4229141204362155 \\
&= (7.6C44198C4A475817)_{16} \\
&= 6C44198C4A475817
\end{aligned} \tag{2.22}$$

1 Sementara itu, untuk operasi *Conditional* pada persamaan 2.21 adalah operasi *AND*,
2 *OR* dan *XOR* dari *bit-bit* setiap *word*. Rumus dari *Conditional* ditunjukkan oleh persamaan
3 2.23.

$$Conditional(x, y, z) = (x \text{ AND } y) \oplus (NOT \ x \text{ AND } z) \tag{2.23}$$

4 Operasi *Rotate* pada persamaan 2.21 adalah hasil *exclusive-or* (*XOR*) dari $RotR_i(x)$.
5 $RotR_i(x)$ merupakan operasi rotasi ke kanan x sebanyak i -bit yang sudah dijelaskan pa-
6 da proses Ekspansi Blok Message (Subbab 2.5.3). Rumus dari *Rotate* ditunjukkan pada
7 persamaan 2.24.

$$Rotate(x) = RotR_{28}(x) \oplus RotR_{34}(x) \oplus RotR_{39}(x) \tag{2.24}$$

8 Hasil pertambahan *bit-bit* operasi *Conditional*, operasi *Rotate*, W_i , K_i , dan *word* H akan
9 ditambahkan dengan *word* D untuk menghasilkan *word* Y .

10 Kemudian, *word* X pada Gambar 2.8 diperoleh dari persamaan 2.25.

$$X = (Majority(A, B, C) + Rotate(A)) + (Conditional(E, F, G) + Rotate(E) + W_i + K_i + H) \tag{2.25}$$

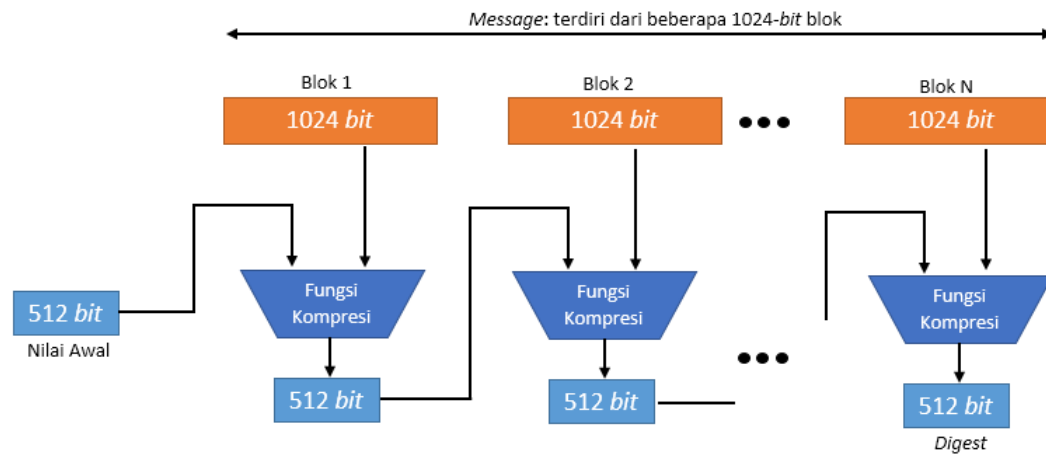
11 Untuk operasi *Conditional* dan *Rotate* sudah dijelaskan pada persamaan 2.23 dan 2.23.
12 Sementara itu, untuk operasi *Majority* pada persamaan 2.25 adalah operasi *AND*, *OR* dan
13 *XOR* dari *bit-bit* setiap *word*. Operasi *Majority* ditunjukkan pada persamaan 2.26.

$$Majority(x, y, z) = (x \text{ AND } y) \oplus (y \text{ AND } z) \oplus (z \text{ AND } x) \tag{2.26}$$

14 Hasil akhir dari fungsi khusus adalah *word* X dan *word* Y . *Word* X akan menjadi *word*
15 keluaran A dan *word* Y akan menjadi *word* keluaran E . Ilustrasi dari hasil keluaran ini
16 ditunjukkan oleh Gambar 2.7.

17 Proses setelah 80 putaran dilakukan adalah operasi pertambahan masing-masing *bit* da-
18 ri blok 512-bit hasil keluaran putaran ke-80 dengan masing-masing *bit* dari blok 512-bit
19 masukan untuk putaran ke-1. Ilustrasi proses ini ditunjukkan oleh Gambar ??.

20 Kemudian, hasil akhir dari proses tersebut berupa blok dengan panjang 512-bit terdiri
21 dari 8 *word*. Blok 512-bit ini akan menjadi hasil akhir (*digest*) atau menjadi masukan
22 untuk fungsi kompresi yang digunakan oleh blok *message* ke-2 dan seterusnya. Gambar 2.9
23 menunjukkan proses yang sudah dijelaskan.



Gambar 2.9: Proses Keseluruhan dari SHA-512

2.6 Otentikasi

Otentikasi adalah proses untuk menentukan keaslian identitas dari sebuah entitas saat akan mengakses sumber daya sebuah sistem. Berdasarkan entitas yang diotentikasi [2], otentikasi dibagi menjadi 2 jenis, yaitu:

1. Otentikasi pesan

Otentikasi pesan adalah proses otentikasi untuk memastikan bahwa pesan berasal dari sumber data yang bisa dipercaya. Otentikasi pesan juga memastikan bahwa pesan tidak diubah saat pengiriman pesan sedang berlangsung. Beberapa teknik otentikasi pesan adalah *Modification Detection Code* dan *Message Authentication Code*.

2. Otentikasi entitas

Otentikasi entitas adalah proses otentikasi untuk memastikan kebenaran identitas seseorang. Entitas yang diotentikasi bisa berupa orang atau pengguna (*user*). Beberapa teknik otentikasi entitas adalah *password*, *zero-knowledge*, *challenge-response*, dan biometrik.

Sementara itu, berdasarkan bentuknya [2], otentikasi dibagi menjadi 3 jenis, yaitu:

1. Sesuatu yang diketahui (*something known*)

Sesuatu yang diketahui oleh pengirim pesan dan kebenarannya bisa dipastikan oleh penerima pesan. Contohnya antara lain adalah *password*, nomor PIN, *passphrase*, dan sebagainya.

2. Sesuatu yang dimiliki (*something possessed*)

Sesuatu yang dimiliki adalah sesuatu yang menunjukkan identitas dari pengirim pesan. Contohnya adalah paspor, KTP, kartu kredit, SIM, dan sebagainya.

3. Sesuatu yang melekat (*something inherent*)

Sesuatu yang melekat adalah sesuatu yang menempel atau sebagai bagian dari pengirim pesan. Contohnya adalah sidik jari, suara, pola retina, dan sebagainya.

2.6.1 Password

Password adalah sekumpulan huruf, angka, dan simbol yang sifatnya rahasia. *Password* merupakan salah satu teknik dari otentikasi entitas. *Password* digunakan saat seseorang hendak mengakses sumber daya sebuah sistem, seperti *email*, akun media sosial, dan sebagainya. *Password* ini sifatnya rahasia dan tidak boleh diketahui oleh pihak yang tidak berhak.

Berdasarkan cara penggunaannya[2], password dibagi menjadi 2 jenis, yaitu:

1. One-Time Password

One-Time Password adalah *password* yang digunakan hanya satu kali untuk setiap akses kepada sistem. Jadi, setiap kali pengguna mengakses sistem dalam sesi waktu yang berbeda, *password* yang digunakan pun akan berbeda-beda. Beberapa contoh dari *One-Time Password* adalah *List of Passwords*, *Sequentially Updated Password*, dan *Lamport One-Time Password*.

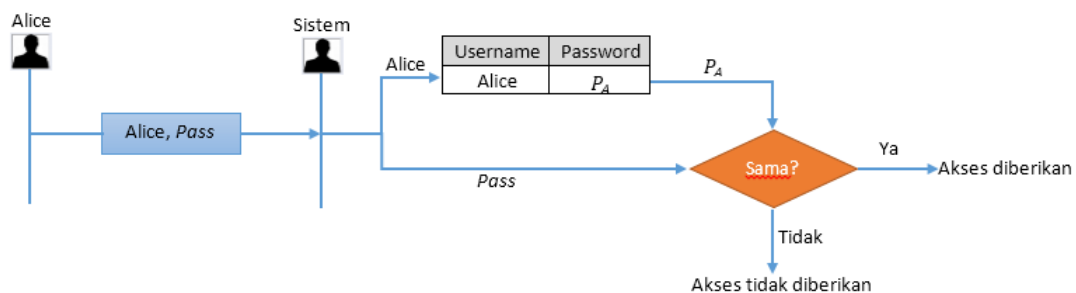
2. Password Tetap

Password tetap adalah *password* yang digunakan berulang-ulang setiap kali pengguna akan mengakses sistem. *Password* yang digunakan untuk mengakses sistem selalu sama. Berikut adalah beberapa skema dari *password* tetap.

Skema 1

Dalam skema ini, sistem menyimpan setiap *password* pada sebuah tabel basis data. *Password* yang disimpan di tabel basis data berupa *plaintext*, artinya bisa dibaca dan dimengerti. Masing-masing dari *password* memiliki *username* yang disimpan juga di tabel basis data. Saat pengguna akan mengakses sistem, pengguna akan memasukkan *username* dan *password*.

Kemudian, saat pengguna sudah memasukkan *username* dan *password*, sistem akan mencari informasi dari pengguna di tabel basis data lewat *username*. Karena setiap *username* memiliki *password*, sistem akan menyesuaikan *username* dan *password* di tabel basis data dengan *username* dan *password* yang dimasukan oleh pengguna saat hendak mengakses sistem. Jika *username* dan *password* yang dimasukan pengguna sesuai dengan *username* dan *password* di tabel basis data maka hak akses sistem akan diberikan. Gambar 2.10 menunjukkan proses yang dijelaskan.



Gambar 2.10: *Username* dan *Password*

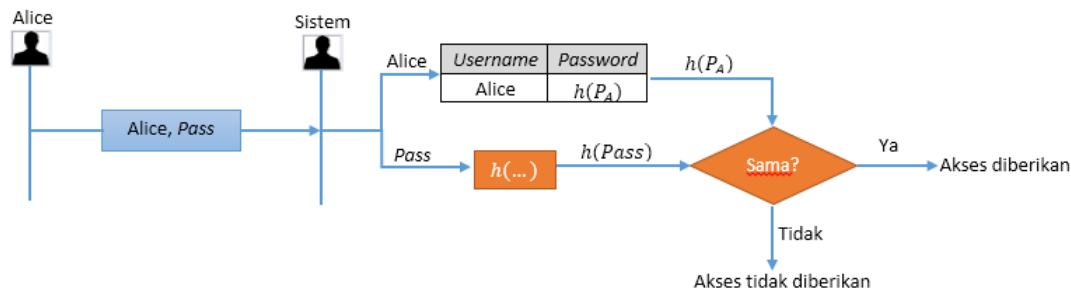
Kelebihan dari skema ini adalah skema ini mudah untuk diimplementasikan dan tidak membutuhkan proses yang rumit. Sementara itu, kekurangan dari skema ini adalah *password*

yang disimpan di tabel basis data bisa dibaca dan dimengerti karena disimpan dalam bentuk *plaintext*. Akibatnya, jika ada pihak yang tidak memiliki hak akses berhasil memperoleh *password* yang disimpan di tabel basis data, maka *password* sudah tidak rahasia lagi.

Skema 2

Dalam skema ini, sistem tetap menyimpan *username* dan *password* dalam tabel basis data. *Password* yang disimpan tidak dalam bentuk *plaintext*nya, tetapi disimpan dalam bentuk *digest*nya. Saat pengguna hendak mengakses sistem, pengguna tetap memasukan *username* dan *password* dalam bentuk *plaintext*.

Kemudian, saat pengguna sudah memasukan *username* dan *password*, sistem akan terlebih dahulu menghitung *digest* dari *password* yang dimasukan menggunakan fungsi *hash*. Setelah itu, *username* dan *digest* akan disesuaikan dengan *username* dan *digest* yang disimpan dalam tabel basis data. Jika sesuai, maka pengguna akan diberikan hak akses ke sistem. Gambar 2.11 menunjukkan proses yang sudah dijelaskan.



Gambar 2.11: Password hashing

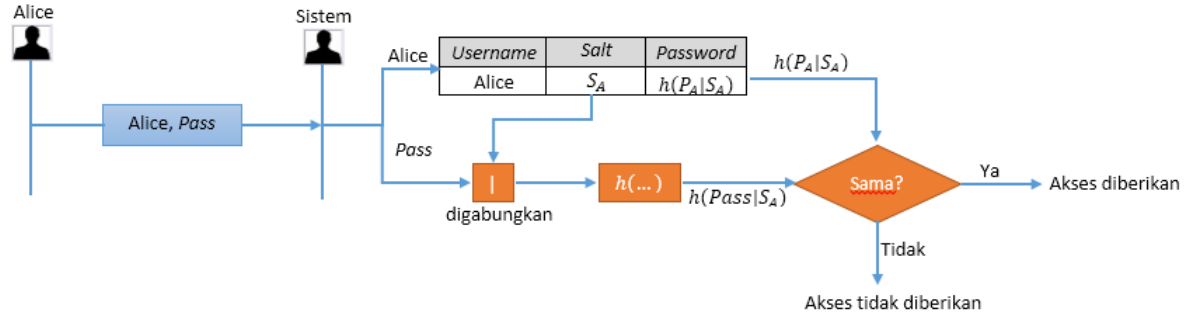
Kelebihan dari skema ini adalah walaupun *password* yang disimpan dalam tabel basis data diketahui oleh pihak yang tidak berhak, *password* tidak akan bisa dimengerti karena disimpan dalam bentuk *digest*nya. Sementara itu, *digest* tidak bisa dikembalikan ke dalam bentuk *plaintext* untuk mendapatkan *password* karena fungsi *hash* adalah fungsi satu arah seperti yang sudah dibahas dalam 2.4. Sementara itu, kekurangan dari skema ini adalah *digest* yang disimpan masih rentan terhadap *dictionary attack*. Penjelasan tentang *dictionary attack* akan dijelaskan pada skema selanjutnya.

Skema 3

Dalam skema 3, sistem tetap menyimpan *username*. *Password* juga disimpan dalam bentuk *digest*nya. Dalam skema ini, sebelum *digest password* dibuat, *password* akan dikonkatenasi dengan *salt*. *Salt* adalah sebuah *string* acak yang bisa berisi angka, huruf, atau simbol.

Penggunaan *salt* disini bertujuan untuk mengurangi tingkat keberhasilan *dictionary attack*. *Dictionary attack* adalah serangan dengan mencoba semua kemungkinan *string* masukan untuk fungsi *hash* sampai menghasilkan *digest* yang sesuai. Dengan adanya penambahan *salt*, maka akan mengurangi kemungkinan keberhasilan dari *dictionary attack* karena banyak kemungkinan dari *string* masukan akan bertambah sehingga semakin sulit untuk mendapatkan *digest* yang sesuai.

1 Karena *salt* dibutuhkan untuk mengurangi tingkat keberhasilan *dictionary attack*, nilai
 2 *salt* akan disimpan juga dalam tabel basis data. Kemudian, saat pengguna sudah memasuk-
 3 an *username* dan *password*, sistem akan menerima *password* yang dimasukan. Selanjutnya,
 4 *password* dikonkatenasi dengan *salt* yang disimpan lalu sistem akan menghitung *digest* dari
 5 hasil konkatenasi *password* dengan *salt*. Setelah itu, sistem akan membandingkan dengan
 6 *digest* yang disimpan dalam tabel basis data. Jika sesuai, pengguna akan diberikan hak
 7 akses ke sistem. Gambar 2.12 menunjukkan proses yang dijelaskan.



Gambar 2.12: *Password salting*

8 Kelebihan dari skema ini adalah *password* tidak akan bisa diketahui dengan mudah
 9 lewat *dictionary attack*. Banyak kemungkinan digest yang semakin bertambah menyebabkan
 10 serangan dengan *dictionary attack* semakin sulit. Sementara itu, kekurangan dari skema ini
 11 adalah rumit karena membutuhkan banyak proses hanya untuk memberikan akses.

12 2.7 Eliminasi Gauss-Jordan

13 Eliminasi Gauss-Jordan adalah suatu metode untuk menyelesaikan sistem persamaan linear
 14 dengan mereduksi matriks menjadi eselon baris tereduksi[3]. Suatu matriks R dikatakan
 15 bentuk eselon baris tereduksi jika memenuhi syarat sebagai berikut[3].

- 16 1. Terdapat baris yang tidak seluruhnya terdiri dari angka 0
- 17 Angka bukan 0 pertama dari sebelah kiri dari baris tersebut disebut 1 utama.
- 18 2. Baris yang seluruhnya terdiri dari angka 0 harus menjadi baris paling bawah.
- 19 3. Pada kolom 1 utama, seluruh angka di bawah 1 utama harus 0.

20 Sebagai contoh, matriks-matriks eselon baris tereduksi ditunjukkan oleh Matriks 2.27.

$$\begin{bmatrix} 1 & 12 & 5 & 4 \\ 0 & 2 & 4 & 8 \\ 0 & 0 & 9 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 2 & 5 \\ 0 & 5 & 4 & 8 \\ 0 & 0 & 4 & 10 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.27)$$

Proses eliminasi Gauss-Jordan dibagi menjadi 2 proses, yaitu proses mereduksi matriks menjadi bentuk eselon baris dan proses substitusi balik ke sistem persamaan linear untuk memperoleh solusi sistem persamaan linear. Diasumsikan sistem persamaan linear yang akan dicari solusinya ditunjukkan oleh persamaan 2.28. Berikut akan dijelaskan proses mereduksi matriks menjadi bentuk eselon baris.

$$\begin{aligned}x + y + z &= 10 \\x + 2y + 4z &= 21 \\x + 3y + 9z &= 38\end{aligned}\tag{2.28}$$

2.7.1 Proses Reduksi Matriks

Proses mereduksi matriks menjadi eselon baris dilakukan dengan cara operasi baris. Operasi baris adalah suatu metode untuk mereduksi matriks menjadi eselon baris dengan cara sebagai berikut.

1. Mengalikan baris dengan konstanta selain 0.
2. Menukar 2 baris.
3. Mengurangi sebuah baris dengan baris lainnya.

Sebagai contoh, diasumsikan bentuk matriks dari persamaan 2.28 ditunjukkan oleh Matriks 2.29.

$$\begin{bmatrix} 1 & 1 & 1 & 10 \\ 1 & 2 & 4 & 21 \\ 1 & 3 & 9 & 38 \end{bmatrix}\tag{2.29}$$

Operasi baris pertama adalah mengurangi baris ke-3 dan baris ke-2 dengan baris ke-1. Maka, hasil pengurangan baris ditunjukkan oleh Matriks 2.30.

$$\begin{bmatrix} 1 & 1 & 1 & 10 \\ 1-1 & 2-1 & 4-1 & 21-10 \\ 1-1 & 3-1 & 9-1 & 38-10 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 10 \\ 0 & 1 & 3 & 11 \\ 0 & 2 & 8 & 28 \end{bmatrix}\tag{2.30}$$

Kemudian, operasi baris kedua adalah mengurangi baris ke-3 dengan baris ke-2 yang dikali dengan konstanta 2. Hasil operasi baris kedua ditunjukkan oleh Matriks 2.31.

$$\begin{bmatrix} 1 & 1 & 1 & 10 \\ 0 & 1 & 3 & 11 \\ 0 & 2 - (1 \cdot 2) & 8 - (3 \cdot 2) & 28 - (11 \cdot 2) \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 10 \\ 0 & 1 & 3 & 11 \\ 0 & 0 & 2 & 6 \end{bmatrix} \quad (2.31)$$

1 Setelah operasi baris kedua, maka diperoleh Matriks 2.32 yang merupakan matriks de-
 2 ngan bentuk eselon baris tereduksi.

$$\begin{bmatrix} 1 & 1 & 1 & 10 \\ 0 & 1 & 3 & 11 \\ 0 & 0 & 2 & 6 \end{bmatrix} \quad (2.32)$$

3 2.7.2 Proses Substitusi Balik

4 Setelah mengubah matriks menjadi bentuk eselon baris tereduksi, proses substitusi balik
 5 adalah proses untuk mencari nilai koefisien dari masing-masing variabel untuk memperoleh
 6 solusi dari persamaan linear 2.28. Kolom paling kanan (kolom ke- n) dari matriks menunjukk-
 7 an nilai solusi dari masing-masing baris. Sementara itu, kolom ke-1 sampai kolom ke- $(n-1)$
 8 menunjukkan koefisien dari persamaan linear.

9 Sebagai contoh, dari Matriks 2.32 diperoleh hasilnya sebagai berikut.

$$\begin{aligned} 2z &= 6 \\ z &= 3 \end{aligned} \quad (2.33)$$

10 Kemudian, untuk nilai y .

$$\begin{aligned} y + 3z &= 11 \\ y + 3 \cdot 3 &= 11 \\ y + 9 &= 11 \\ y &= 2 \end{aligned} \quad (2.34)$$

11 Kemudian, untuk nilai x .

$$\begin{aligned} x + y + z &= 10 \\ x + 2 + 3 &= 10 \\ x + 5 &= 10 \\ x &= 5 \end{aligned} \quad (2.35)$$

Jadi, solusi dari persamaan 2.28 yang diselesaikan dengan eliminasi Gauss-Jordan adalah $x = 5, y = 2$, dan $z = 3$.

2.8 Secret Sharing Shamir

Pada bagian ini akan dijelaskan mengenai sejarah singkat yang mengawali munculnya secret sharing Shamir dan pembahasan mengenai secret sharing Shamir.

2.8.1 Sejarah Singkat

Secret sharing adalah metode untuk membagi informasi (rahasia) menjadi beberapa bagian. Bagian-bagian tersebut disebut *share* dan setiap bagian dibagikan kepada beberapa partisipan. Untuk mendapatkan kembali informasi, maka dibutuhkan setiap *share*.

Permasalahan muncul jika *share* dan partisipan bertambah banyak. Proses untuk mendapatkan kembali rahasia akan menjadi sulit karena setiap *share* harus ada. Karena permasalahan ini, pada tahun 1979 Adi Shamir memublikasikan pengembangan dari metode *secret sharing* dalam esai yg berjudul '*How to Share a Secret*'[4]. Metode yang dikembangkan Adi Shamir dinamakan *secret sharing* Shamir.

2.8.2 Pembahasan Secret Sharing Shamir

Untuk mengatasi permasalahan yang sudah dibahas, Shamir mengubah cara untuk mendapatkan kembali informasi. Misalkan, informasi diasumsikan sebagai data D . Dalam metode *secret sharing* Shamir data D yang dibagi menjadi n *share* hanya memerlukan minimal k *share* untuk memperoleh kembali D . Skema yang dikembangkan Shamir ini dinamakan skema *threshold*(k, n),

Skema *Threshold*(k, n)

Skema *threshold*(k, n) adalah skema *secret sharing* dimana hanya minimal k *share* dari n *share* dibutuhkan untuk mengembalikan data D . Skema ini memiliki ketentuan sebagai berikut[4].

- Jika *share* yang dimiliki sebanyak k *share* atau lebih, D bisa dibentuk kembali.
- Jika *share* yang ada hanya sebanyak $k-1$ atau kurang maka D tidak bisa dibentuk kembali.

Ada 2 proses dalam skema *threshold*(k, n), yaitu proses pembangunan *share* dari rahasia dan proses rekonstruksi rahasia dari *share* yang dimiliki. Diasumsikan rahasia adalah D . Proses pertama adalah proses pembangunan *share* dari D . Berikut akan dijelaskan proses pembangunan *share*.

Proses Pembangunan *Share*

Langkah pertama adalah memilih nilai k . Kemudian, setelah memilih nilai k langkah selanjutnya adalah membentuk $k - 1$ derajat fungsi $f(x)$. Persamaan 2.36 menunjukkan fungsi

$f(x)$ yang dibentuk.

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \quad (2.36)$$

dimana $a_0 = D$.

Setelah membentuk fungsi $f(x)$, langkah selanjutnya adalah memilih banyak share, yaitu nilai n . Setelah memilih n , $x = 1$ sampai $x = n$ akan dipetakan dengan fungsi $f(x)$ untuk memperoleh D_i . Persamaan 2.37 menunjukkan hasil pemetaan dengan fungsi $f(x)$.

$$D_1 = f(1), D_2 = f(2), \dots, D_i = f(i), \dots, D_n = f(n) \quad (2.37)$$

Nilai D_1 sampai D_n adalah *share* dari data D .

Proses Rekonstruksi Rahasia

Pada bagian ini akan dijelaskan proses rekonstruksi D dari D_1, D_2, \dots, D_n yang sudah dibangun dalam Proses Pembangunan *Share*. Langkah pertama adalah membentuk $k - 1$ derajat fungsi $f(x)$ dari k yang sudah dipilih dalam Proses Pembangunan *Share*. Persamaan 2.38 menunjukkan fungsi $f(x)$ yang dibentuk.

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \quad (2.38)$$

Setelah itu, langkah selanjutnya adalah membentuk k fungsi $f(x)$ dengan memetakan k *share* yang dimiliki dengan fungsi $f(x)$. Hasil pemetaan dengan fungsi $f(x)$ ini adalah *share* yang sudah dibangun pada Proses Pembangunan *Share*. Persamaan 2.39 menunjukkan hasil pemetaan masing-masing fungsi $f(x)$.

$$\begin{aligned} f(1) &= a_0 + a_1 \cdot 1 + a_2 \cdot 1^2 + \dots + a_{k-1} \cdot 1^{k-1} = D_1 \\ f(2) &= a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + \dots + a_{k-1} \cdot 2^{k-1} = D_2 \\ &\vdots \\ f(k) &= a_0 + a_1 \cdot k + a_2 \cdot k^2 + \dots + a_{k-1} \cdot k^{k-1} = D_k \end{aligned} \quad (2.39)$$

Dari hasil pemetaan yang ditunjukkan persamaan 2.39, langkah selanjutnya adalah membentuk persamaan linear. persamaan 2.40 menunjukkan persamaan linear yang dibentuk.

$$\begin{aligned} a_0 + a_1 \cdot 1 + a_2 \cdot 1^2 + \dots + a_{k-1} \cdot 1^{k-1} &= D_1 && \dots \textcircled{1} \\ a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + \dots + a_{k-1} \cdot 2^{k-1} &= D_2 && \dots \textcircled{2} \\ &\vdots && \\ a_0 + a_1 \cdot k + a_2 \cdot k^2 + \dots + a_{k-1} \cdot k^{k-1} &= D_k && \dots \textcircled{k} \end{aligned} \quad (2.40)$$

Setelah membentuk persamaan linear, langkah selanjutnya adalah menyelesaikan persamaan linear tersebut dengan metode Eliminasi Gauss-Jordan yang sudah dijelaskan pada

Subbab 2.7. Tujuannya adalah untuk memperoleh nilai a_1, a_2, \dots, a_{k-1} . Dengan menggunakan Proses Substitusi Balik dalam metode Eliminasi Gauss-Jordan, dapat diperoleh nilai a_0 yang adalah data D .

2.9 Probabilitas

Probabilitas atau peluang merupakan salah cara dalam ilmu matematika untuk mengukur tingkat kepercayaan akan suatu kejadian. Teori probabilitas sangat luas penggunaannya, baik dalam kehidupan sehari-hari maupun dalam percobaan-percobaan ilmiah. Teori probabilitas ini seringkali digunakan oleh para pengambil keputusan untuk memprediksi suatu kejadian sehingga nantinya bisa mengambil keputusan yang tepat.

Seluruh kemungkinan keluaran yang akan terjadi dalam probabilitas disebut ruang sampel sedangkan masing-masing kemungkinan yang dapat terjadi dalam ruang sampel dinamakan elemen kejadian atau anggota dari ruang sampel. Ruang sampel dilambangkan dengan huruf S dan elemen kejadian dilambangkan dengan huruf x_i . Dalam ruang sampel S dengan i elemen kejadian, ditunjukkan pada persamaan 2.41.

$$S = x_1, x_2, x_3, \dots, x_i \quad (2.41)$$

Sedangkan probabilitas kejadian x_i akan terjadi dilambangkan dengan $P(x_i)$. Maka, rumus matematikanya ditunjukkan pada persamaan 2.42.

$$P(x_i) = \frac{n}{N} \quad (2.42)$$

dimana n adalah banyaknya kemunculan kejadian x_i dalam sebuah ruang sampel S dan N adalah banyaknya kejadian yang terjadi dalam ruang sampel S .

2.9.1 Distribusi Binom

Setiap eksperimen atau percobaan yang dilakukan secara berkali-kali pasti memiliki dua keluaran, yaitu sukses atau gagal. Untuk setiap keluaran yang diperoleh (baik sukses maupun gagal) bisa ditetapkan sebagai sukses. Proses ini dinamakan proses Bernouli dan setiap eksperimen yang dilakukan untuk setiap proses bernouli dinamakan percobaan Bernouli. Ada beberapa syarat sebuah eksperimen bisa dinamakan percobaan Bernouli[5]:

1. Eksperimen harus diulang sebanyak n kali.
2. Hasil keluaran setiap perulangan hanya 2 kemungkinan, yaitu keluaran sukses atau keluaran gagal.
3. Hasil keluaran setiap perulangan tidak mempengaruhi dengan perulangan yang lain.
4. Probabilitas bahwa hasil keluarannya sukses, p , harus selalu sama untuk setiap kali perulangan.

Percobaan Bernouli digunakan untuk menghitung probabilitas x buah hasil keluaran yang sukses dari n percobaan. Diasumsikan bahwa probabilitas hasil keluaran setiap perulangan sukses adalah p . Sebaliknya, probabilitas hasil keluaran setiap perulangan gagal

1 adalah $q = 1 - p$. Persamaan 2.43 untuk menghitung probabilitas x hasil keluaran yang
 2 sukses dari n percobaan.

$$P(x, n, p) = \binom{n}{x} p^x q^{n-x} \quad (2.43)$$

$$x = 0, 1, 2, \dots, n$$

3 $\binom{n}{x}$ pada persamaan 2.43 menunjukkan bahwa dari n percobaan akan dipilih x hasil
 4 keluaran yang sukses.

5 2.10 Entropi

6 Pada bagian ini akan dijelaskan mengenai entropi dimulai dari sejarah singkat entropi dan
 7 pembahasan mengenai entropi.

8 2.10.1 Sejarah Singkat

9 Istilah entropi muncul pertama kali dalam esai '*A Mathematical Theory of Communication*'
 10 pada tahun 1948. Esai ini dibuat oleh Claude E. Shannon seorang ilmuwan asal Amerika
 11 Serikat. Dalam esainya, Shannon menulis bahwa entropi adalah konsep keacakan atau suatu
 12 ketidakpastian[6]. Istilah dari entropi ini dinamakan Shannon *Entropy*.

13 2.10.2 Pembahasan

14 Entropi adalah rata-rata suatu informasi yang dimiliki oleh sebuah pesan. Informasi yang
 15 dimaksud adalah kejadian yang spesifik atau sebuah elemen tertentu yang dimiliki oleh
 16 pesan. Maka dari itu, entropi bisa dijadikan alat ukur ketidakpastian yang dimiliki oleh
 17 sebuah pesan atau sumber informasi.

18 Nilai entropi yang tinggi menunjukkan bahwa informasi yang dimiliki sebuah pesan cu-
 19 kup tinggi. Nilai informasi yang cukup tinggi memiliki arti bahwa isi dari pesan bisa dip-
 20 rediksi. Sementara itu, jika nilai entropi yang rendah menunjukkan bahwa informasi yang
 21 dimiliki sebuah pesan cukup rendah. Nilai informasi yang cukup rendah memiliki arti bahwa
 22 isi dari pesan tidak bisa dengan mudah diprediksi.

23 Sebagai contoh, nilai entropi akan rendah untuk memastikan panjang umur seseorang
 24 karena tidak bisa diketahui kapan orang tersebut akan meninggal. Contoh yang lain adalah
 25 nilai entropi akan tinggi untuk kasus melemparkan koin karena hasilnya hanya ada dua
 26 kemungkinan yaitu, kepala atau buntut.

27 Dari penjelasan mengenai entropi yang sudah dijelaskan, diasumsikan probabilitas ke-
 28 munculan informasi x_i dalam sebuah pesan X adalah p_i . p_i ditunjukkan oleh persamaan
 29 2.44.

$$P(x_i) = p_i \log\left(\frac{1}{p_i}\right) \quad (2.44)$$

- ¹ Maka, nilai entropi pesan X untuk setiap informasi p_1, p_2, \dots, p_m ditunjukkan oleh per-
² samaan [2.45](#).

$$\begin{aligned} H(X) &= p_1 \log\left(\frac{1}{p_1}\right) + p_2 \log\left(\frac{1}{p_2}\right) + \dots + p_m \log\left(\frac{1}{p_m}\right) \\ &= \sum_{i=1}^m p_i \log\left(\frac{1}{p_i}\right) \end{aligned} \tag{2.45}$$

BAB 3

ANALISIS

Pada bab ini akan dibahas analisis terhadap teori-teori yang telah dibahas sebelumnya. Analisis akan meliputi studi kasus untuk penerapan metode *secret sharing* Shamir, pemilihan n dan k , analisis proses, dan perancangan diagram.

3.1 Studi Kasus

Pada bagian ini akan dibahas studi kasus tentang bagaimana penerapan metode *secret sharing* Shamir untuk banyak *password*. Studi kasus meliputi pengenalan kasus, proses penyimpanan *password*, dan proses rekonstruksi *password*.

3.1.1 Pengenalan Kasus

Langkah awal yang dibutuhkan untuk mengembalikan banyak *password* dengan metode *secret sharing* Shamir, diperlukan beberapa tahap proses. Proses pertama adalah proses penyimpanan *password*. Kemudian, proses selanjutnya adalah proses untuk mengembalikan banyak *password*. Proses pertama membutuhkan beberapa *password*. Untuk n buah *password*, maka akan dibuat n buah pertanyaan keamanan. Sementara itu, untuk proses mengembalikan *password* dibutuhkan pertanyaan keamanan yang sudah dibuat dalam proses sebelumnya.

Untuk kedua proses di atas, diasumsikan banyak *password* yang akan disimpan sebanyak 5 buah. Setiap *password* akan diberi label p_1 , p_2 , sampai p_5 . Persamaan 3.1 sampai 3.5 menunjukkan p_1 sampai p_5 .

$$p_1 = 123456 \quad (3.1)$$

$$p_2 = \textit{password} \quad (3.2)$$

$$p_3 = \textit{hello123} \quad (3.3)$$

$$p_4 = \textit{secret} \quad (3.4)$$

$$p_5 = \textit{foobar} \quad (3.5)$$

3.1.2 Proses Penyimpanan *Password*

Proses penyimpanan *password* dibagi menjadi 2 proses, yaitu proses pembangunan *share* untuk masing-masing *password* dan proses enkripsi dari setiap *share* yang sudah dibangun. Pada bagian ini akan dibahas kedua proses tersebut.

1 Proses Pembangunan *Share*

2 Pada proses ini, akan dilakukan pembangunan *share* dari masing-masing *password*. Langkah-
3 langkah untuk membangun *share* adalah sebagai berikut.

- 4 1. Membagi setiap *password* p_i menjadi beberapa karakter, masing-masing karakter akan
5 diubah menjadi nilai ASCII-nya, c_1, c_2, \dots, c_m .
- 6 2. Memilih nilai n , yaitu banyak *share* yang akan dibangun.
- 7 3. Memilih nilai k , yaitu banyak minimal pertanyaan keamanan yang harus dijawab de-
8 ngan benar, dimana $0 < k \leq n$.
- 9 4. Memilih $k - 1$ angka acak, d_1, d_2, \dots, d_{k-1} , untuk masing-masing karakter c_1, c_2, \dots, c_m .
5. Membentuk fungsi $f_m(x)$ untuk masing-masing karakter c_1, c_2, \dots, c_m . Komponen dari
fungsi $f_m(x)$ terdiri atas c_m sebagai konstanta tanpa koefisien, d_1, d_2, \dots, d_{k-1} sebagai
konstanta dengan koefisien. Persamaan 3.6 menunjukkan persamaan dari fungsi $f_m(x)$
yang harus dibentuk.

$$f_m(x) = c_m + d_1x + d_2x^2 + d_3x^3 + \dots + d_{k-1}x^{k-1} \quad (3.6)$$

- 10 6. Menghitung masing-masing nilai x dari $x = 1, x = 2, \dots, x = n$ untuk fungsi $f_m(x)$.
- 11 7. Nilai $f_m(1)$ sampai $f_m(n)$ adalah nilai *share* untuk password p_i .

12 Kembali kepada kasus pada Subbab 3.1.1, misalkan *password* yang akan dibangun *share*-
13 *sharenya* adalah p_1 . Langkah pertama adalah membagi p_1 menjadi beberapa karakter dan
14 mengubah masing-masing karakter menjadi nilai ASCII-nya. Persamaan 3.7 sampai 3.13
15 menunjukkan langkah pertama.

$$p_1 = 123456 \quad (3.7)$$

$$c_1 = '1' = 49 \quad (3.8)$$

$$c_2 = '2' = 50 \quad (3.9)$$

$$c_3 = '3' = 51 \quad (3.10)$$

$$c_4 = '4' = 52 \quad (3.11)$$

$$c_5 = '5' = 53 \quad (3.12)$$

$$c_6 = '6' = 54 \quad (3.13)$$

16 Langkah selanjutnya adalah memilih nilai n . Karena banyak *password* p_i adalah 5, maka
17 banyak *share* untuk masing-masing *password* sebanyak 5. Maka, $n = 5$.

18 Setelah memilih nilai n , langkah berikutnya adalah memilih nilai k . Nilai k ini nanti
19 akan berhubungan dengan banyak minimal pertanyaan keamanan yang harus dijawab benar
20 untuk mengembalikan *password*. Untuk kasus ini, dipilih $k = 3$.

21 Langkah selanjutnya adalah memilih $k - 1$ angka acak untuk masing-masing karakter c_1
22 sampai c_6 . Karena $k = 3$, maka dipilih 2 angka acak untuk masing-masing karakter. Berikut
23 angka acak untuk masing-masing karakter.

- 1 • c_1 : 12 dan 6.
- 2 • c_2 : 15 dan 11.
- 3 • c_3 : 22 dan 1.
- 4 • c_4 : 21 dan 3.
- 5 • c_5 : 19 dan 8.
- 6 • c_6 : 25 dan 17.

7 Setelah memilih angka acak untuk masing-masing karakter, langkah selanjutnya adalah
 8 membentuk fungsi $f(x)$ untuk masing-masing karakter. Maka, fungsi $f_1(x)$ sampai $f_6(x)$
 9 yang dibentuk adalah sebagai ditunjukkan pada persamaan 3.14 sampai 3.19.

$$f_1(x) = 49 + 12x + 6x^2 \quad (3.14)$$

$$f_2(x) = 50 + 15x + 11x^2 \quad (3.15)$$

$$f_3(x) = 51 + 22x + x^2 \quad (3.16)$$

$$f_4(x) = 52 + 21x + 3x^2 \quad (3.17)$$

$$f_5(x) = 53 + 19x + 8x^2 \quad (3.18)$$

$$f_6(x) = 54 + 25x + 17x^2 \quad (3.19)$$

10 Langkah selanjutnya adalah menghitung nilai $x = 1, x = 2, \dots, x = n$ untuk fungsi $f_1(x)$
 11 sampai $f_6(x)$. Tabel 3.1 menunjukkan nilai $x = 1$ sampai $x = 5$ untuk masing-masing fungsi
 12 $f(x)$.

Tabel 3.1: Nilai x untuk masing-masing $f(x)$

| | $f_1(x)$ | $f_2(x)$ | $f_3(x)$ | $f_4(x)$ | $f_5(x)$ | $f_6(x)$ |
|---|----------|----------|----------|----------|----------|----------|
| 1 | 67 | 76 | 74 | 76 | 80 | 96 |
| 2 | 97 | 124 | 99 | 106 | 123 | 172 |
| 3 | 139 | 194 | 126 | 142 | 182 | 282 |
| 4 | 193 | 286 | 155 | 184 | 257 | 426 |
| 5 | 259 | 400 | 186 | 232 | 348 | 604 |

13 Setiap nilai x pada Tabel 3.1 adalah nilai *share-share* untuk password p_1 . Setiap nilai x
 14 ini akan diberi label s_{11} untuk *share* pertama dari fungsi pertama, s_{21} untuk *share* kedua
 15 dari fungsi pertama, dan seterusnya sampai s_{56} untuk *share* kelima dari fungsi keenam. Nilai
 16 *share* yang sudah diberi label ditunjukkan pada persamaan 3.20.

$$s_{11} = 67, s_{21} = 97, \dots, s_{34} = 155, \dots, s_{56} = 604 \quad (3.20)$$

17 Sementara itu, untuk menghitung nilai *share* dari *password* p_2 sampai p_5 , proses yang
 18 sama untuk menghitung *password* p_1 akan dilakukan. Setelah menghitung nilai *share* untuk
 19 *password* p_1 , langkah selanjutnya adalah proses enkripsi masing-masing *share* ini.

1 Proses Enkripsi *Share*

2 Pada proses ini, sebelum masing-masing *share* disimpan, masing-masing *share* harus dienkripsi terlebih dahulu. Dalam proses ini juga, n buah pertanyaan keamanan akan dibuat.
3
4 Langkah-langkah proses enkripsi *share* adalah sebagai berikut.

- 5 1. Membuat n pertanyaan keamanan, q_1, q_2, \dots, q_n .
- 6 2. Menentukan jawaban dari masing-masing pertanyaan keamanan, a_1, a_2, \dots, a_n .
- 7 3. Menentukan nilai *salt*, r_s .
4. Menghitung *digest* untuk masing-masing konkatenasi dari pertanyaan, jawaban, dan *salt*. Persamaan 3.21 menunjukkan proses menghitung *digest*.

$$h_n = H(q_n + a_n + r_s) \quad (3.21)$$

5. Setiap nilai *share*, $s_{11}, s_{21}, \dots, s_{56}$ akan dienkripsi dengan menggunakan *digest* sebagai kunci. Persamaan 2.1 menunjukkan langkah enkripsi *share*.

$$E_{h_n}(s_{nm}) = c_{nm} \quad (3.22)$$

8 Pada persamaan 2.1, m merupakan banyak karakter dari masing-masing *password* p_i .

9 Kembali kepada kasus pada Subbab 3.1.1, misalkan *password* yang akan dienkripsi *share*-
10 *sharenya* adalah p_1 . Langkah pertama adalah membuat n pertanyaan keamanan, karena
11 $n = 5$ maka ada 5 pertanyaan keamanan. Setiap pertanyaan keamanan akan diberi label
12 q_1, q_2, \dots, q_5 . Untuk kasus ini, diasumsikan pertanyaan keamanan yang dibuat adalah sebagai
13 berikut.

- 14 1. Siapa nama anda? (q_1)
- 15 2. Dimana kota tempat anda tinggal? (q_2)
- 16 3. Apa jenis kelamin anda? (q_3)
- 17 4. Pada bulan apa anda lahir? (q_4)
- 18 5. Apa nama belakang anda? (q_5)

19 Setelah membuat pertanyaan keamanan yang akan digunakan, langkah selanjutnya ada-
20 lah menentukan jawaban dari masing-masing pertanyaan keamanan. Setiap jawaban untuk
21 pertanyaan keamanan akan diberi label a_1 untuk q_1 , a_2 untuk q_2 , dan seterusnya sampai a_5
22 untuk q_5 . Jawaban dari masing-masing pertanyaan keamanan adalah sebagai berikut.

- 23 1. Samuel (a_1)
- 24 2. Bandung (a_2)
- 25 3. Laki-laki (a_3)
- 26 4. Juli (a_4)
- 27 5. Christian (a_5)

- 1 Langkah selanjutnya adalah memilih nilai *salt*, r_s . Untuk kasus ini, misalkan $r_s = 31$.
 2 Setelah memilih nilai *salt*, langkah selanjutnya adalah menghitung *digest*. Masing-masing
 3 dari pertanyaan keamanan akan dikonkatenasi dengan jawabannya dan r_s . Asumsi hasil
 4 penghitungan *digest*, h_n , untuk setiap pertanyaan ditunjukkan pada persamaan 3.23 sampai
 5 3.27.

$$h_1 = (q_1 + a_1 + r_s) = 7a916 \quad (3.23)$$

$$h_2 = (q_2 + a_2 + r_s) = cdc62 \quad (3.24)$$

$$h_3 = (q_3 + a_3 + r_s) = de09b \quad (3.25)$$

$$h_4 = (q_4 + a_4 + r_s) = d1320 \quad (3.26)$$

$$h_5 = (q_5 + a_5 + r_s) = b59e9 \quad (3.27)$$

- 6 Langkah selanjutnya adalah mengenkripsi setiap nilai *share* yang sudah dibangun dengan
 7 *digest* yang sudah dihitung sebagai kuncinya. Asumsi hasil enkripsi setiap *share* untuk p_1 ,
 8 ditunjukkan pada Tabel 3.2.

Tabel 3.2: Hasil Enkripsi setiap *Share* untuk *Password* Pertama

| | c_1 | c_2 | c_3 | c_4 | c_5 | c_6 |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|
| s_1 | <i>aa7cm</i> | <i>a45sf</i> | <i>1xz5q</i> | <i>x15z6</i> | <i>cx96v</i> | <i>6zx51</i> |
| s_2 | <i>ff3ds</i> | <i>5cv1s</i> | <i>rf51s</i> | <i>xcq89</i> | <i>a9er8</i> | <i>9wrt8</i> |
| s_3 | <i>fg9e5</i> | <i>afa65</i> | <i>ge65r</i> | <i>we65q</i> | <i>s6dv5</i> | <i>xf8xj</i> |
| s_4 | <i>d3d64</i> | <i>eq89v</i> | <i>85vbn</i> | <i>nm6f5</i> | <i>51gvq</i> | <i>x91qw</i> |
| s_5 | <i>a54q1</i> | <i>z1x56</i> | <i>as46c</i> | <i>na6e5</i> | <i>cz98q</i> | <i>ha658</i> |

- 9 Kolom pada Tabel 3.2 menunjukkan urutan karakter dari *password*. Sementara itu, baris
 10 menunjukkan urutan *share* dari masing-masing karakter. Sebagai contoh, kolom c_1 baris s_1
 11 menunjukkan hasil enkripsi untuk *share* pertama dari karakter pertama *password* p_1 .

- 12 Langkah enkripsi setiap *share* ini dilakukan untuk setiap *password* p_2 sampai p_5 . Kemu-
 13 dian setelah proses enkripsi ini, pertanyaan keamanan, jawaban, hasil enkripsi (*ciphertext*),
 14 nilai *salt*, dan nilai k akan disimpan.

15 3.1.3 Proses Pengembalian *Password*

- 16 Setelah *password* disimpan dalam proses Penyimpanan *Password* (Subbab 3.1.2), pada ba-
 17 gian ini akan dijelaskan proses bagaimana *password* bisa dikembalikan dengan menggunakan
 18 metode *secret sharing* Shamir. Proses pengembalian *password* ini dibagi menjadi 2 proses,
 19 yaitu proses dekripsi setiap *share* dan proses rekonstruksi kembali *password* dari *share-share*
 20 yang sudah didekripsi.

21 Proses Dekripsi *Share*

- 22 Proses dekripsi *share* adalah proses mengembalikan *ciphertext* dari masing-masing *share*
 23 kembali kepada bentuk *plaintext*nya. Langkah-langkah dari proses dekripsi *share* adalah
 24 sebagai berikut.

1 1. Menjawab n pertanyaan keamanan yang sebelumnya disimpan, q_1, q_2, \dots, q_n untuk
 2 menghasilkan jawaban a'_1, a'_2, \dots, a'_n .

2. Menghitung *digest* untuk masing-masing konkatenasi dari pertanyaan yang disimpan, jawaban, dan *salt* yang disimpan. Persamaan 3.28 menunjukkan proses menghitung *digest*.

$$h'_n = H(q_n + a'_n + r_s) \quad (3.28)$$

3. Mendekripsi $c_{11}, c_{21}, \dots, c_{nm}$ dengan menggunakan h'_1, h'_2, \dots, h'_n sebagai kunci. Persamaan 3.29 menunjukkan langkah yang dijelaskan.

$$D_{h'_n}(c_{nm}) = s'_{nm} \quad (3.29)$$

3 Kembali kepada kasus yang dijelaskan pada Subbab 3.1.1, langkah pertama adalah men-
 4 jawab pertanyaan keamanan yang sebelumnya disimpan. Berikut pertanyaan keamanan
 5 yang disimpan dan jawaban untuk masing-masing pertanyaan keamanan.

- 6 1. Siapa nama anda? (q_1): Samuel (a'_1)
- 7 2. Dimana kota tempat anda tinggal? (q_2): Bandung (a'_2)
- 8 3. Apa jenis kelamin anda? (q_3): Laki-laki (a'_3)
- 9 4. Pada bulan apa anda lahir? (q_4): Juli (a'_4)
- 10 5. Apa nama belakang anda? (q_5): Christian (a'_5)

11 Kemudian, langkah selanjutnya adalah menghitung *digest* masing-masing konkatenasi
 12 dari pertanyaan yang disimpan, jawaban, dan *salt* yang disimpan, $r_s = 31$. Asumsi hasil
 13 penghitungan *digest*, h'_n , untuk setiap pertanyaan ditunjukkan pada persamaan 3.30 sampai
 14 3.34.

$$h'_1 = (q_1 + a'_1 + r_s) = 7a916 \quad (3.30)$$

$$h'_2 = (q_2 + a'_2 + r_s) = cdc62 \quad (3.31)$$

$$h'_3 = (q_3 + a'_3 + r_s) = de09b \quad (3.32)$$

$$h'_4 = (q_4 + a'_4 + r_s) = d1320 \quad (3.33)$$

$$h'_5 = (q_5 + a'_5 + r_s) = b59e9 \quad (3.34)$$

15 Setelah memperoleh *digest*, langkah selanjutnya adalah mendekripsi setiap *share* dalam
 16 Tabel 3.2 dengan *digest* h'_1, h'_2, \dots, h'_5 sebagai kunci. Persamaan 3.35 dan 3.36 menunjukkan
 17 langkah dari dekripsi salah satu *share*.

$$c_{11} = aa7cm \quad (3.35)$$

$$D_{h'_1}(c_{11}) = s_{11} = 67 \quad (3.36)$$

18 Kemudian, proses dekripsi diulang untuk setiap *share* dari *password* p_1 . Tabel 3.3 me-
 19 nunjukkan hasil dari dekripsi setiap *share*.

Tabel 3.3: Hasil Dekripsi *Share*

| | c_1 | c_2 | c_3 | c_4 | c_5 | c_6 |
|-------|-------|-------|-------|-------|-------|-------|
| s_1 | 67 | 76 | 74 | 76 | 80 | 96 |
| s_2 | 97 | 124 | 99 | 106 | 123 | 172 |
| s_3 | 139 | 194 | 126 | 142 | 182 | 282 |
| s_4 | 193 | 286 | 155 | 184 | 257 | 426 |
| s_5 | 259 | 400 | 186 | 232 | 348 | 604 |

Kolom pada Tabel 3.3 menunjukkan urutan karakter dari *password* p_1 , sedangkan baris pada Tabel 3.3 menunjukkan urutan *share* untuk masing-masing karakter. Sebagai contoh, baris s_1 kolom c_1 menunjukkan *share* pertama untuk karakter pertama dari *password* p_1 dan seterusnya sampai baris s_5 kolom c_6 menunjukkan *share* kelima untuk karakter keenam *password* p_1 .

6 Proses Rekonstruksi *Password*

Setelah memperoleh hasil dekripsi *share* untuk masing-masing karakter dari masing-masing *password* p_1 sampai p_5 , proses selanjutnya adalah proses rekonstruksi masing-masing *password*. Dalam kasus ini, *password* yang akan direkonstruksi adalah p_1 . Berikut langkah-langkah dari rekonstruksi p_i .

1. Membentuk fungsi dasar $f(x)$ untuk masing-masing karakter dari *password* p_i berdasarkan nilai k yang disimpan. Nilai k mempengaruhi derajat dari fungsi $f(x)$ yang akan dibentuk. Persamaan 3.37 menunjukkan fungsi $f(x)$ yang akan dibentuk.

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \quad (3.37)$$

2. Setiap karakter dari *password* p_i diwakili oleh 1 fungsi $f(x)$. Maka, untuk setiap karakter dibentuk fungsi $f_m(x)$ masing-masing, dimana m adalah banyak karakter dari *password* p_i . Persamaan 3.38 menunjukkan langkah yang dijelaskan.

$$f_m(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \quad (3.38)$$

3. Menghitung nilai *share* yang dimiliki untuk masing-masing fungsi $f(x)$ setiap karakter. Persamaan 3.39 menunjukkan langkah yang dijelaskan.

$$f_m(n) = a_0 + a_1n + a_2n^2 + \dots + a_{k-1}n^{k-1} = s_{nm} \quad (3.39)$$

4. Menghitung konstanta bebas dari berdasarkan fungsi $f(x)$ yang ada untuk masing-masing karakter, dari $f_1(x), f_2(x), \dots, f_m(x)$.

5. Mengubah konstanta bebas yang diperoleh dari langkah sebelumnya menjadi karakter ASCII.

Setelah diperoleh nilai setiap *share* yang ditunjukkan pada Tabel 3.3, langkah pertama yang dilakukan untuk mengembalikan *password* adalah membentuk fungsi dasar $f(x)$ untuk

- 1 masing-masing karakter dari password p_i berdasarkan nilai k yang disimpan. Dalam kasus
 2 Subbab 3.1.1, k yang dipilih adalah $k = 3$, maka fungsi $f(x)$ yang dibentuk memiliki derajat
 3 $k - 1$. Persamaan 3.40 menunjukkan fungsi $f(x)$ yang dibentuk.

$$f(x) = c + bx + ax^2 \quad (3.40)$$

- 4 Langkah selanjutnya adalah membentuk fungsi $f(x)$ untuk setiap karakter *password* p_1 .
 5 Persamaan 3.41 sampai 3.46 menunjukkan fungsi $f(x)$ untuk setiap karakter *password* p_1 .

$$f_1(x) = c + bx + ax^2 \quad (3.41)$$

$$f_2(x) = c + bx + ax^2 \quad (3.42)$$

$$f_3(x) = c + bx + ax^2 \quad (3.43)$$

$$f_4(x) = c + bx + ax^2 \quad (3.44)$$

$$f_5(x) = c + bx + ax^2 \quad (3.45)$$

$$f_6(x) = c + bx + ax^2 \quad (3.46)$$

- 6 Setelah itu, langkah selanjutnya adalah menghitung nilai *share* yang dimiliki pada fungsi
 7 $f(x)$ yang sudah dibentuk. Untuk langkah ini, akan ditunjukkan proses pengembalian salah
 8 satu karakter dari *password* p_1 , yaitu karakter pertama.

- 9 Diasumsikan *share* yang digunakan untuk rekonstruksi karakter pertama adalah s_{11}, s_{21} ,
 10 dan s_{31} . Maka, nilai masing-masing *share* ini pada fungsi $f_1(x)$ ditunjukkan pada persamaan
 11 3.47 sampai 3.49.

$$f_1(1) = c + b + a = 67 \quad (3.47)$$

$$f_1(2) = c + 2b + 4a = 97 \quad (3.48)$$

$$f_1(3) = c + 3b + 9a = 139 \quad (3.49)$$

- 12 Langkah selanjutnya adalah menghitung konstanta bebas, yaitu dalam kasus ini konstan-
 13 ta bebas c . Proses eliminasi Gauss-Jordan digunakan dalam menghitung konstanta bebas.
 14 Langkah pertama adalah transformasi $f_1(x), f_2(x)$, dan $f_3(x)$ menjadi matriks. Matriks 3.50
 15 menunjukkan hasil tranformasi $f_1(x), f_2(x)$, dan $f_3(x)$.

$$\begin{bmatrix} 1 & 1 & 1 & 67 \\ 1 & 2 & 4 & 97 \\ 1 & 3 & 9 & 139 \end{bmatrix} \quad (3.50)$$

- 16 Kolom paling kanan dari Matriks 3.50 menunjukkan nilai $f_1(x), f_2(x)$, dan $f_3(x)$, se-
 17 dangkan kolom lainnya menunjukkan nilai koefisien dari setiap variabel dalam $f_1(x), f_2(x)$,
 18 dan $f_3(x)$. Kemudian, setiap baris akan diberi label. Baris 1 diberi label L_1 , baris 2 diberi
 19 label L_2 , dan baris 3 diberi label L_3 .

1 Setelah transformasi matriks, langkah selanjutnya adalah operasi setiap baris untuk
 2 memperoleh matriks bentuk eselon baris tereduksi. Operasi pertama yang dilakukan di-
 3 tunjukkan oleh persamaan 3.51.

$$\begin{aligned} L_3 - L_1 \\ L_2 - L_1 \end{aligned} \quad (3.51)$$

4 Operasi pertama menghasilkan Matriks 3.52.

$$\begin{bmatrix} 1 & 1 & 1 & 67 \\ 0 & 1 & 3 & 30 \\ 0 & 2 & 8 & 72 \end{bmatrix} \quad (3.52)$$

5 Langkah selanjutnya adalah operasi baris kembali sampai memperoleh matriks bentuk
 6 eselon baris tereduksi. Operasi kedua ditunjukkan pada persamaan 3.53.

$$L_3 - 2L_2 \quad (3.53)$$

7 Operasi kedua menghasilkan Matriks 3.54.

$$\begin{bmatrix} 1 & 1 & 1 & 67 \\ 0 & 1 & 3 & 30 \\ 0 & 0 & 2 & 12 \end{bmatrix} \quad (3.54)$$

8 Setelah operasi kedua, diperoleh matriks segitiga atas yang ditunjukkan oleh Matriks
 9 3.54. Langkah selanjutnya setelah memperoleh matriks bentuk eselon baris tereduksi adalah
 10 substitusi balik untuk memperoleh masing-masing nilai koefisien untuk setiap variabel $a, b,$
 11 dan c . Proses substitusi balik pertama adalah untuk memperoleh nilai a . Persamaan 3.55
 12 menunjukkan proses substitusi balik pertama.

$$\begin{aligned} 2a &= 12 \\ a &= 6 \end{aligned} \quad (3.55)$$

13 Proses substitusi balik kedua adalah untuk memperoleh nilai b . Proses substitusi balik
 14 kedua ditunjukkan pada persamaan 3.56.

$$\begin{aligned}
b + 3a &= 30 \\
b + 3 \cdot 6 &= 30 \\
b + 18 &= 30 \\
b &= 12
\end{aligned} \tag{3.56}$$

1 Proses substitusi balik ketiga adalah untuk memperoleh nilai c . Proses substitusi balik
2 ketiga ditunjukkan pada persamaan 3.57.

$$\begin{aligned}
c + b + a &= 67 \\
c + 12 + 6 &= 67 \\
c + 18 &= 67 \\
c &= 49
\end{aligned} \tag{3.57}$$

3 Setelah proses substitusi balik ketiga diperoleh konstanta bebas $c = 49$ untuk karak-
4 ter pertama. Langkah selanjutnya setelah memperoleh konstanta bebas adalah mengubah
5 konstanta bebas menjadi karakter ASCII. Karakter ASCII ke-49 adalah '1'. Maka, untuk
6 karakter pertama dari p_1 adalah '1'.

7 Proses yang sama akan dilakukan untuk karakter kedua, ketiga, sampai karakter keenam.
8 Setelah semua karakter diperoleh, setiap karakter akan dikonkatenasi menjadi sebuah *string*.
9 Maka, hasil akhir dari p_1 ditunjukkan pada persamaan 3.58.

$$p_1 = 123456 \tag{3.58}$$

10 3.2 Pemilihan Nilai n dan k

11 Pengguna dapat memilih n dan k sesuai dengan kebutuhan. Pemilihan n dan k yang baik,
12 tidak hanya dapat membuat *password* tidak akan mudah dikembalikan oleh pihak yang tidak
13 berhak, tetapi dapat juga membuat pengguna bisa dengan mudah mengembalikan *passwo-*
14 *rd*[7]. Pada bagian ini, akan dijelaskan bagaimana pemilihan n dan k dapat mempengaruhi
15 kedua hal tersebut.

16 3.2.1 Pemilihan Nilai k

17 Nilai k adalah banyak minimal pertanyaan benar yang perlu dijawab agar bisa memperoleh
18 *password*. Setiap dari pertanyaan keamanan memiliki kemungkinan jawabannya masing-
19 masing. Setiap pertanyaan keamanan ini memiliki nilai entropi e_i dilihat dari kemungkinan
20 jawaban setiap pertanyaan. Jenis pertanyaan keamanan yang dibuat akan mempengaruhi
21 nilai entropi e_i .

22 Maka dengan bertambahnya nilai k dan diasumsikan e_i dari setiap pertanyaan sangat
23 kecil, maka kemungkinan jawaban dari setiap pertanyaan akan bervariasi sehingga jawaban
24 dari masing-masing pertanyaan keamanan tidak bisa dengan mudah ditebak atau dipre-
25 diksi. Namun, nilai k yang terlalu besar juga akan menyulitkan pemilik *password* untuk

- 1 mengembalikan *password* karena semakin banyak pertanyaan yang harus dijawab dengan
2 tepat.

3 3.2.2 Pemilihan Nilai n

- 4 Nilai n adalah banyak pertanyaan keamanan yang dibuat. Banyak pertanyaan yang dibuat,
5 n , memiliki hubungan yang erat dengan kemampuan pengguna menjawab setiap pertanyaan
6 dengan benar. Diasumsikan jika probabilitas sebuah pertanyaan dijawab dengan benar
7 adalah P_0 . Maka, probabilitas bahwa pengguna bisa menjawab benar k pertanyaan dari n
8 pertanyaan[7]:

$$P_1(k, n, P_0) = \binom{n}{k} P_0^k (1 - P_0)^{n-k} \quad (3.59)$$

- 9 Maka, dari persamaan 3.59 untuk n pertanyaan, probabilitas dari P_i akan bertambah
10 besar dengan bertambah besarnya nilai k . Melalui hal tersebut, dapat disimpulkan proba-
11 bilitas pengguna akan berhasil mengembalikan *password*[7]:

$$P_2(k, n, P_0) = \sum_{i=k}^n P_1(i, n, P_0) \quad (3.60)$$

- 12 P_2 akan mempengaruhi pasangan n dan k yang harus dipilih. Jika diasumsikan $P_0 = 0.95$
13 dan $P_2 = 0.99998$, maka pasangan n dan k yang ideal dari hasil penghitungan menggunakan
14 persamaan 3.60[7]:

Tabel 3.4: Tabel Pasangan n dan k

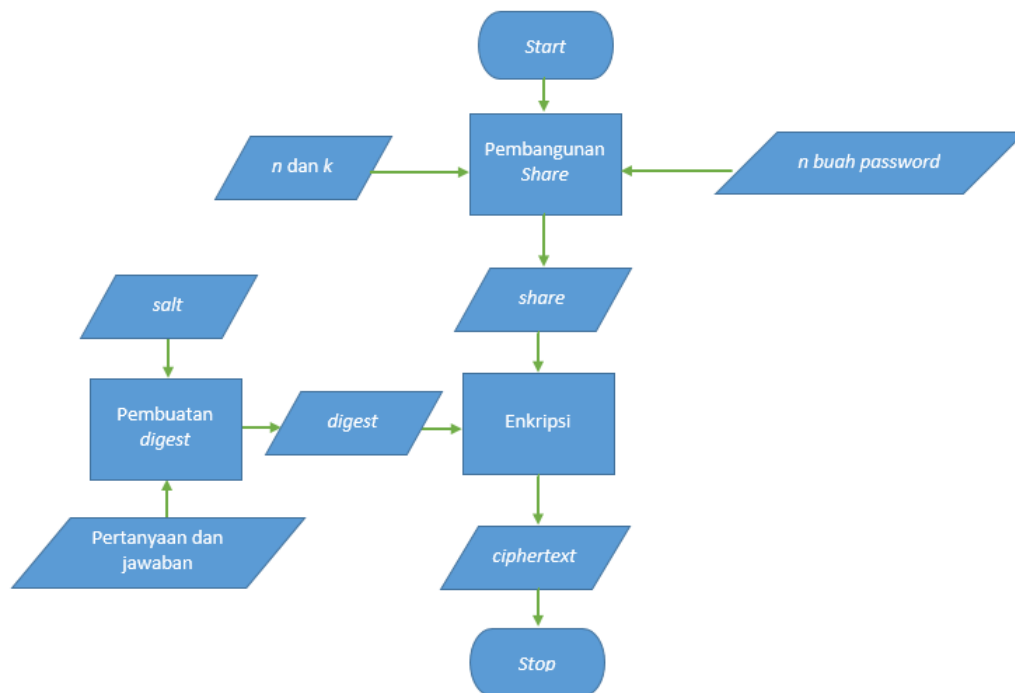
| n | k | n | k | n | k |
|-----|-------|-----|-------|-----|-------|
| 5 | 1 | 6 | 1 | 7 | 1..2 |
| 8 | 1..3 | 9 | 1..3 | 10 | 1..4 |
| 11 | 1..5 | 12 | 1..6 | 13 | 1..7 |
| 14 | 1..7 | 15 | 1..8 | 16 | 1..9 |
| 17 | 1..10 | 18 | 1..11 | 19 | 1..11 |
| 20 | 1..12 | 21 | 1..13 | 22 | 1..14 |
| 23 | 1..15 | 24 | 1..16 | 25 | 1..17 |
| 26 | 1..17 | 27 | 1..18 | 28 | 1..19 |
| 29 | 1..20 | 30 | 1..21 | ... | ... |

15 3.3 Analisis Proses

- 16 Pada bagian ini akan dijelaskan mengenai perancangan perangkat lunak yang akan diba-
17 ngun berdasarkan pada proses-proses yang sudah dipaparkan pada bagian sebelumnya. Pa-
18 da bagian sebelumnya sudah dibahas mengenai proses penyimpanan *password* dan proses
19 rekonstruksi *password*. Pada bagian ini akan dibahas setiap tahapan dari proses-proses ter-
20 sebut dan setiap tahapan akan digambarkan dalam bentuk alur proses (*flowchart*) sebelum
21 dilakukan proses pembuatan diagram-diagram untuk membangun perangkat lunak.

3.3.1 Proses Penyimpanan *Password*

- Pada bagian ini akan dijelaskan tahapan dari proses penyimpanan *password*. Flowchart dari proses penyimpanan password ditunjukkan pada Gambar 3.1.



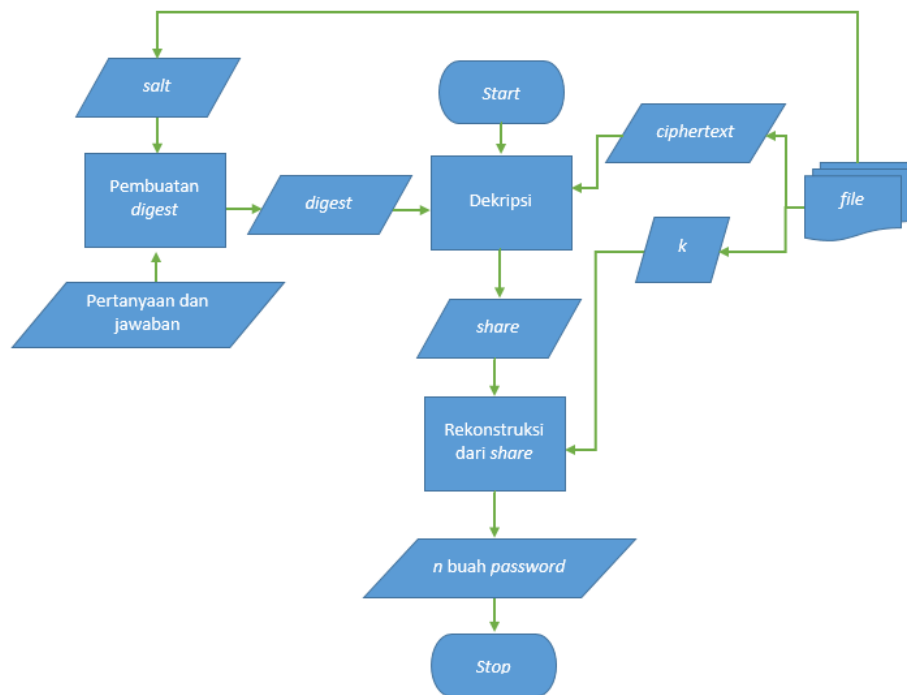
Gambar 3.1: Proses Penyimpanan *Password*

- Tahapan-tahapan pada proses yang ditunjukkan oleh Gambar 3.1 adalah tahapan-tahapan untuk menyimpan *password*. Dalam penelitian ini, tahapan pembangunan *share* menggunakan metode *secret sharing* Shamir. Berikut penjelasan masing-masing tahapan:

- Tahap Pembangunan *Share*
 Pada tahap ini, dibutuhkan data-data masukan seperti n , k , dan n *password*. *Password* diperoleh dari masukan pengguna. n diperoleh dari banyak *password* yang dimasukan pengguna. k ditentukan dengan cara yang sudah dijelaskan pada Subbab 3.2.
- Tahap Pembuatan Digest
 Pada tahap ini, dibutuhkan data-data masukan juga seperti pertanyaan keamanan, jawaban dari pertanyaan keamanan, dan *salt*. Pertanyaan keamanan dan jawaban dari pertanyaan keamanan diperoleh dari masukan pengguna. Sementara itu, *salt* dipilih secara acak. Setelah itu, pertanyaan, jawaban, dan *salt* akan dikonkatenasi menjadi sebuah *string* yang akan dibuat *digest*nya. Dalam tahap ini, pembuatan *digest* akan menggunakan algoritma *SHA-512* (Subbab 2.5).
- Tahap Enkripsi
 Pada tahap ini, *share* yang dihasilkan pada tahap pembangunan *share* akan dienkrpsi dengan menggunakan *Data Encryption Standard* (Subbab 2.3). *Digest* yang dihasilkan pada tahap pembuatan *digest* akan digunakan sebagai kunci proses enkripsi. Kemudian, *ciphertext* dari *share* hasil enkripsi akan disimpan.

3.3.2 Proses Rekonstruksi *Password*

Pada bagian ini akan dijelaskan tahapan dari proses rekonstruksi *password*. Flowchart dari proses rekonstruksi *password* ditunjukkan pada Gambar 3.2.



Gambar 3.2: Proses Rekonstruksi *Password*

Tahapan-tahapan pada proses yang ditunjukkan oleh Gambar 3.2 adalah tahapan-tahapan untuk merekonstruksi *password*. Tujuan dari proses rekonstruksi *password* adalah mengembalikan *password* yang sudah disimpan. Berikut dijelaskan tahapan-tahapan dalam proses rekonstruksi *password*:

- Tahap Pembuatan *Digest*

Tahap pembuatan *digest* ini sama dengan tahap pembuatan *digest* dalam proses penyimpanan *password*. Perbedaannya adalah pertanyaan keamanan dan nilai *salt* diperoleh dari berkas teks yang disimpan. Sementara itu, jawaban dari pertanyaan keamanan tetap diperoleh dari masukan pengguna.

- Tahap Dekripsi

Pada tahap ini, *ciphertext* yang sudah disimpan akan didekripsi. Namun, sebelum proses dekripsi dilakukan, dibutuhkan beberapa data masukan *digest* yang diperoleh dari tahap pembuatan *digest*. Data masukan *digest* digunakan sebagai kunci dalam proses dekripsi. Proses dekripsi pada tahap ini menggunakan *Data Encryption Standard* (Subbab 2.3).

- Tahap Rekonstruksi *Share*

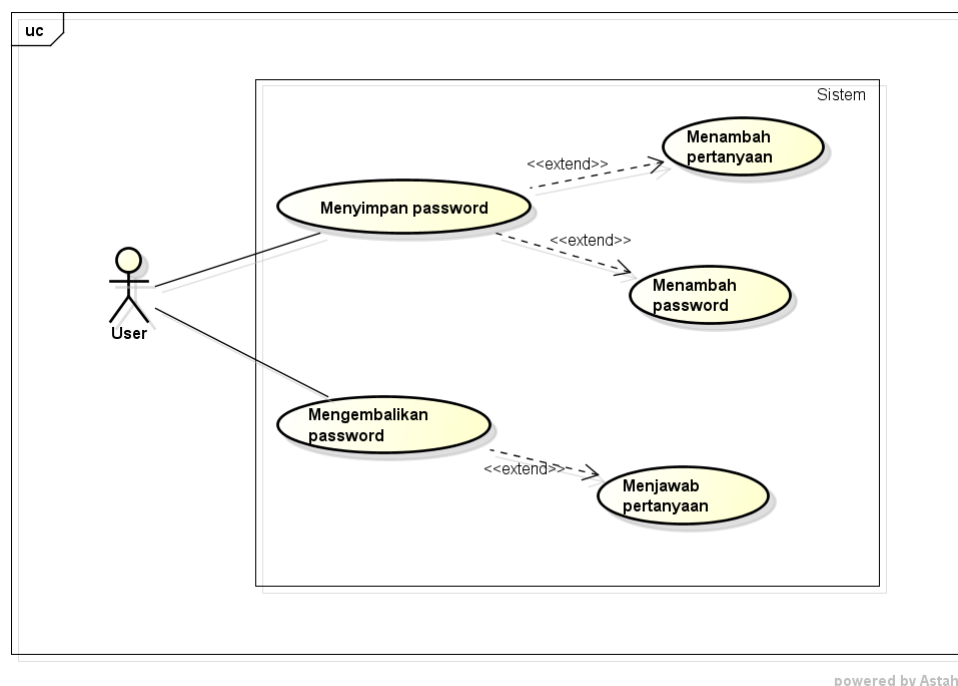
Pada tahap ini, *share* yang dihasilkan dari tahap dekripsi akan direkonstruksi untuk memperoleh *password*. Tidak ada masukan pengguna dalam tahap ini. Metode rekonstruksi *password* yang digunakan pada tahap ini adalah *secret sharing* Shamir.

3.4 Diagram

Pada bagian ini akan dibuat diagram-diagram untuk perangkat lunak yang akan dibangun. Diagram-diagram ini dibuat berdasarkan analisis proses yang sudah dijelaskan pada Subbab 3.3.

3.4.1 Diagram *Use Case*

Perangkat lunak yang dibangun akan memiliki 2 fitur utama, yaitu menyimpan *password* beserta pertanyaan keamanan yang sifatnya personal dan mengembalikan *password*. Saat menyimpan *password*, pengguna akan diminta untuk menambahkan pertanyaan keamanan yang sifatnya personal dan saat mengembalikan *password*, pengguna akan diminta untuk menjawab pertanyaan keamanan yang sudah disimpan saat menyimpan *password*. Gambar 3.3 menunjukkan diagram *use case* dari perangkat lunak.



Gambar 3.3: Diagram *use case* dari perangkat lunak

Adapun skenario-skenario yang pengguna dapat lakukan. Skenario untuk menyimpan *password* ditunjukkan oleh Tabel 3.5. Aktor dari skenario ini adalah pengguna dari sistem. Data masukan dari skenario ini adalah n *password* dan n pertanyaan keamanan.

Tabel 3.5: Skenario Menyimpan *Password*

| | |
|-----------------------|---|
| Nama | Menyimpan <i>Password</i> |
| Aktor | Pengguna |
| Deskripsi | Pengguna menyimpan <i>password</i> |
| Kondisi Awal | Aplikasi sudah dijalankan |
| Kondisi Akhir | <i>share</i> sudah disimpan dalam berkas teks dalam bentuk <i>ciphertext</i> |
| Skenario Utama | <ol style="list-style-type: none"> 1. Pengguna memasukkan <i>password</i> dan pertanyaan keamanan 2. <i>Password</i> dan pertanyaan keamanan diterima oleh sistem |
| Eksepsi | Bila masukan tidak valid, sistem akan memberi peringatan |

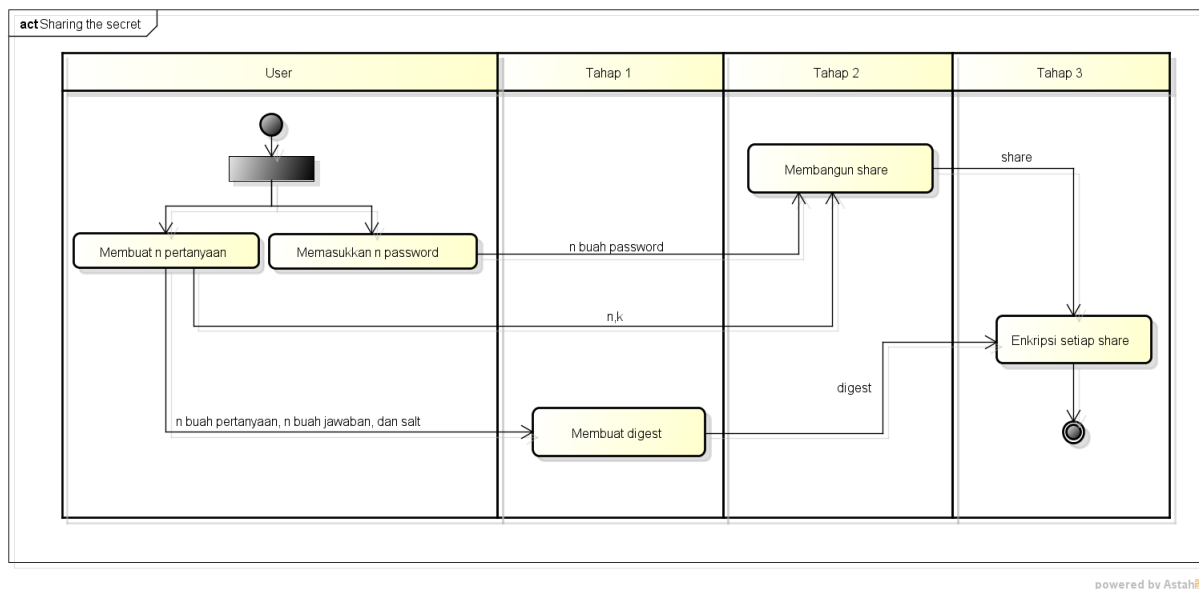
- 1 Skenario untuk mengembalikan *password* ditunjukkan oleh Tabel 3.6. Aktor dari skenario
 2 ini adalah pengguna dari sistem. Data masukan dari skenario ini adalah n jawaban dari
 3 pertanyaan keamanan yang sudah dibuat.

Tabel 3.6: Skenario Menyimpan *Password*

| | |
|-----------------------|---|
| Nama | Mengembalikan <i>Password</i> |
| Aktor | Pengguna |
| Deskripsi | Pengguna mengembalikan <i>password</i> |
| Kondisi Awal | Aplikasi sudah dijalankan serta <i>share</i> , pertanyaan keamanan, dan <i>salt</i> sudah disimpan |
| Kondisi Akhir | Sebanyak n <i>password</i> bisa dikembalikan |
| Skenario Utama | <ol style="list-style-type: none"> 1. Pengguna menjawab setiap pertanyaan keamanan sebagai masukan 2. Masukan diterima oleh sistem 3. Sistem memberikan hasil pemrosesan berupa n <i>password</i> |
| Eksepsi | Bila masukan tidak valid, sistem akan memberi peringatan |

4 3.4.2 Diagram Aktivitas

- 5 Perangkat lunak yang dibangun memiliki 2 proses, yaitu menyimpan *password* dan meng-
 6 embalikan *password*. Urutan aktivitas yang dilakukan perangkat lunak dalam proses me-
 7 nyimpan *password* ditunjukkan pada diagram aktivitas 3.4.

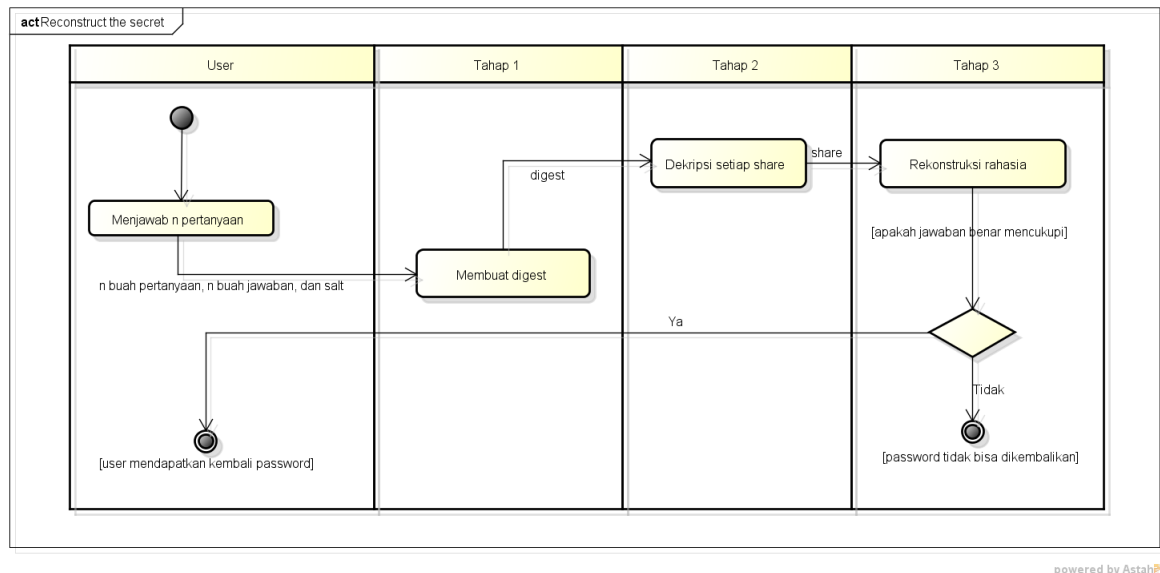


Gambar 3.4: Diagram aktivitas untuk menyimpan *password*

1 Dalam proses menyimpan *password*, pengguna memasukkan *n password*. Kemudian,
 2 setelah memasukkan *n password*, pengguna akan membuat *n pertanyaan keamanan* dan
 3 jawaban masing-masing pertanyaan keamanan. Sistem akan menghitung *k*, yaitu minimal
 4 banyak pertanyaan yang harus dijawab benar oleh pengguna. Setelah itu, sistem akan
 5 memilih secara acak nilai *salt*.

6 Setelah semua masukan yang dibutuhkan ada, sistem akan membuat *digest* dari kon-
 7 katenasi setiap pertanyaan keamanan, jawaban, dan nilai *salt*. Sistem juga akan memba-
 8 ngun *share* dari masukan *password*. Selanjutnya sistem akan mengenkripsi setiap *share* dari
 9 masing-masing *password* dan menggunakan *digest* dari konkatenasi setiap pertanyaan kea-
 10 manan, jawaban, dan nilai *salt* sebagai kunci proses enkripsi. Setelah proses enkripsi, sistem
 11 akan menyimpan seluruh *share* yang sudah dibangun, pertanyaan yang dibuat, *k*, dan nilai
 12 *salt*.

13 Selanjutnya adalah proses mengembalikan *password*. Gambar 3.5 menunjukkan diagram
 14 aktivitas untuk proses mengembalikan *password*.

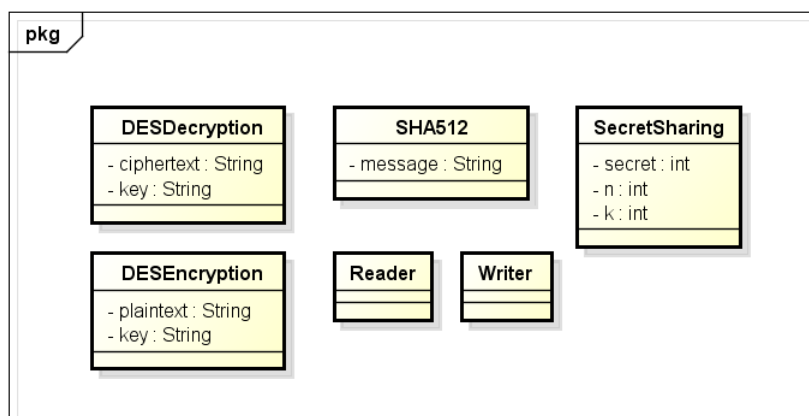
Gambar 3.5: Diagram aktivitas untuk mengembalikan *password*

1 Dalam proses untuk mengembalikan *password*, pengguna akan diminta untuk menja-
 2 wab n pertanyaan keamanan yang disimpan saat proses penyimpanan *password*. Sistem
 3 akan membuat digest dari konkatenasi setiap pertanyaan keamanan, jawaban pertanyaan
 4 keamanan dari masukan pengguna, dan nilai *salt* yang disimpan.

5 Selanjutnya, sistem akan melakukan proses dekripsi setiap *share* yang sudah disimpan
 6 dan menggunakan *digest* sebagai kunci proses dekripsi. Setelah semua *share* didekripsi,
 7 sistem akan merekonstruksi *password* dengan *share* dan nilai k yang disimpan sebagai ma-
 8 sukan. Jika proses rekonstruksi berhasil, maka pengguna dapat mengembalikan *password*.
 9 Sebaliknya, jika proses rekonstruksi gagal, maka *password* tidak bisa dikembalikan.

10 3.4.3 Diagram Kelas

11 Pada bagian ini akan dibuat diagram kelas dari perangkat lunak yang akan dibangun. Kelas-
 12 kelas ini merupakan rancangan kelas yang dibutuhkan untuk membangun perangkat lunak
 13 dan dibuat berdasarkan proses-proses yang sudah dijelaskan pada Subbab 3.3. Rancangan
 14 diagram kelas dari perangkat lunak yang dibangun ditunjukkan oleh Gambar 3.6.



Gambar 3.6: Rancangan Diagram Kelas

Adapun diagram kelas yang ditunjukkan oleh Gambar 3.6 terdiri dari:

1. Kelas *DESEncryption*

Kelas *DESEncryption* merupakan kelas yang berperan untuk melakukan proses enkripsi setiap *share*. Kelas ini mengimplementasikan algoritma *Data Encryption Standard* untuk proses enkripsinya.

2. Kelas *DESDecryption*

Kelas *DESDecryption* merupakan kelas yang berperan untuk melakukan proses dekripsi setiap *share*. Kelas ini mengimplementasikan algoritma *Data Encryption Standard* untuk proses dekripsinya.

3. Kelas *SHA512*

Kelas ini berperan untuk membuat *digest* dari konkatenasi masing-masing pertanyaan, jawaban, dan *salt*. Kelas ini mengimplementasikan algoritma *SHA-512* untuk proses pembuatan *digest*.

4. Kelas *Secret Sharing*

Kelas ini berperan dalam pembangunan *share*. Kelas ini menggunakan metode *secret sharing* Shamir untuk proses pembangunan *share*.

5. Kelas *Writer*

Kelas yang berperan untuk menyimpan setiap keluaran ke dalam berkas teks.

6. Kelas *Reader*

Kelas yang berperan untuk membaca berkas teks dan hasil bacaannya akan digunakan sebagai masukan.

BAB 4

PERANCANGAN

Pada bab ini akan dibahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak akan mencakup diagram kelas rinci, perancangan berorientasi objek, dan perancangan antarmuka.

4.1 Diagram Kelas Rinci

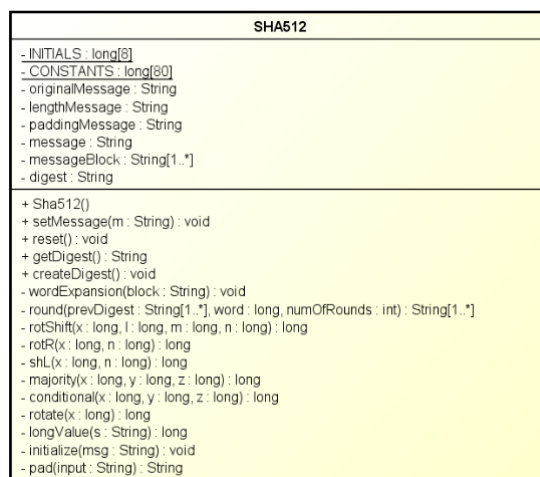
Diagram kelas rinci digunakan sebagai gambaran umum untuk setiap kelas yang ada dalam perangkat lunak yang dibangun serta keterkaitan setiap kelas. Diagram kelas rinci dapat dilihat pada Gambar 4.1. Ada perbedaan antara diagram kelas pada Gambar 4.1 dengan kelas diagram pada Bab 3. Pada diagram kelas rinci ditambahkan beberapa atribut dan fungsi sesuai dengan kebutuhan dari masing-masing kelas.

4.2 Deskripsi Kelas dan Fungsi

Pada bagian ini akan berisi mengenai penjelasan secara rinci masing-masing kelas. Tujuannya adalah menjelaskan peran setiap kelas dalam perangkat lunak yang dibangun.

4.2.1 Kelas *SHA512*

Kelas *SHA512* merupakan kelas yang mengimplementasikan *Secure Hashing Algorithm 512* (SHA-512). Cara kerja algoritma dapat dilihat pada bagian 2.5. Struktur kelas *SHA512* ditunjukkan pada Gambar 4.2.



Gambar 4.2: Kelas SHA512

Adapun atribut dari kelas *SHA512*, yaitu *INITIALS*, *CONSTANTS*, *originalMessage*, *lengthMessage*, *paddingMessage*, *message*, *messageBlock*, dan *digest*. Berikut penjelasan masing-masing atribut tersebut:

1. *long[8] INITIALS*

Atribut yang berguna untuk menyimpan nilai dari konstanta awal.

2. *long[80] INITIALS*

Atribut yang berguna untuk menyimpan konstanta yang digunakan dalam setiap putaran SHA-512.

3. *String originalMessage*

Atribut yang berguna untuk menyimpan *message* yang belum dipadding dalam bentuk *string* biner.

4. *String lengthMessage*

Atribut yang berguna untuk menyimpan informasi mengenai panjang atribut *originalMessage* dalam bentuk *string* biner.

5. *String paddingMessage*

Atribut yang berguna untuk menyimpan blok *padding* dalam bentuk *string* biner.

6. *String message*

Atribut yang berguna untuk menyimpan *message* yang sudah dipadding dalam bentuk *string* biner.

7. *String digest*

Atribut yang berguna untuk menyimpan *digest* dari atribut *message* dalam bentuk *string* biner.

Adapun fungsi yang membangun kelas *SHA512*, yaitu *Sha512*, *setMessage*, *reset*, *getDigest*, *wordExpansion*, *round*, *rotShift*, *rotR*, *shL*, *majority*, *conditional*, *rotate*, *longValue*, *initialize*, dan *pad*. Berikut penjelasan masing-masing fungsi tersebut:

1. *Sha512*

Merupakan konstruktor dari kelas *SHA512*.

2. *void setMessage(String m)*

Fungsi yang berguna untuk menyimpan nilai *string m* ke dalam atribut *originalMessage*.

3. *void reset*

Fungsi yang berguna untuk mengembalikan nilai atribut *originalMessage*, *lengthMessage*, *paddingMessage*, *message*, dan *digest* menjadi *string* kosong dan mengembalikan nilai atribut *messageBlock* menjadi *array string* kosong.

4. *String getDigest*

Fungsi yang berguna untuk mengembalikan nilai atribut *digest* dalam bentuk heksadesimal.

- 1 5. *void createDigest*
- 2 Fungsi yang berperan untuk membuat *digest* dari *message*. Algoritma untuk membuat
- 3 *digest* ini dapat dilihat pada bagian 2.5.
- 4 6. *void wordExpansion(String block)*
- 5 Fungsi yang berperan untuk melakukan proses ekspansi blok *message*.
- 6 7. *String[] round(String[] prevDigest, long word, int numOfRounds)*
- 7 Fungsi yang berperan untuk melakukan 1 putaran dari *SHA512*. Proses yang terjadi
- 8 dalam 1 putaran *SHA512* dapat dilihat pada Bab 2.5.
- 9 8. *long rotShift(long x, long m, long n)*
- 10 Fungsi yang berperan untuk melakukan operasi *RotShift_{l-m-n}(x)* (Subbab 2.5.3).
- 11 9. *long rotR(long x, long n)*
- 12 Fungsi yang berperan untuk melakukan operasi *RotR_i(x)* (Subbab 2.5.3).
- 13 10. *long shL(long x, long n)*
- 14 Fungsi yang berperan untuk melakukan operasi *ShL_i(x)* (Subbab 2.5.3).
- 15 11. *long majority(long x, long y, long z)*
- 16 Fungsi yang berperan untuk melakukan operasi *Majority(x, y, z)* (Subbab 2.5.4).
- 17 12. *long conditional(long x, long y, long z)*
- 18 Fungsi yang berperan untuk melakukan operasi *Conditional(x, y, z)* (Subbab 2.5.4).
- 19 13. *long rotate(long x)*
- 20 Fungsi yang berperan untuk melakukan operasi *Rotate(x)* (Subbab 2.5.4).
- 21 14. *long longValue(String s)*
- 22 Fungsi yang berperan mengubah tipe data *string s* menjadi tipe data *long*.
- 23 15. *void initialize(String msg)*
- 24 Fungsi yang berperan untuk melakukan proses *message padding* dan memecah menjadi
- 25 blok-blok *message*.
- 26 16. *String pad(String input)*
- 27 Fungsi yang berperan untuk menambahkan *padding* angka 0 pada *string* biner *input*.
- 28 Algoritma fungsi ini ditunjukkan pada Algoritma 1.

Algorithm 1 pad

```

1: function PAD(input)
2:   if input.length mod 8! = 0 then
3:     add = 8 - (input.length mod 8)
4:   end if
5:   for i < add do
6:     res = res + "0"
7:   end for
8:   res = res + input
9:   return res
10: end function

```

4.2.2 Kelas *Function*

- Kelas *Function* merupakan kelas yang merepresentasikan sebuah fungsi polinomial $f(x)$. Kelas *Function* ditunjukkan pada Gambar 4.3.

| Function |
|--|
| - function : int[1..*] |
| + Function(func : int[1..*]) + countFunction(x : int) : int |

Gambar 4.3: Kelas *Function*

Adapun atribut dari kelas *Function* adalah *function*. Atribut *function* berguna untuk menyimpan nilai setiap koefisien dari fungsi polinomial $f(x)$ dalam tipe data *array* bilangan bulat.

Sementara itu, fungsi yang dimiliki oleh kelas *Function*, yaitu *Function* dan *countFunction*. Berikut penjelasan masing-masing fungsi:

1. *Function*(int[] func)

Merupakan konstruktor dari kelas *Function* yang menerima masukan *array* bilangan bulat.

2. *int countFunction*(int x)

Menghitung nilai x untuk fungsi polinomial $f(x)$. Algoritma dari fungsi ini ditunjukkan pada Algoritma 2.

Algorithm 2 countFunction

```

1: function COUNTFUNCTION(x)
2:   for  $i < \text{function.length}$  do
3:      $\text{res} = \text{res} + (\text{function}[i] \cdot x^i)$ 
4:   end for
5:   return res
6: end function

```

4.2.3 Kelas *EquationSolver*

Kelas *EquationSolver* adalah kelas yang mengimplementasikan eliminasi Gauss-Jordan (Bagian 2.7). Kelas ini berperan untuk menyelesaikan persamaan linear. Gambar 4.4 menunjukkan struktur dari kelas *EquationSolver*.

| EquationSolver |
|--|
| - equationMatrix : double[1..*][1..*] - solutionMatrix : double[1..*] |
| + EquationSolver(equation : double[1..*][1..*], solution : double[1..*]) + reset() : void + solve() : double[1..*] |

Gambar 4.4: Kelas *EquationSolver*

Adapun atribut dari kelas *EquationSolver*, yaitu *equationMatrix* dan *solutionMatrix*. Berikut penjelasan masing-masing atribut:

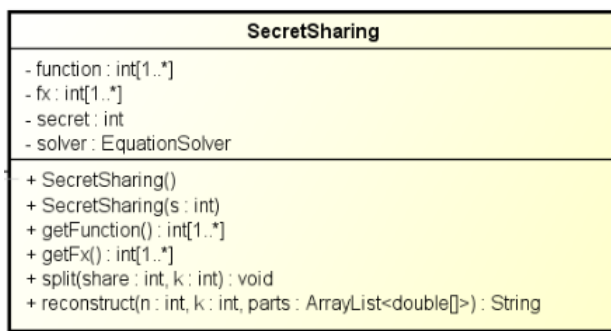
1. *double[][] equationMatrix*
Atribut yang menyimpan bentuk matriks dari persamaan linear.
2. *double[] solutionMatrix*
Atribut yang menyimpan bentuk matriks dari solusi masing-masing persamaan linear.

Sementara itu, fungsi yang dimiliki oleh kelas *EquationSolver*, yaitu *EquationSolver*, *reset*, dan *solve*. Berikut penjelasan masing-masing fungsi:

1. *EquationSolver(double[][] equation, double[] solution)*
Merupakan konstruktor dari kelas *EquationSolver* yang menerima masukan berupa matriks persamaan dan matriks solusi.
2. *void reset*
Mengembalikan nilai atribut *equationMatrix* dan *solutionMatrix* menjadi *array* kosong.
3. *double[] solve*
Mencari solusi dari persamaan linear dan mengembalikan solusi dalam bentuk *array*.

4.2.4 Kelas *SecretSharing*

Kelas *SecretSharing* adalah kelas yang mengimplementasikan skema *threshold(k,n)*. Kelas ini berperan untuk membangun *share* dan juga merekonstruksi rahasia dari *share*. Struktur kelas *SecretSharing* ditunjukkan oleh Gambar ??.



Gambar 4.5: Kelas *DESEncryption*

Kelas *SecretSharing* memiliki beberapa atribut, yaitu *function*, *fx*, *secret*, dan *solver*. Berikut penjelasan masing-masing atribut:

1. *int[] function*
Atribut yang menyimpan nilai setiap koefisien dari fungsi polinomial $f(x)$ dalam tipe data *array* bilangan bulat.
2. *int[] fx*
Atribut yang menyimpan nilai setiap koefisien dari rumus dasar fungsi $f(x)$ dalam tipe data *array* bilangan bulat.

1 3. *int secret*

2 Atribut yang menyimpan rahasia yang akan dibangun *share-share*nya.

3 4. *EquationSolver solver*

4 Atribut dari objek kelas *EquationSolver*.

5 Kelas *SecretSharing* memiliki beberapa fungsi, yaitu *SecretSharing*, *SecretSharing(int s)*,
6 *getFunction*, *getFx*, *split*, dan *reconstruct*. Berikut penjelasan masing-masing fungsi:

7 1. *SecretSharing*

8 Merupakan konstruktor kosong dari kelas *SecretSharing*.

9 2. *SecretSharing(int s)*

10 Merupakan konstruktor dengan masukan *int s* yang akan disimpan ke dalam atribut
11 *secret*.

12 3. *int[] getFunction*

13 Fungsi yang berguna untuk mengembalikan nilai atribut *function*.

14 4. *int[] getFx*

15 Fungsi yang berguna untuk mengembalikan nilai atribut *fx*.

16 5. *split(int share, int k)*

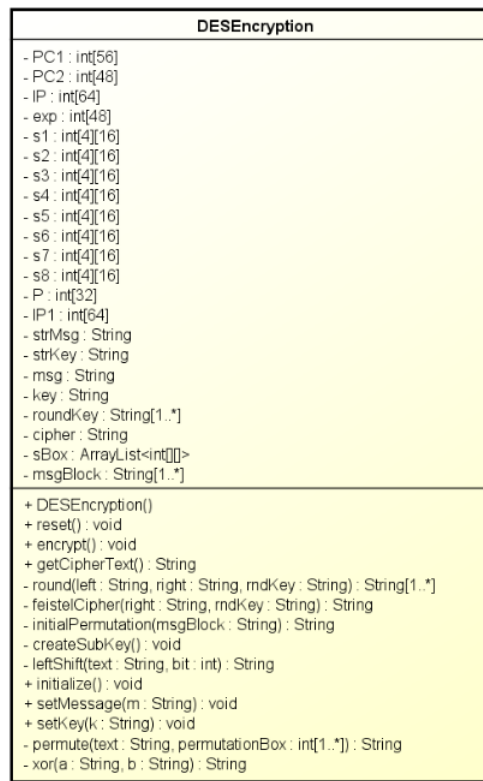
17 Fungsi yang berperan dalam proses pembangunan *share-share*. Cara pembangunan
18 *share* dapat dilihat pada Bab 2.8.

19 6. *String reconstruct(int n, int k, ArrayList<double[]> parts)*

20 Fungsi yang berperan dalam proses rekonstruksi rahasia dari *share-share* yang dimiliki.
21 *Share-share* yang dimiliki ini disimpan dalam masukan *ArrayList<double[]> parts*.
22 Proses rekonstruksi rahasia dapat dilihat pada Bab 2.8.

23 4.2.5 Kelas *DESEncryption*

24 Kelas *DESEncryption* adalah kelas yang mengimplementasikan algoritma *Data Encryption*
25 *Standard* (Bab 2.3). Kelas ini berperan untuk melakukan proses enkripsi menggunakan *Data*
26 *Encryption Standard*. Struktur kelas *DESEncryption* ditunjukkan oleh Gambar 4.6.

Gambar 4.6: Kelas *DESEncryption*

1 Kelas DESEncryption memiliki beberapa atribut, yaitu *PC1*, *PC2*, *IP*, *exp*, *s1*, *s2*, *s3*,
 2 *s4*, *s5*, *s6*, *s7*, *s8*, *P*, *IP1*, *strMsg*, *strKey*, *msg*, *key*, *roundKey*, *cipher*, *sBox*, dan *msgBlock*.
 3 Berikut penjelasan masing-masing atribut:

4 1. *int[56] PC1*

5 Atribut yang menyimpan matriks *parity drop* yang digunakan dalam proses pembang-
 6 kitan kunci putaran.

7 2. *int[48] PC2*

8 Atribut yang menyimpan matriks permutasi *P-box* yang digunakan dalam proses pem-
 9 bangkitan kunci putaran.

10 3. *int[64] IP*

11 Atribut yang menyimpan matriks permutasi awal.

12 4. *int[48] exp*

13 Atribut yang menyimpan matriks ekspansi *P-box*.

14 5. *int[4][16] s1*

15 Atribut yang menyimpan matriks *s-box* ke-1.

16 6. *int[4][16] s2*

17 Atribut yang menyimpan matriks *s-box* ke-2.

18 7. *int[4][16] s3*

19 Atribut yang menyimpan matriks *s-box* ke-3.

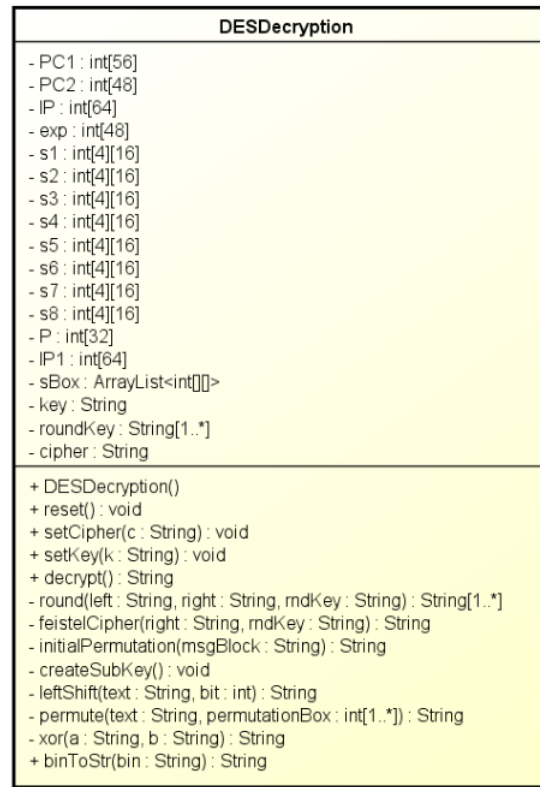
- 1 8. *int[4][16] s4*
- 2 Atribut yang menyimpan matriks *s-box* ke-4.
- 3 9. *int[4][16] s5*
- 4 Atribut yang menyimpan matriks *s-box* ke-5.
- 5 10. *int[4][16] s6*
- 6 Atribut yang menyimpan matriks *s-box* ke-6.
- 7 11. *int[4][16] s7*
- 8 Atribut yang menyimpan matriks *s-box* ke-7.
- 9 12. *int[4][16] s8*
- 10 Atribut yang menyimpan matriks *s-box* ke-8.
- 11 13. *int[32] P*
- 12 Atribut yang menyimpan matriks permutasi pada tahap terakhir fungsi DES.
- 13 14. *int[64] IP1*
- 14 Atribut yang menyimpan matriks permutasi akhir.
- 15 15. *String strMsg*
- 16 Atribut yang menyimpan *plaintext* dalam bentuk *string*.
- 17 16. *String strKey*
- 18 Atribut yang menyimpan kunci dalam bentuk *string*.
- 19 17. *String msg*
- 20 Atribut yang menyimpan *plaintext* dalam bentuk *string* biner.
- 21 18. *String key*
- 22 Atribut yang menyimpan kunci dalam bentuk *string* biner.
- 23 19. *String[] roundKey*
- 24 Atribut yang menyimpan kunci untuk setiap putaran dalam *array string* biner.
- 25 20. *String cipher*
- 26 Atribut yang menyimpan *ciphertext* hasil enkripsi dalam bentuk *string* biner.
- 27 21. *ArrayList<int[][]> sBox*
- 28 Atribut yang menyimpan seluruh matriks *s-box*.
- 29 22. *String[] msgBlock*
- 30 Atribut yang menyimpan blok-blok *plaintext* dalam bentuk *array string* biner. Setiap
- 31 elemen *array* berisi *string* biner dengan panjang 64-bit.

32 Adapun fungsi-fungsi yang dimiliki oleh kelas *DESEncryption*, yaitu *DESEncryption*,
 33 *reset*, *encrypt*, *getCipherText*, *round*, *feistelCipher*, *initialPermutation*, *createSubKey*, *lef-*
 34 *tShift*, *initialize*, *setMessage*, *setKey*, *permute*, dan *xor*. Berikut penjelasan masing-masing
 35 fungsi:

- 1 1. *DESEncryption*
2 Merupakan konstruktor dari kelas *DESEncryption*.
- 3 2. *void reset*
4 Fungsi yang berperan untuk mengembalikan atribut *strMsg*, *strKey*, *msg*, *key*, dan
5 *cipher* menjadi *string* kosong. Fungsi ini juga mengembalikan atribut *roundKey* dan
6 *msgBlock* menjadi *array string* kosong.
- 7 3. *void encrypt*
8 Fungsi yang berperan untuk melakukan proses enkripsi DES.
- 9 4. *String getCipherText*
10 Fungsi yang mengembalikan nilai atribut *cipher* dalam bentuk *string* heksadesimal.
- 11 5. *String[] round(String left, String right, String rndKey)*
12 Fungsi yang berperan untuk melakukan 1 putaran dari DES. Fungsi ini mengembalikan
13 *array string* yang berguna untuk blok masukan putaran berikutnya.
- 14 6. *String feistelCipher(String right, String rndKey)*
15 Fungsi yang berperan untuk melakukan proses jaringan Feistel kepada blok bagian
16 kanan dari *plaintext*.
- 17 7. *String initialPermutation(String msgBlock)*
18 Fungsi ini berperan untuk melakukan proses permutasi awal.
- 19 8. *void createSubKey*
20 Fungsi yang berperan untuk membangkitkan kunci putaran.
- 21 9. *String leftShift(String text, int bit)*
22 Fungsi yang berperan untuk melakukan *shift left* dari masukan *string text* sebanyak
23 masukan *int bit*.
- 24 10. *void initialize*
25 Fungsi yang berperan untuk melakukan *padding* terhadap *strMsg* dan memecah *strMsg*
26 menjadi blok-blok *string*. Blok-blok *string* ini akan disimpan pada atribut *msgBlock*.
- 27 11. *void setMessage(String m)*
28 Fungsi untuk menyimpan *string m* ke dalam atribut *strMsg*.
- 29 12. *void setKey(String k)*
30 Fungsi untuk menyimpan *string k* ke dalam atribut *strKey*.
- 31 13. *String permute(String text, int[] permutationBox)*
32 Fungsi yang berperan untuk melakukan proses permutasi.
- 33 14. *String xor(String a, String b)*
34 Fungsi yang berperan untuk melakukan operasi XOR.

4.2.6 Kelas *DESDecryption*

- Kelas *DESDecryption* adalah kelas yang berperan untuk melakukan proses dekripsi menggunakan DES. Struktur kelas *DESDecryption* ditunjukkan pada Gambar 4.7.



Gambar 4.7: Kelas *DESDecryption*

- Kelas *DESDecryption* memiliki beberapa atribut, yaitu *PC1*, *PC2*, *IP*, *exp*, *s1*, *s2*, *s3*, *s4*, *s5*, *s6*, *s7*, *s8*, *P*, *IP1*, *sBox*, *key*, *roundKey*, dan *cipher*. Berikut penjelasan masing-masing atribut:

1. *int[56] PC1*

Atribut yang menyimpan matriks *parity drop* yang digunakan dalam proses pembangkitan kunci putaran.

2. *int[48] PC2*

Atribut yang menyimpan matriks permutasi *P-box* yang digunakan dalam proses pembangkitan kunci putaran.

3. *int[64] IP*

Atribut yang menyimpan matriks permutasi awal.

4. *int[48] exp*

Atribut yang menyimpan matriks ekspansi *P-box*.

5. *int[4][16] s1*

Atribut yang menyimpan matriks *s-box* ke-1.

6. *int[4][16] s2*

Atribut yang menyimpan matriks *s-box* ke-2.

- 1 7. *int[4][16] s3*
- 2 Atribut yang menyimpan matriks *s-box* ke-3.
- 3 8. *int[4][16] s4*
- 4 Atribut yang menyimpan matriks *s-box* ke-4.
- 5 9. *int[4][16] s5*
- 6 Atribut yang menyimpan matriks *s-box* ke-5.
- 7 10. *int[4][16] s6*
- 8 Atribut yang menyimpan matriks *s-box* ke-6.
- 9 11. *int[4][16] s7*
- 10 Atribut yang menyimpan matriks *s-box* ke-7.
- 11 12. *int[4][16] s8*
- 12 Atribut yang menyimpan matriks *s-box* ke-8.
- 13 13. *int[32] P*
- 14 Atribut yang menyimpan matriks permutasi pada tahap terakhir fungsi DES.
- 15 14. *int[64] IP1*
- 16 Atribut yang menyimpan matriks permutasi akhir.
- 17 15. *ArrayList<int[][]> sBox*
- 18 Atribut yang menyimpan seluruh matriks *s-box*.
- 19 16. *String key*
- 20 Atribut yang menyimpan kunci dalam bentuk *string* biner.
- 21 17. *String[] roundKey*
- 22 Atribut yang menyimpan kunci untuk setiap putaran dalam *array string* biner.
- 23 18. *String cipher*
- 24 Atribut yang menyimpan *ciphertext* yang akan didekripsi dalam bentuk *string* biner.
- 25 Adapun fungsi-fungsi yang dimiliki kelas *DESDecryption*, yaitu *DESDecryption*, *reset*,
- 26 *setCipher*, *setKey*, *decrypt*, *round*, *feistelCipher*, *initialPermutation*, *createSubKey*, *leftShift*,
- 27 *permute*, *xor*, dan *binToStr*. Berikut penjelasan masing-masing fungsi:
- 28 1. *DESDecryption*
- 29 Merupakan konstruktor dari kelas *DESDecryption*.
- 30 2. *void reset*
- 31 Fungsi yang berperan untuk mengembalikan atribut *key* dan *cipher* menjadi *string*
- 32 kosong. Fungsi ini juga mengembalikan atribut *roundKey* menjadi *array string* kosong.
- 33 3. *void setCipher(String c)*
- 34 Fungsi untuk menyimpan *string c* pada atribut *cipher* dalam bentuk *string* biner.
- 35 4. *void setKey(String k)*
- 36 Fungsi untuk menyimpan *string k* pada atribut *key* dalam bentuk *string* biner.

5. *String decrypt*

Fungsi yang berperan untuk melakukan proses dekripsi menggunakan DES.

6. *String[] round(String left, String right, String rndKey)*

Fungsi yang berperan untuk melakukan 1 putaran dari DES. Fungsi ini mengembalikan *array string* yang berguna untuk blok masukan putaran berikutnya.

7. *String feistelCipher(String right, String rndKey)*

Fungsi yang berperan untuk melakukan proses jaringan Feistel kepada blok bagian kanan dari *plaintext*.

8. *String initialPermutation(String msgBlock)*

Fungsi ini berperan untuk melakukan proses permutasi awal.

9. *void createSubKey*

Fungsi yang berperan untuk membangkitkan kunci putaran.

10. *String leftShift(String text, int bit)*

Fungsi yang berperan untuk melakukan *shift left* dari masukan *string text* sebanyak masukan *int bit*.

11. *String permute(String text, int[] permutationBox)*

Fungsi yang berperan untuk melakukan proses permutasi.

12. *String xor(String a, String b)*

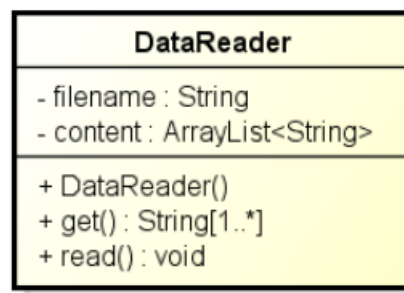
Fungsi yang berperan untuk melakukan operasi XOR.

13. *String binToStr(String bin)*

Fungsi yang berperan untuk mengubah *string* biner menjadi *string*.

4.2.7 Kelas *DataReader*

Kelas *DataReader* merupakan kelas yang berperan untuk membaca berkas teks. Kelas *DataReader* ditunjukkan pada Gambar 4.8.



Gambar 4.8: Kelas *DataReader*

Kelas ini memiliki 2 atribut, yaitu *filename* dan *content*. Berikut penjelasan masing-masing atribut:

1. *String filename*

Atribut yang menyimpan nama dari berkas teks yang dibaca.

1 2. *ArrayList<String> content*

2 Atribut yang menyimpan isi dari berkas teks yang dibaca.

3 Adapun kelas ini memiliki 3 fungsi, yaitu *DataReader*, *get*, dan *read*. Berikut penjelasan
4 masing-masing fungsi:

5 1. *DataReader*

6 Merupakan konstruktor dari kelas *DataReader*.

7 2. *String[] get*

8 Fungsi yang berguna untuk mengembalikan atribut *content*.

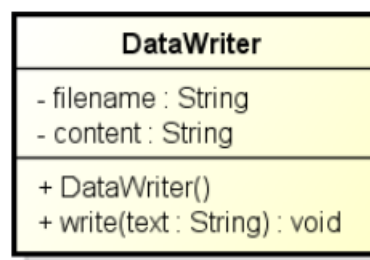
9 3. *void read*

10 Fungsi yang berperan membaca berkas teks.

11 4.2.8 Kelas *DataWriter*

12 Kelas *DataWriter* adalah kelas yang berperan untuk menulis keluaran ke dalam berkas teks.

13 Kelas *DataWriter* ditunjukkan pada Gambar 4.9.



Gambar 4.9: Kelas *DataWriter*

14 Kelas ini memiliki 2 atribut, yaitu *filename* dan *content*. Berikut penjelasan masing-
15 masing atribut:

16 1. *String filename*

17 Atribut yang menyimpan nama dari berkas teks yang akan ditulis.

18 2. *String content*

19 Atribut yang menyimpan isi dari berkas teks yang akan ditulis.

20 Adapun kelas ini memiliki 2 fungsi, yaitu *DataWriter* dan *write*. Berikut penjelasan
21 masing-masing fungsi:

22 1. *DataWriter*

23 Merupakan konstruktor dari kelas *DataWriter*.

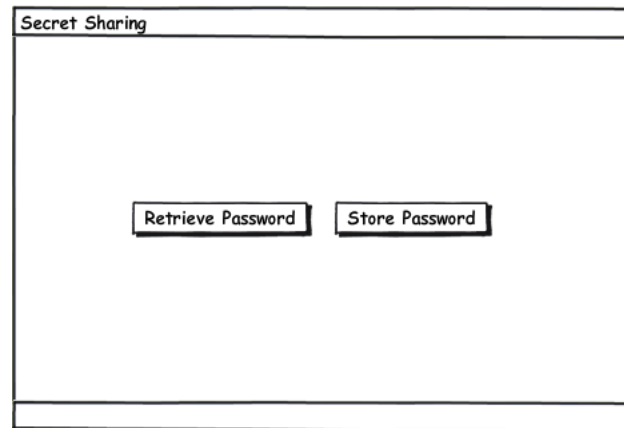
24 2. *write(String text)*

25 Fungsi yang berperan untuk menulis isi dari berkas teks.

4.3 Perancangan Antarmuka

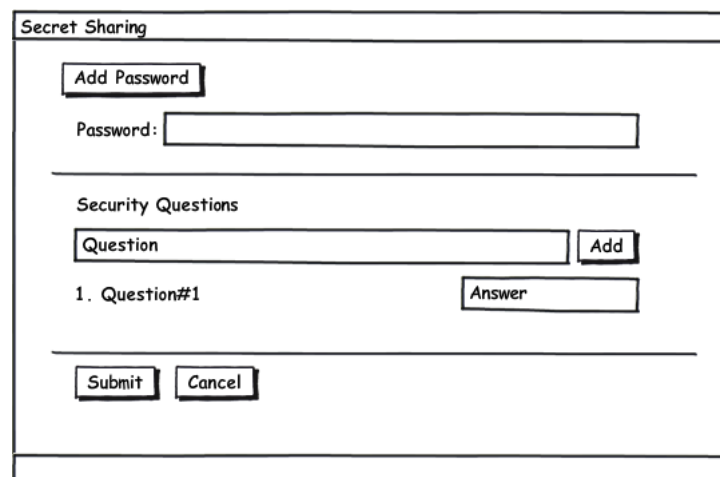
Perangkat lunak yang dikembangkan akan memiliki 3 tampilan utama, tampilan untuk menyimpan *password*, tampilan untuk mengembalikan *password*, dan tampilan untuk memilih menyimpan *password* atau mengembalikan *password*.

Gambar 4.10 menunjukkan tampilan awal yang akan dimunculkan pertama kali untuk memilih menyimpan *password* atau mengembalikan *password*.



Gambar 4.10: Perancangan Tampilan Awal

Tampilan utama ini cukup sederhana. Dalam tampilan utama pada Gambar 4.10, hanya terdapat 2 pilihan, yaitu *store password* untuk menyimpan *password* dan *retrieve password* untuk mengembalikan *password*. Selanjutnya, jika pengguna memilih *store password*, maka akan ditampilkan halaman *store password*.

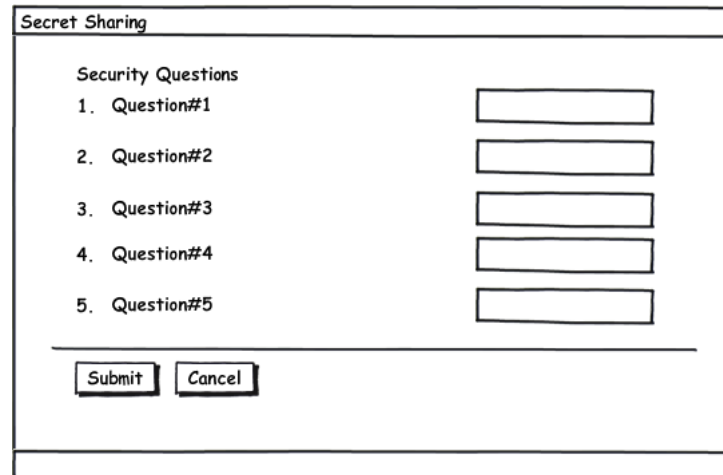


Gambar 4.11: Perancangan Tampilan Menyimpan *Password*

Pada tampilan menyimpan *password* di Gambar 4.11, tombol "Add Password" berfungsi untuk menambah *text box password*, pada bagian ini pengguna bisa mengisi *password* yang akan disimpan. Bagian "Security Questions" berisi pertanyaan keamanan yang dibuat oleh pengguna. Setelah pengguna mengisi pertanyaan personal pada *text box* di bagian "Security Questions" dan menekan tombol "Add", akan muncul pertanyaan yang sudah dibuat, kemudian pengguna harus mengisi jawaban dari pertanyaan keamanan yang sudah dibuat.

1 Setelah mengisi seluruh pertanyaan keamanan, pengguna bisa menyimpan *password* de-
2 ngan menekan tombol "*Submit*". Tombol "*Cancel*" berfungsi untuk kembali ke tampilan
3 awal. Setelah tombol "*Submit*" ditekan, maka *password* sudah disimpan dan akan kembali
4 ditampilkan tampilan awal.

5 Berikutnya adalah tampilan untuk mengembalikan *password*. Gambar 4.12 menunjukkan
6 tampilan untuk mengembalikan *password*.

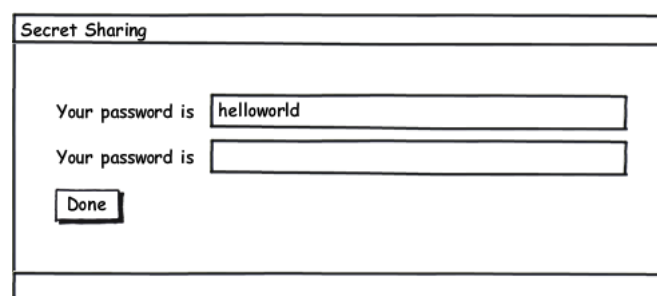


The image shows a window titled "Secret Sharing". Inside, under the heading "Security Questions", there are five numbered questions: "1. Question#1", "2. Question#2", "3. Question#3", "4. Question#4", and "5. Question#5". To the right of each question is a rectangular input field. At the bottom of the window, there are two buttons: "Submit" and "Cancel".

Gambar 4.12: Perancangan Tampilan Mengembalikan *Password*

7 Pada bagian untuk mengembalikan *password*, tampilannya cukup sederhana dan penggu-
8 na hanya cukup memasukkan setiap jawaban dari pertanyaan keamanan yang sudah dibuat
9 sebelumnya di bagian penyimpanan *password*. Pada bagian ini, pengguna bebas untuk
10 memilih mengisi setiap pertanyaan atau tidak menjawab pertanyaan keamanan. Setelah se-
11 luruh pertanyaan sudah dijawab, pengguna dapat menekan tombol "*Submit*" yang kemudian
12 akan menunjukkan *password* pengguna.

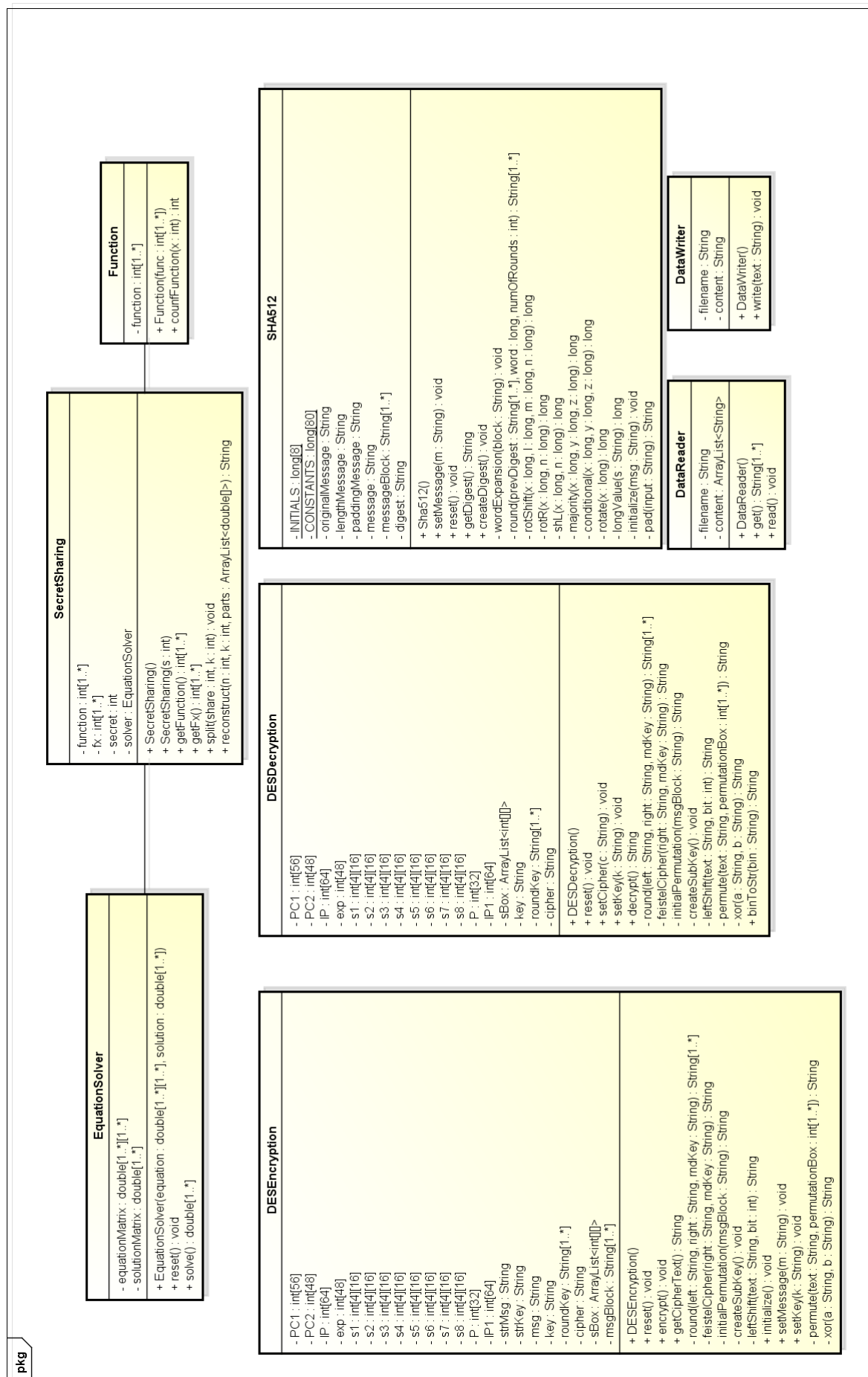
13 Gambar 4.13 menunjukkan tampilan sesudah pengguna menekan tombol "*Submit*" pada
14 bagian di Gambar 4.12.



The image shows the same "Secret Sharing" window after the user has submitted their answers. The text "Your password is" is now followed by the password "helloworld" in the first input field. Below it, there is another empty input field with the label "Your password is" to its left. At the bottom, there is a single button labeled "Done".

Gambar 4.13: Perancangan Tampilan Mengembalikan *Password*

15 Jika banyak pertanyaan keamanan yang dijawab benar oleh pengguna sesuai dengan
16 minimal banyak pertanyaan keamanan yang dijawab benar maka pengguna bisa melihat
17 *password* yang sudah disimpan. Tapi, jika banyak pertanyaan keamanan yang dijawab benar
18 oleh pengguna kurang dari minimal banyak pertanyaan keamanan yang harus dijawab benar
19 maka pengguna tidak bisa melihat *password* yang sudah disimpan.



Gambar 4.1: Diagram Kelas Rinci

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan berisi mengenai implementasi perangkat lunak dan pengujian perangkat lunak yang dibangun.

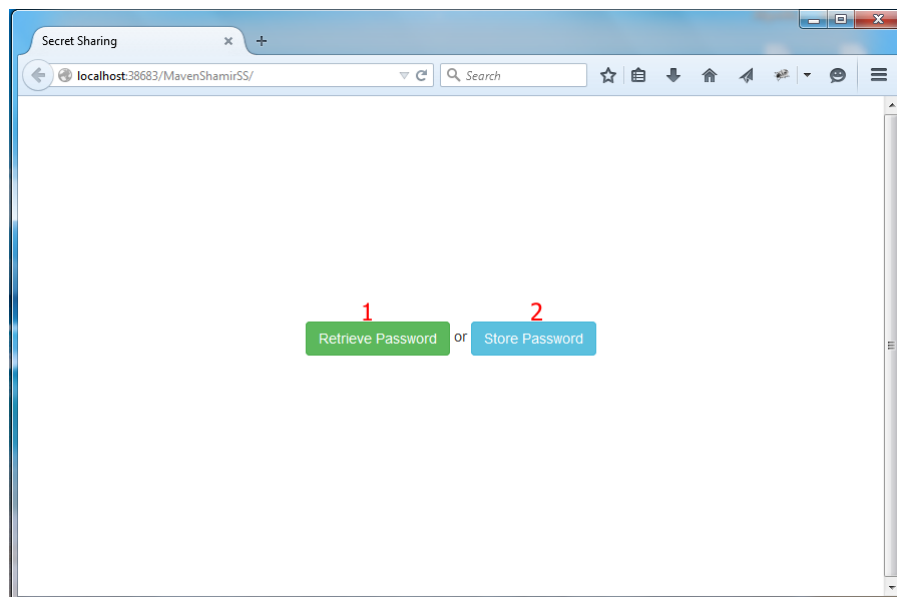
5.1 Implementasi Perangkat Lunak

Pada bagian ini akan dibahas mengenai tampilan antarmuka perangkat lunak yang sudah dibangun.

5.1.1 Tampilan Antarmuka Perangkat Lunak

Tampilan antarmuka awal perangkat lunak dapat dilihat pada Gambar 5.1 dengan keterangan bagian-bagian sebagai berikut.

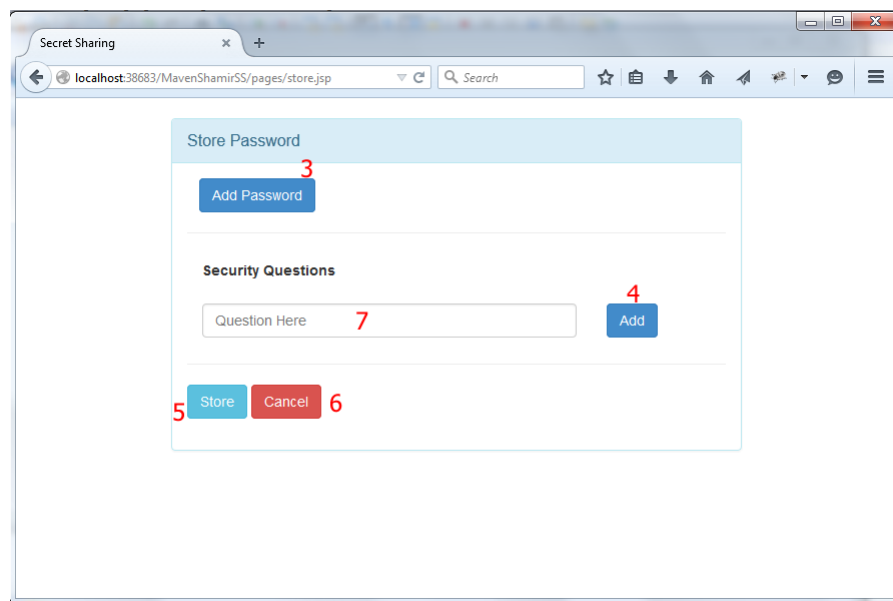
- Bagian nomor 1 merupakan tombol untuk mengembalikan *password*.
- Bagian nomor 2 merupakan tombol untuk menyimpan *password*.



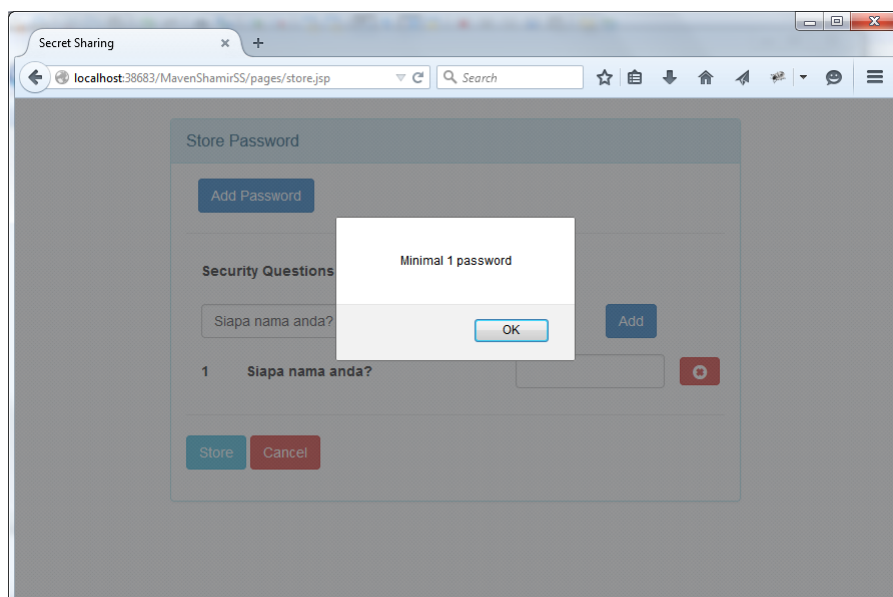
Gambar 5.1: Tampilan antarmuka awal

Setelah tombol "*Store Password*" ditekan, tampilan antarmuka perangkat lunak akan terlihat seperti pada Gambar 5.2 dengan keterangan sebagai berikut.

- 1 • Bagian nomor 3 merupakan tombol untuk menambah *password*. Pengguna minimal
- 2 harus menambahkan 1 *password*, jika tidak maka akan muncul notifikasi seperti pada
- 3 Gambar 5.3.
- 4 • Bagian nomor 4 merupakan tombol untuk menambah pertanyaan keamanan.
- 5 • Bagian nomor 5 merupakan tombol untuk melanjutkan menyimpan *password*.
- 6 • Bagian nomor 6 merupakan tombol untuk kembali ke tampilan antarmuka awal.
- 7 • Bagian nomor 7 merupakan teks masukan untuk pertanyaan keamanan yang hendak
- 8 ditambahkan.

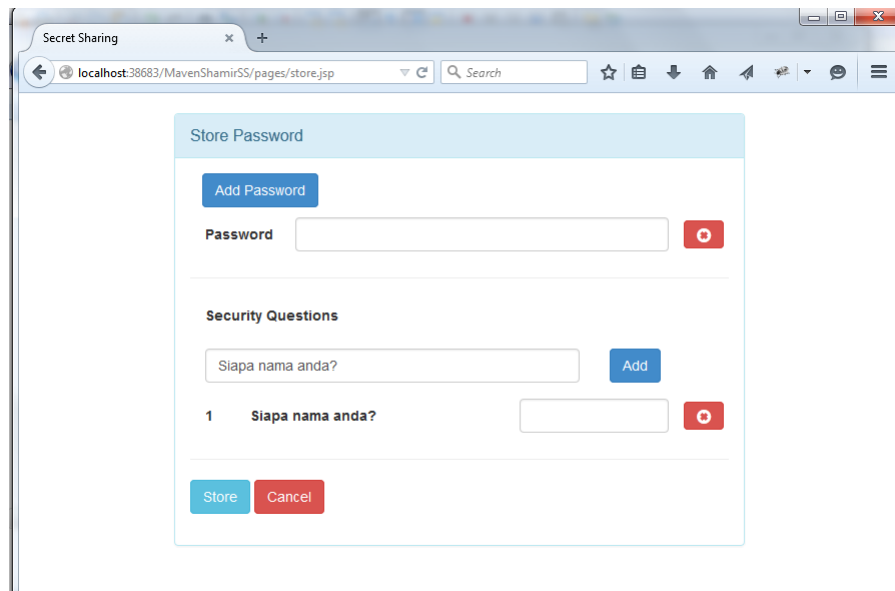


Gambar 5.2: Tampilan antarmuka untuk menyimpan *password*



Gambar 5.3: Tampilan notifikasi pada antarmuka jika *password* kurang

1 Setelah tombol "*Add Password*" ditekan, maka tampilan antarmuka akan menambahk-
2 an masukkan teks untuk memasukkan *password* yang hendak disimpan. Setelah tombol
3 "*Add*" ditekan, maka tampilan antarmuka akan menambahkan teks masukkan untuk jawab-
4 an dari pertanyaan keamanan yang sudah diisi di Bagian no 7. Tampilan yang ditunjukkan
5 perangkat lunak dapat dilihat pada Gambar 5.4.



Gambar 5.4: Tampilan antarmuka menambah *password*

6 Setelah tombol "*Store*" ditekan, maka tampilan antarmuka perangkat lunak akan kembali
7 ke tampilan antarmuka awal. *Password* sudah berhasil disimpan. Kemudian, setelah tombol
8 "*Retrieve Password*" ditekan, maka tampilan perangkat lunak akan terlihat seperti pada
9 Gambar 5.5 dengan keterangan sebagai berikut.

- 10 • Bagian nomor 8 merupakan bagian dari pertanyaan keamanan yang harus dijawab
11 oleh pengguna.
- 12 • Bagian nomor 9 merupakan bagian dari jawaban setiap pertanyaan keamanan yang
13 harus dijawab.
- 14 • Bagian nomor 10 merupakan tombol untuk mengembalikan *password*.
- 15 • Bagian nomor 11 merupakan tombol untuk kembali ke tampilan antarmuka awal.

Secret Sharing

localhost:38683/MavenShamirSS/pages/retrieve.jsp

Retrieve Password

Security Questions

1. Siapa nama anda?
2. Kapan anda lahir?
3. Dimana anda bersekolah?
4. Siapa guru kelas 5 SD anda?
5. Apa nama binatang peliharaan anda?

Submit Cancel

Gambar 5.5: Tampilan antarmuka untuk mengembalikan *password*

- 1 Setelah tombol "Submit" pada Gambar 5.5 ditekan, perangkat lunak akan memroses
- 2 setiap pertanyaan dan jawaban. Jika banyak jawaban benar dari pertanyaan keamanan
- 3 yang dijawab oleh pengguna sesuai dengan minimal banyak pertanyaan keamanan yang
- 4 dijawab benar yang sudah ditentukan sebelumnya, maka tampilan perangkat lunak akan
- 5 terlihat seperti pada Gambar 5.6.

Secret Sharing

localhost:38683/MavenShamirSS/pages/retrieve_process.jsp

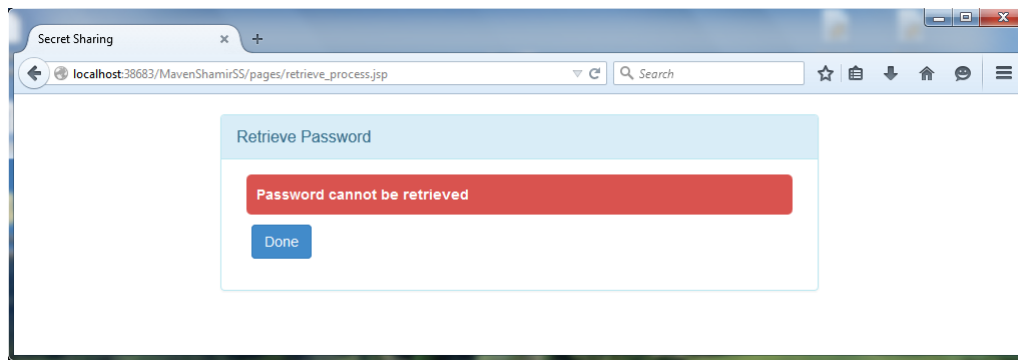
Retrieve Password

Your password is password

Done

Gambar 5.6: Tampilan antarmuka untuk mengembalikan *password*

- 6 Sedangkan, jika pertanyaan keamanan yang dijawab benar kurang dari minimal per-
- 7 tanyaan keamanan yang harus dijawab benar, maka tampilan perangkat lunak akan me-
- 8 nunjukkan notifikasi bahwa *password* tidak bisa dikembalikan. Gambar 5.7 menunjukkan
- 9 tampilan antarmuka perangkat lunak dengan notifikasi *password* tidak bisa dikembalikan.



Gambar 5.7: Tampilan antarmuka untuk mengembalikan *password*

5.2 Pengujian Perangkat Lunak

Pada bagian ini akan berisi tentang metode pengujian, hasil pengujian, analisis pengujian, dan kesimpulan dari pengujian perangkat lunak yang sudah dibangun.

5.2.1 Metode Pengujian

Pengujian terhadap perangkat lunak yang sudah dibangun akan dibagi menjadi 2 bagian, yaitu pengujian fungsional dan pengujian survei. Pada bagian ini akan dijelaskan masing-masing dari pengujian dan kasus yang akan digunakan dalam masing-masing pengujian yang dilakukan.

Pengujian Fungsional

Pengujian fungsional bertujuan untuk menguji apakah perangkat lunak dapat berfungsi sesuai dengan harapan. Dalam penelitian ini, perangkat lunak yang dibangun diharapkan dapat menyimpan *password* dalam bentuk *share-share* dan bisa mengembalikan banyak *password* sekaligus dengan menjawab beberapa pertanyaan keamanan.

Kasus yang digunakan dalam pengujian fungsional terdiri dari 2 kasus. Kasus pertama adalah kasus dimana jika sebagian besar pertanyaan keamanan dapat dijawab dengan benar maka *password* akan bisa dikembalikan. Kasus kedua adalah kasus dimana jika pertanyaan keamanan yang dijawab benar tidak mencapai minimal pertanyaan keamanan yang dijawab benar sehingga *password* tidak bisa dikembalikan.

Pengujian Survei

Pengujian survei bertujuan untuk menguji kualitas dari pertanyaan keamanan yang dibuat saat proses menyimpan *password*. Pengujian survei akan menguji pertanyaan keamanan dibuat berpengaruh pada mudah atau tidaknya *password* bisa dikembalikan. Setiap pertanyaan ini akan dikelompokkan menjadi beberapa topik kasus yang sesuai dengan jenisnya.

Kasus yang digunakan dalam terdiri dari 4 kasus yang masing-masing terbagi atas topiknya masing-masing. Berikut penjelasan masing-masing topik kasus.

1. Topik 1

Pertanyaan keamanan yang kemungkinan jawabannya hanya 2, yaitu Ya atau Tidak.

Selain itu, dalam topik ini digunakan pertanyaan keamanan yang jawabannya bisa dicari di *internet* atau media sosial.

2. Topik 2

Pertanyaan keamanan yang sebagian besar jawabannya mengenai angka, seperti tanggal, bulan, tahun, dan sebagainya.

3. Topik 3

Pertanyaan keamanan yang jawabannya mengenai hal-hal personal.

4. Topik 4

Topik 4 akan berisi gabungan dari topik 1, topik 2, dan topik 3.

5.2.2 Hasil Pengujian Fungsional

Sebelum dibahas hasil pengujian terhadap kasus-kasus yang sudah dijelaskan pada bagian metode pengujian fungsional, langkah pertama yang harus dilakukan adalah menyimpan *password*. Untuk pengujian fungsional ini, n yang dipilih adalah $n = 5$ dan k yang dipilih adalah $k = 4$. Karena itu, ada 5 *password* yang akan disimpan dan ada 5 pertanyaan keamanan yang dibuat untuk masing-masing *password*. Langkah ini ditunjukkan pada Gambar 5.8.

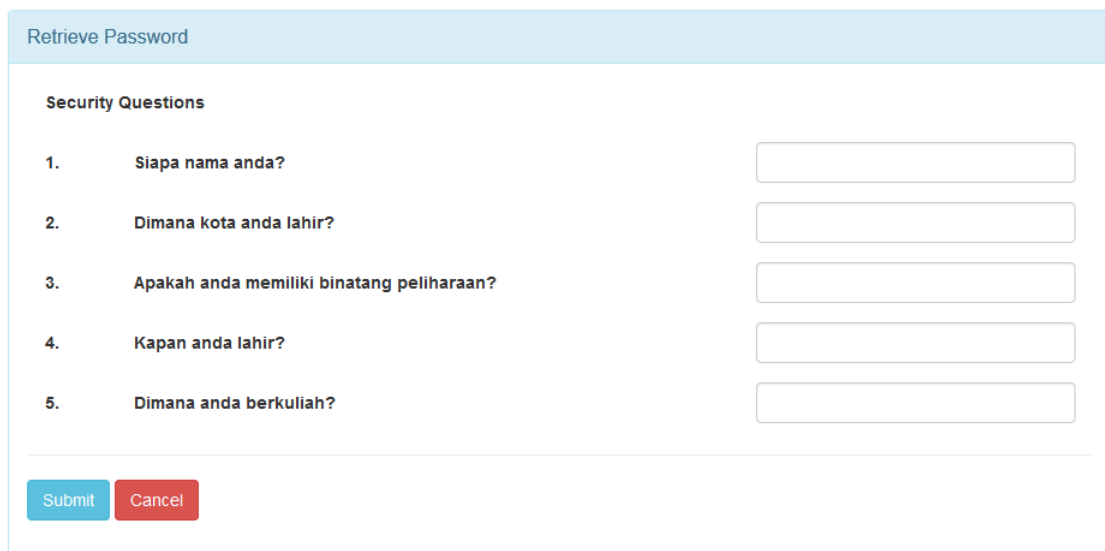
Gambar 5.8: Langkah menyimpan *password*

Masukan password akan pada Gambar 5.8 ditunjukkan hanya sebagai bagian dari pengujian saja. Selanjutnya, setelah tombol "Store" ditekan, maka password akan disimpan dan

1 tampilan antarmuka perangkat lunak akan kembali ke tampilan antarmuka awal. Tampilan
2 antarmuka awal ditunjukkan pada Gambar 5.1.

3 Sesudah menyimpan password, langkah selanjutnya adalah melakukan pengujian terha-
4 dap kasus-kasus yang sudah dibahas pada bagian sebelumnya. Kasus pertama dari pengujian
5 fungsional adalah kasus pertanyaan yang dijawab dengan benar sebanyak nilai k atau lebih.
6 Hasil yang diharapkan dari kasus pertama adalah password bisa dikembalikan karena banyak
7 pertanyaan yang dijawab dengan benar sebanyak nilai k atau lebih.

8 Langkah pertama yang perlu dilakukan dalam pengujian terhadap kasus-kasus adalah
9 menjawab pertanyaan keamanan yang sudah disimpan. Untuk bisa menjawab pertanyaan
10 keamanan yang sudah disimpan, tombol "*Retrieve Password*" pada tampilan antarmuka
11 awal harus ditekan. Setelah tombol tersebut ditekan, perangkat lunak akan menampilkan
12 tampilan pada Gambar 5.9. Setiap pertanyaan pada Gambar 5.9 dijawab dengan mengisi
13 masukan teks yang ada di samping masing-masing pertanyaan keamanan.



The image shows a web form titled "Retrieve Password". Below the title is a section labeled "Security Questions". It contains five numbered questions, each followed by a text input field:

1. Siapa nama anda?
2. Dimana kota anda lahir?
3. Apakah anda memiliki binatang peliharaan?
4. Kapan anda lahir?
5. Dimana anda berkuliah?

At the bottom of the form are two buttons: "Submit" (blue) and "Cancel" (red).

Gambar 5.9: Tampilan Menjawab Pertanyaan Keamanan

14 Kasus 1

15 Dalam kasus pertama, hasil yang diharapkan adalah password bisa dikembalikan. Maka dari
16 itu, masukan teks ini akan diisi dengan jawaban yang benar dari masing-masing pertanyaan.
17 Gambar 5.10 menunjukkan masukan teks yang sudah diisi dengan jawaban yang benar dari
18 masing-masing pertanyaan.

Retrieve Password

Security Questions

| | | |
|----|---|---|
| 1. | Siapa nama anda? | <input type="text" value="Samuel"/> |
| 2. | Dimana kota anda lahir? | <input type="text" value="Bandung"/> |
| 3. | Apakah anda memiliki binatang peliharaan? | <input type="text" value="Ya"/> |
| 4. | Kapan anda lahir? | <input type="text" value="31 Juli 1993"/> |
| 5. | Dimana anda berkuliah? | <input type="text" value="UNPAR"/> |

Gambar 5.10: Tampilan Menjawab Pertanyaan Keamanan Kasus Pertama

1 Setelah mengisi jawaban untuk masing-masing pertanyaan, langkah berikutnya adalah
 2 memroses jawaban dari masing-masing pertanyaan ini dengan metode skema $\text{threshold}(k, n)$
 3 yang sudah dibahas pada Bab 2.8. Untuk memroses hal ini, maka tombol "*Submit*" pada
 4 Gambar 5.10 perlu ditekan.

5 Setelah tombol "*Submit*" pada Gambar 5.10 ditekan, perangkat lunak akan memroses
 6 setiap jawaban masing-masing pertanyaan. Dalam kasus pertama, seluruh pertanyaan bisa
 7 dijawab dengan benar, maka *password* bisa dikembalikan. Gambar 5.11 menunjukkan bahwa
 8 *password* bisa dikembalikan.

Retrieve Password

| | |
|-------------------------|--|
| Your password is | <input type="text" value="password1"/> |
| Your password is | <input type="text" value="password2"/> |
| Your password is | <input type="text" value="password3"/> |
| Your password is | <input type="text" value="password4"/> |
| Your password is | <input type="text" value="password5"/> |

Gambar 5.11: Hasil Pengujian Fungsional Kasus Pertama

9 Kasus 2

10 Kasus selanjutnya adalah kasus kedua dimana *password* tidak bisa dikembalikan. Dalam
 11 kasus ini, diasumsikan hanya 3 pertanyaan saja yang bisa dijawab dengan benar. Gambar
 12 5.12 menunjukkan tampilan menjawab pertanyaan untuk kasus 2.

The screenshot shows a 'Retrieve Password' form with the following security questions and answers:

| Security Questions | Answers |
|--|--------------|
| 1. Siapa nama anda? | Samuel |
| 2. Dimana kota anda lahir? | Surabaya |
| 3. Apakah anda memiliki binatang peliharaan? | Tidak |
| 4. Kapan anda lahir? | 31 Juli 1993 |
| 5. Dimana anda berkuliah? | UNPAR |

At the bottom of the form are two buttons: 'Submit' (blue) and 'Cancel' (red).

Gambar 5.12: Tampilan Menjawab Pertanyaan Keamanan Kasus Kedua

1 Pada Gambar 5.12 dapat dilihat bahwa hanya ada 3 pertanyaan yang dijawab dengan
 2 benar. Pertanyaan nomor 2 dan 3 diberi tanda untuk menunjukkan bahwa jawaban dari
 3 pertanyaan tersebut tidak tepat.

4 Langkah selanjutnya adalah memroses jawaban dari masing-masing pertanyaan dengan
 5 menekan tombol "Submit". Setelah tombol "Submit" ditekan, perangkat lunak akan me-
 6 nampilkan hasil pengujian kasus kedua. Dalam kasus kedua, karena k yang dipilih $k = 4$ dan
 7 hanya 3 pertanyaan yang dijawab dengan benar, maka *password* tidak bisa dikembalikan.
 8 Gambar 5.13 menunjukkan langkah yang sudah dijelaskan.

The screenshot shows the 'Retrieve Password' form with a red error message box that says "Password cannot be retrieved". Below the message is a blue button labeled "Done".

Gambar 5.13: Hasil Pengujian Fungsional Kasus Kedua

9 5.2.3 Analisis Hasil Pengujian Fungsional

10 Pada bagian ini akan dibahas analisis dari hasil pengujian fungsional yang sudah dilakuk-
 11 an. Seperti yang sudah dibahas, pengujian fungsional dibagi menjadi 2 kasus, yaitu kasus
 12 *password* bisa dikembalikan dan kasus dimana *password* tidak bisa dikembalikan.

13 Untuk kasus pertama, dapat dilihat bahwa jika pertanyaan keamanan yang dijawab
 14 benar lebih besar atau sama dengan nilai k yang sudah ditentukan, maka semua *password*
 15 bisa dikembalikan. Dalam kasus pertama, seluruh pertanyaan dapat dijawab dengan benar,
 16 $pertanyaanbenar = 5$. Kemudian untuk kasus pertama, nilai $k = 4$ dan $pertanyaanbenar >$
 17 k . Maka dari itu, *password* bisa dikembalikan.

18 Untuk kasus kedua, dapat dilihat bahwa jika pertanyaan keamanan yang dijawab benar
 19 kurang dari k , maka *password* tidak bisa dikembalikan. Dalam kasus kedua, banyak perta-

nyaan yang dijawab benar hanya 3 pertanyaan, $pertanyaanbenar = 3$. Jadi, karena $k = 4$ dan $pertanyaanbenar < k$, *password* tidak bisa dikembalikan.

Jadi, kesimpulan yang dapat diambil dari hasil pengujian kasus pertama dan kedua adalah perangkat lunak sudah bisa mengimplementasikan metode *secret sharing* Shamir untuk mengembalikan n password dengan menjawab n pertanyaan.

5.2.4 Analisis dan Hasil Pengujian Survei

Pada bagian ini akan ditunjukkan hasil pengujian survei. Seperti yang sudah dijelaskan, pengujian survei bertujuan untuk menilai kualitas dari pertanyaan keamanan dengan melihat tingkat kesulitan untuk menebak atau menjawab jawaban benar. Asumsi yang digunakan dalam pengujian ini adalah seluruh jawaban relevan dengan pertanyaan keamanan.

Pengujian survei ini terbagi atas 4 kasus. Responden melakukan survei dengan cara mencoba untuk menebak jawaban dari pertanyaan keamanan untuk mengembalikan password. Setiap orang bebas memilih cara untuk mendapatkan jawaban dari pertanyaan selain tidak bertanya kepada pembuat pertanyaan keamanan.

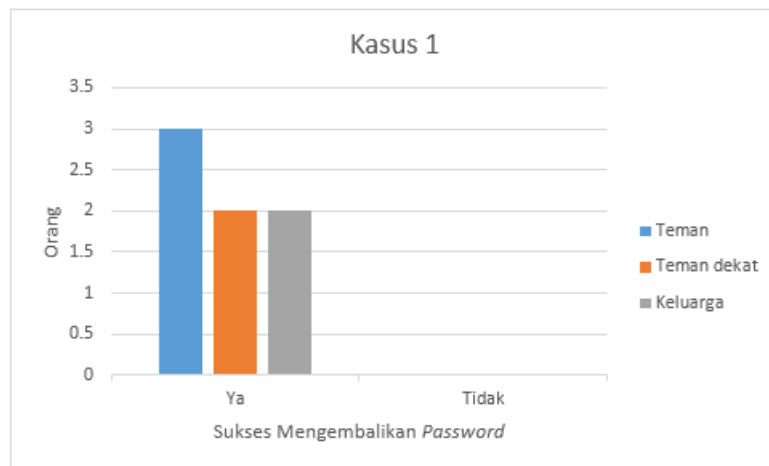
Kasus 1, 2, dan 3 memiliki 10 pertanyaan keamanan, $n = 10$, dan minimal 4 pertanyaan keamanan dijawab benar, $k = 4$. Sementara itu, untuk kasus 4 nilai n dan k ditambah. Dalam kasus 4, terdapat 15 pertanyaan keamanan, $n = 15$, dan minimal 6 pertanyaan keamanan dijawab benar, $k = 6$. Berikut tabel hasil survei untuk setiap kasus beserta dengan penjelasannya. Untuk kasus 1 dan 2 survei dilakukan terhadap 7 orang responden, sedangkan untuk kasus 3 dan 4 survei dilakukan terhadap 20 orang responden.

Kasus 1

Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 1:

1. Apa jenis kelamin anda? (Laki-laki/Perempuan)
2. Apakah anda pernah ke luar negeri?
3. Apakah anda mempunyai binatang peliharaan?
4. Apakah anda bermain alat musik?
5. Apakah anda pernah tidak naik kelas?
6. Apakah anda pernah mengalami kecelakaan?
7. Apakah anda menyukai kegiatan olahraga?
8. Apa nama belakang anda?
9. Siapa nama ibu anda?
10. Siapa nama ayah anda?

Kemudian, hasil dari survei kasus 1 ditunjukkan oleh Grafik 5.14.



Gambar 5.14: Pengujian survei kasus 1

1 Analisis Kasus 1

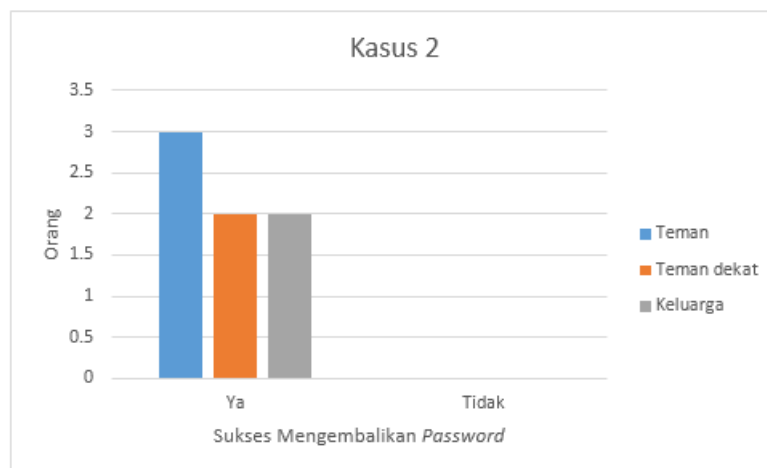
2 Dilihat dari grafik 5.14 untuk topik kasus 1, seluruh responden bisa berhasil mengembalik-
 3 an *password*. Hal ini karena mayoritas kemungkinan jawaban dari pertanyaan keamanan
 4 adalah kemungkinan biner dengan hanya 2 kemungkinan saja (Ya atau Tidak). Pertanyaan
 5 keamanan yang kemungkinan jawabannya hanya 2 kemungkinan saja bisa dengan mudah
 6 ditebak. Karena setiap jawaban bisa dengan mudah ditebak, maka seluruh responden bisa
 7 mengembalikan *password* dengan mudah.

8 Kasus 2

9 Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 2:

- 10 1. Pada tahun berapa anda lahir?
- 11 2. Pada tanggal berapa anda lahir?
- 12 3. Pada bulan apa anda lahir?
- 13 4. Berapa perbedaan umur anda dengan ayah anda?
- 14 5. Berapa perbedaan umur anda dengan ibu anda?
- 15 6. Berapa orang saudara anda?
- 16 7. Berapa nomor rumah tempat anda tinggal?
- 17 8. Dimana anda tinggal?
- 18 9. Apa merek kendaraan yang anda pakai?
- 19 10. Pada hari apa anda lahir?

20 Kemudian, hasil dari survei kasus 2 ditunjukkan oleh Grafik 5.15.



Gambar 5.15: Pengujian survei kasus 2

1 Analisis Kasus 2

2 Dilihat dari grafik 5.15 untuk topik kasus 2, seluruh responden berhasil untuk mengembalik-
 3 an password. Hal ini disebabkan karena kemungkinan jawaban dari pertanyaan keamanan
 4 hanya berupa angka saja, khususnya hanya tanggal ulang tahun, bulan lahir, atau tahun
 5 lahir.

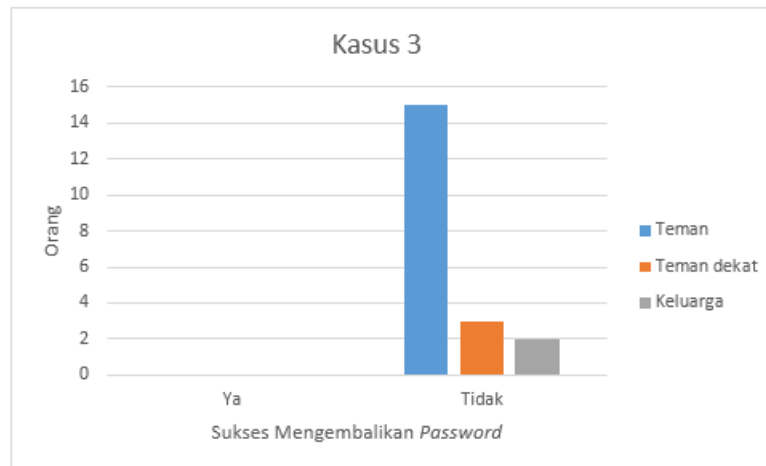
6 Responden dapat menjawab tanggal lahir karena hanya memiliki 30-31 kemungkinan,
 7 sedangkan untuk bulan hanya ada 12 kemungkinan, dan juga beberapa pertanyaan lain
 8 yang menyangkut angka. Dapat dilihat juga, bahwa beberapa jawaban untuk pertanyaan
 9 keamanan merupakan informasi yang sering ditunjukkan dalam profil sosial media, karena
 10 dari itu jawaban yang tepat bisa dengan mudah didapatkan.

11 Kasus 3

12 Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 3:

- 13 1. Pada jam berapa anda lahir?(jj:mm)
- 14 2. Apa nama sekolah dasar tempat anda bersekolah?
- 15 3. Siapa nama belakang sepupu paling tua dari keluarga sisi ibu anda?
- 16 4. Siapa nama belakang sepupu paling tua dari keluarga sisi ayah anda?
- 17 5. Apa cita-cita anda dulu sewaktu kecil?
- 18 6. Siapa nama anak paling tua dari nenek sisi ibu anda?
- 19 7. Apa binatang peliharaan pertama anda?
- 20 8. Apa alat musik yang anda mainkan pertama kali?
- 21 9. Dimana kerabat terdekat anda tinggal/berasal?
- 22 10. Siapa nama guru kelas 3 SD anda?

23 Kemudian, hasil dari survei kasus 3 ditunjukkan oleh Grafik 5.16.



Gambar 5.16: Pengujian survei kasus 3

1 Analisis Kasus 3

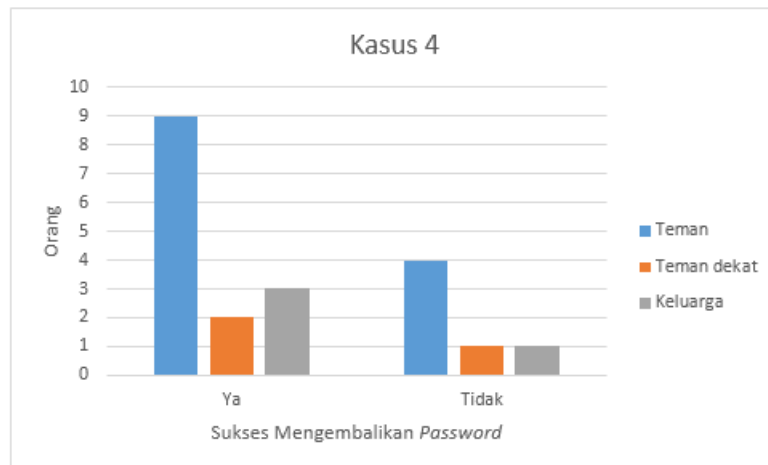
2 Untuk topik kasus 3, tidak ada responden yang berhasil mengembalikan *password*. Hal ini
 3 karena beberapa pertanyaan sifatnya sangat personal dan jawaban tidak bisa dengan mudah
 4 ditebak atau dicari di *internet* atau media sosial. Pertanyaan yang sifatnya sangat personal
 5 akan mempersulit untuk mengembalikan *password* kecuali bagi pembuat pertanyaan.

6 Kasus 4

7 Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 4:

- 8 1. Apakah anda mempunyai binatang peliharaan?
- 9 2. Apakah anda bermain alat musik?
- 10 3. Apakah anda pernah tidak naik kelas?
- 11 4. Apa nama belakang anda?
- 12 5. Siapa nama ibu anda?
- 13 6. Pada hari apa anda lahir?
- 14 7. Pada tanggal berapa anda lahir?
- 15 8. Pada bulan apa anda lahir?
- 16 9. Berapa perbedaan umur anda dengan ayah anda?
- 17 10. Berapa nomor rumah tempat anda tinggal?
- 18 11. Pada jam berapa anda lahir?(jj:mm)
- 19 12. Apa cita-cita anda dulu sewaktu kecil?
- 20 13. Siapa nama anak paling tua dari nenek sisi ibu anda?
- 21 14. Apa binatang peliharaan pertama anda?
- 22 15. Siapa nama guru kelas 3 SD anda?

23 Kemudian, Grafik 5.17 menunjukkan hasil survei kasus 4.



Gambar 5.17: Pengujian survei kasus 4

1 Analisis Kasus 4

2 Dilihat dari grafik 5.17 untuk topik kasus 4, tingkat keberhasilannya tetap tinggi walaupun
 3 topik kasus 4 ini merupakan gabungan dari topik kasus 1, 2, dan 3. Hal ini disebabkan
 4 karena mayoritas terdiri pertanyaan dari kasus 1 dan kasus 2.

5 Responden hanya cukup menjawab 6 pertanyaan benar dari 15 pertanyaan dalam kasus
 6 4, maka responden pun cukup menjawab 3 pertanyaan dari kasus 1 dan 3 pertanyaan dari
 7 kasus 2 dengan benar, responden tidak perlu menjawab satupun pertanyaan dari kasus 3.
 8 Dapat disimpulkan, bahwa meningkatkan banyak pertanyaan keamanan tidak mempersulit
 9 untuk mengembalikan *password*.

10 5.2.5 Kesimpulan Pengujian

11 Dari 4 kasus pengujian yang dilakukan maka bisa ditarik beberapa kesimpulan dalam penila-
 12 ian kualitas pertanyaan keamanan personal. Pertanyaan keamanan personal harus memiliki
 13 5 sifat:

- 14 • Aman
 15 Pertanyaan keamanan harus tidak mudah ditebak dan tidak mudah diselidiki (*googling*).
 16
- 17 • Stabil
 18 Pertanyaan keamanan tidak boleh berubah seiring berjalannya waktu.
- 19 • Mudah diingat
 20 Pertanyaan keamanan harus sifatnya personal sehingga mudah untuk diingat.
- 21 • Sederhana
 22 Pertanyaan keamanan harus sederhana tetapi sifatnya tetap personal.
- 23 • Memiliki banyak kemungkinan jawaban
 24 Pertanyaan keamanan tidak boleh hanya memiliki kemungkinan jawaban yang sedikit
 25 karena akan mudah ditebak (dengan teknik *brute force*).

1 Namun, beberapa pertanyaan keamanan mungkin memiliki banyak kemungkinan jawab-
2 an dan aman sehingga tidak mudah ditebak tetapi tidak mudah diingat karena jawabannya
3 terlalu rumit. Beberapa pertanyaan keamanan juga mungkin tidak sesuai dengan situasi
4 atau keadaan dari pembuat pertanyaan. Sehingga, tidak ada pertanyaan keamanan yang
5 memiliki tepat 5 sifat yang diatas.

BAB 6

KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dan saran dari penelitian yang dilakukan. Kesimpulan akan menjawab rumusan masalah yang sudah dibuat pada Bab 1 dan saran akan berisi pengembangan lebih lanjut dari penelitian ini.

6.1 Kesimpulan

Dari hasil pengujian fungsional yang dilakukan, dapat dilihat bahwa pengguna cukup menjawab beberapa saja pertanyaan keamanan untuk mengembalikan banyak *password* sehingga pengguna tidak perlu mengingat setiap jawaban dari pertanyaan keamanan. Dari hasil ini, dapat disimpulkan bahwa perangkat lunak yang mengimplementasikan *secret sharing* Shamir berhasil dibangun.

Dari hasil pengujian survei, dapat diambil hal penting yang berhubungan dengan pemilihan jenis pertanyaan keamanan yang dibuat. Jenis pertanyaan keamanan yang dibuat dapat dinilai dengan melihat 5 sifat berikut:

1. Aman
2. Stabil
3. Mudah diingat
4. Sederhana
5. Memiliki banyak kemungkinan jawaban

Dari hasil pengujian survei juga, dapat diketahui bahwa tidak ada pertanyaan keamanan yang memiliki kelima sifat secara sekaligus, beberapa dari sifat ada yang berlawanan sehingga tidak mungkin dapat dimiliki oleh sebuah pertanyaan keamanan sekaligus. Selain itu, jenis pertanyaan keamanan yang dibuat dapat mempengaruhi nilai entropi dari pertanyaan keamanan. Pertanyaan keamanan yang memiliki nilai entropi tinggi dapat dengan mudah ditebak atau diprediksi.

Jadi, dari hasil pengujian untuk penelitian yang dilakukan, dapat diambil kesimpulan bahwa metode *secret sharing* Shamir dapat digunakan untuk mengembalikan n password dengan n pertanyaan keamanan, perangkat lunak pengingat *password* yang mengimplementasikan metode *secret sharing* Shamir berhasil dibangun, dan kualitas dari pertanyaan keamanan dapat dinilai dari 5 sifat yang sudah dipaparkan di atas.

1 6.2 Saran

2 Dari penelitian ini, terdapat beberapa saran untuk pengembangan perangkat lunak lebih
3 lanjut, yaitu:

- 4 • Algoritma enkripsi yang digunakan bisa diganti dengan menggunakan algoritma enk-
5 ripsi yang menggunakan panjang kunci lebih panjang dari 64-bit. Pada penelitian ini,
6 algoritma enkripsi yang digunakan adalah *data encryption standard* (DES) dengan
7 panjang kunci maksimal 64-bit. Untuk ukuran keamanan informasi, 64-bit merupakan
8 ukuran yang kurang dan nantinya untuk pengembangan lebih lanjut bisa digunakan
9 algoritma enkripsi yang memiliki panjang kunci lebih dari 64-bit.
- 10 • Metode *secret sharing* Shamir diharapkan dapat diimplementasikan tidak hanya pa-
11 da perangkat lunak perorangan seperti dalam penelitian ini, tetapi bisa diimplemen-
12 tasikan pada sebuah sistem besar yang memiliki subsistem dan masing-masing dari
13 subsistem ini menyimpan banyak informasi penting.

DAFTAR REFERENSI

- [1] R. Munir, *Matematika Diskrit*. Informatika Bandung, 2010.
- [2] B. A. Forouzan, *Cryptography & Network Security*. McGraw-Hill, Inc., 2007.
- [3] D. Norman and D. Wolczuk, *Introduction to Linear Algebra for Science and Engineering*. Pearson, 2012.
- [4] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [5] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability and statistics for engineers and scientists*, vol. 5. Macmillan New York, 1993.
- [6] C. Shannon, “A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656,” *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948.
- [7] C. Ellison, C. Hall, R. Milbert, and B. Schneier, “Protecting secret keys with personal entropy,” *Future Generation Computer Systems*, vol. 16, no. 4, pp. 311–318, 2000.

LAMPIRAN A

THE SOURCE CODE

Listing A.1: DESEncryption.java

```
3
4 package DES;
5
6 import java.util.ArrayList;
7
8 /**
9  *
10  * @author Samuel Christian
11  */
12 public class DESEncryption {
13
14     /** ATTRIBUTES
15      */
16
17     //PC1 -> subkey permutation box
18     private int[] PC1 = {
19         57, 49, 41, 33, 25, 17, 9,
20         1, 58, 50, 42, 34, 26, 18,
21         10, 2, 59, 51, 43, 35, 27,
22         19, 11, 3, 60, 52, 44, 36,
23         63, 55, 47, 39, 31, 23, 15,
24         7, 62, 54, 46, 38, 30, 22,
25         14, 6, 61, 53, 45, 37, 29,
26         21, 13, 5, 28, 20, 12, 4
27     };
28
29     //PC2 -> subkey permutation final box
30     private int[] PC2 = {
31         14, 17, 11, 24, 1, 5,
32         3, 28, 15, 6, 21, 10,
33         23, 19, 12, 4, 26, 8,
34         16, 7, 27, 20, 13, 2,
35         41, 52, 31, 37, 47, 55,
36         30, 40, 51, 45, 33, 48,
37         44, 49, 39, 56, 34, 53,
38         46, 42, 50, 36, 29, 32
39     };
40
41     //initial permutation
42     private int[] IP = {
43         58, 50, 42, 34, 26, 18, 10, 2,
44         60, 52, 44, 36, 28, 20, 12, 4,
45         62, 54, 46, 38, 30, 22, 14, 6,
46         64, 56, 48, 40, 32, 24, 16, 8,
47         57, 49, 41, 33, 25, 17, 9, 1,
48         59, 51, 43, 35, 27, 19, 11, 3,
49         61, 53, 45, 37, 29, 21, 13, 5,
50         63, 55, 47, 39, 31, 23, 15, 7
51     };
52
53     //expansion P-box
54     private int[] exp = {
55         32, 1, 2, 3, 4, 5,
56         4, 5, 6, 7, 8, 9,
57         8, 9, 10, 11, 12, 13,
58         12, 13, 14, 15, 16, 17,
59         16, 17, 18, 19, 20, 21,
60         20, 21, 22, 23, 24, 25,
61         24, 25, 26, 27, 28, 29,
62         28, 29, 30, 31, 32, 1
63     };
64
65     //s-boxes
66     private int[][] s1 = {
67         {14, 4, 14, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
68         {0, 15, 7, 4, 14, 2, 13, 10, 3, 6, 12, 11, 9, 5, 3, 8},
69         {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
70         {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
71     };
72     private int[][] s2 = {
73         {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
74         {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
75         {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
76         {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
77     };
```

```

1      };
2
3      private int[][] s3 = {
4          {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
5          {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
6          {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
7          {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}
8      };
9
10     private int[][] s4 = {
11         {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
12         {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
13         {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
14         {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
15     };
16
17     private int[][] s5 = {
18         {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
19         {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
20         {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
21         {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}
22     };
23
24     private int[][] s6 = {
25         {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
26         {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
27         {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
28         {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 10, 0, 8, 13}
29     };
30
31     private int[][] s7 = {
32         {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
33         {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
34         {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
35         {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}
36     };
37
38     private int[][] s8 = {
39         {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
40         {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 10, 14, 9, 2},
41         {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
42         {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}
43     };
44
45     //straight permutation box after s-box substitution
46     private int[] P = {
47         16, 7, 20, 21,
48         29, 12, 28, 17,
49         1, 15, 23, 26,
50         5, 18, 31, 10,
51         2, 8, 24, 14,
52         32, 27, 3, 9,
53         19, 13, 30, 6,
54         22, 11, 4, 25
55     };
56
57     //final permutation
58     private int[] IP1 = {
59         40, 8, 48, 16, 56, 24, 64, 32,
60         39, 7, 47, 15, 55, 23, 63, 31,
61         38, 6, 46, 14, 54, 22, 62, 30,
62         37, 5, 45, 13, 53, 21, 61, 29,
63         36, 4, 44, 12, 52, 20, 60, 28,
64         35, 3, 43, 11, 51, 19, 59, 27,
65         34, 2, 42, 10, 50, 18, 58, 26,
66         33, 1, 41, 9, 49, 17, 57, 25
67     };
68
69     private String strMsg;
70     private String strKey;
71     private String msg;
72     private String key;
73     private String[] roundKey;
74     private String cipher;
75     private ArrayList<int[][]> sBox;
76     private String[] msgBlock;
77
78     public Encryption() {
79         strMsg = "";
80         strKey = "";
81         msg = "";
82         key = "";
83         roundKey = new String[16];
84         cipher = "";
85         msgBlock = new String[1];
86
87         sBox = new ArrayList<int[][]>();
88         sBox.add(s1);
89         sBox.add(s2);
90         sBox.add(s3);
91         sBox.add(s4);
92         sBox.add(s5);
93         sBox.add(s6);
94         sBox.add(s7);
95         sBox.add(s8);
96     }
97
98     public void reset() {
99         strMsg = "";
100        strKey = "";
101        msg = "";
102        key = "";
103        roundKey = new String[16];

```

```

1         cipher = "";
2         msgBlock = new String[1];
3     }
4
5     public void encrypt() {
6         createSubKey();
7         String tempCipher = "";
8         for(int block = 0; block < msgBlock.length; block++) {
9             String init = initialPermutation(msgBlock[block]);
10            String L0 = init.substring(0,init.length()/2);
11            String R0 = init.substring(init.length()/2, init.length());
12
13            //16 rounds
14            String[] arr = {L0, R0};
15            for(int i = 0; i < 16; i++) {
16                arr = round(arr[0], arr[1], roundKey[i]);
17            }
18
19            tempCipher = arr[1] + arr[0];
20            tempCipher = permute(tempCipher, IP1);
21            cipher += tempCipher;
22        }
23    }
24
25    public String getCipherText() {
26        String cipherText = "";
27        String[] temp = cipher.split("(?<=\\G.{4})");
28        for(int i = 0; i < temp.length; i++) {
29            cipherText += Integer.toHexString(Integer.parseInt(temp[i], 2));
30        }
31        return cipherText;
32    }
33
34    private String[] round(String left, String right, String rndKey) {
35        String Ln = right;
36        String Rn = xor(left, feistelCipher(right, rndKey));
37        String[] arr = {Ln, Rn};
38        return arr;
39    }
40
41    private String feistelCipher(String right, String rndKey) {
42        String res = "";
43
44        res = xor(rndKey, permute(right, exp));
45
46        String[] B = res.split("(?<=\\G.{6})");
47        String temp = "";
48        for(int i = 0; i < sBox.size(); i++) {
49            String index = B[i];
50            int[][] sbox = sBox.get(i);
51            int row = Integer.parseInt(index.charAt(0) + "" + index.charAt(index.length()-1) + ""
52                ,2);
53            int column = Integer.parseInt(index.substring(1, index.length()-1), 2);
54            temp += String.format("%4s", Integer.toBinaryString(sbox[row][column])).replace(' ', '
55                0');
56        }
57
58        res = permute(temp, P);
59
60        return res;
61    }
62
63    private String initialPermutation(String msgblock) {
64        String res = permute(msgblock, IP);
65        return res;
66    }
67
68    private void createSubKey() {
69        String K2 = permute(key, PC1);
70        String left = K2.substring(0, K2.length()/2);
71        String right = K2.substring(K2.length()/2, K2.length());
72        for(int i = 0; i < 16; i++) {
73            int shift = 2;
74            int round = i + 1;
75            if(round == 1 || round == 2 || round == 9 || round == 16) {
76                shift = 1;
77            }
78            String leftN = leftShift(left, shift);
79            String rightN = leftShift(right, shift);
80            roundKey[i] = leftN+rightN;
81            left = leftN;
82            right = rightN;
83        }
84
85        for(int i = 0; i < roundKey.length; i++) {
86            roundKey[i] = permute(roundKey[i], PC2);
87        }
88    }
89
90    private String leftShift(String text, int bit) {
91        String res = "";
92        char[] ch = new char[text.length()];
93        for(int i = 0; i < ch.length; i++) {
94            int shift = (i-bit) < 0 ? text.length()+(i-bit) : (i-bit);
95            ch[shift] = text.charAt(i);
96        }
97        res = new String(ch);
98        return res;
99    }
100
101    public void initialize() {
102        String tempMessage = "";
103        for(int i = 0; i < strMsg.length(); i++) {

```

```

1      tempMessage += String.format("%8s", Integer.toBinaryString(strMsg.charAt(i))).replace(
2          '\u0000', '0');
3      }
4
5      int padLength = (64 - (tempMessage.length() % 64)) / 8;
6      String padding = "";
7      for(int i = 0; i < padLength; i++) {
8          padding += String.format("%8s", Integer.toBinaryString(32)).replace('\u0000', '0');
9      }
10
11     msg = tempMessage + padding;
12
13     msgBlock = msg.split("(?<=\\G.{64})");
14
15     for(int i = 0; i < strKey.length(); i++) {
16         key += String.format("%8s", Integer.toBinaryString(strKey.charAt(i))).replace('\u0000', '0'
17             );
18     }
19 }
20 public void setMessage(String m) {
21     strMsg = m;
22 }
23
24 public void setKey(String k) {
25     strKey = k;
26 }
27
28 private String permute(String text, int[] permutationBox) {
29     String res = "";
30     for(int i = 0; i < permutationBox.length; i++) {
31         res += text.charAt(permutationBox[i]-1);
32     }
33     return res;
34 }
35
36 private String xor(String a, String b) {
37     String res = "";
38     for(int i = 0; i < a.length(); i++) {
39         int temp1 = a.charAt(i)-48;
40         int temp2 = b.charAt(i)-48;
41         int xor = temp1 ^ temp2;
42         res += xor;
43     }
44     return res;
45 }
46 }

```

Listing A.2: DESDecryption.java

```

47
48 package DES;
49
50 import java.util.ArrayList;
51
52 /**
53  *
54  * @author Samuel Christian
55  */
56 public class DESDecryption {
57
58     /**
59      * ATTRIBUTES
60      */
61
62     //PC1 -> subkey permutation box
63     private int[] PC1 = {
64         57, 49, 41, 33, 25, 17, 9,
65         1, 58, 50, 42, 34, 26, 18,
66         10, 2, 59, 51, 43, 35, 27,
67         19, 11, 3, 60, 52, 44, 36,
68         63, 55, 47, 39, 31, 23, 15,
69         7, 62, 54, 46, 38, 30, 22,
70         14, 6, 61, 53, 45, 37, 29,
71         21, 13, 5, 28, 20, 12, 4
72     };
73
74     //PC2 -> subkey permutation final box
75     private int[] PC2 = {
76         14, 17, 11, 24, 1, 5,
77         3, 28, 15, 6, 21, 10,
78         23, 19, 12, 4, 26, 8,
79         16, 7, 27, 20, 13, 2,
80         41, 52, 31, 37, 47, 55,
81         30, 40, 51, 45, 33, 48,
82         44, 49, 39, 56, 34, 53,
83         46, 42, 50, 36, 29, 32
84     };
85
86     //initial permutation
87     private int[] IP = {
88         58, 50, 42, 34, 26, 18, 10, 2,
89         60, 52, 44, 36, 28, 20, 12, 4,
90         62, 54, 46, 38, 30, 22, 14, 6,
91         64, 56, 48, 40, 32, 24, 16, 8,
92         57, 49, 41, 33, 25, 17, 9, 1,
93         59, 51, 43, 35, 27, 19, 11, 3,
94         61, 53, 45, 37, 29, 21, 13, 5,
95         63, 55, 47, 39, 31, 23, 15, 7
96     };
97
98     //expansion P-box
99     private int[] exp = {

```



```

1      32, 1, 2, 3, 4, 5,
2      4, 5, 6, 7, 8, 9,
3      8, 9, 10, 11, 12, 13,
4      12, 13, 14, 15, 16, 17,
5      16, 17, 18, 19, 20, 21,
6      20, 21, 22, 23, 24, 25,
7      24, 25, 26, 27, 28, 29,
8      28, 29, 30, 31, 32, 1
9  };
10
11  //s-boxes
12  private int [][] s1 = {
13      {14, 4, 14, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
14      {0, 15, 7, 4, 14, 2, 13, 10, 3, 6, 12, 11, 9, 5, 3, 8},
15      {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
16      {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
17  };
18  private int [][] s2 = {
19      {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
20      {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
21      {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
22      {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
23  };
24
25  private int [][] s3 = {
26      {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
27      {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
28      {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
29      {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}
30  };
31
32  private int [][] s4 = {
33      {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
34      {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
35      {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
36      {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
37  };
38
39  private int [][] s5 = {
40      {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
41      {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
42      {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
43      {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}
44  };
45
46  private int [][] s6 = {
47      {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
48      {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
49      {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
50      {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 10, 0, 8, 13}
51  };
52
53  private int [][] s7 = {
54      {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
55      {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
56      {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
57      {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}
58  };
59
60  private int [][] s8 = {
61      {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
62      {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 10, 14, 9, 2},
63      {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
64      {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}
65  };
66
67  //straight permutation box after s-box substitution
68  private int [] P = {
69      16, 7, 20, 21,
70      29, 12, 28, 17,
71      1, 15, 23, 26,
72      5, 18, 31, 10,
73      2, 8, 24, 14,
74      32, 27, 3, 9,
75      19, 13, 30, 6,
76      22, 11, 4, 25
77  };
78
79  //final permutation
80  private int [] IP1 = {
81      40, 8, 48, 16, 56, 24, 64, 32,
82      39, 7, 47, 15, 55, 23, 63, 31,
83      38, 6, 46, 14, 54, 22, 62, 30,
84      37, 5, 45, 13, 53, 21, 61, 29,
85      36, 4, 44, 12, 52, 20, 60, 28,
86      35, 3, 43, 11, 51, 19, 59, 27,
87      34, 2, 42, 10, 50, 18, 58, 26,
88      33, 1, 41, 9, 49, 17, 57, 25
89  };
90
91  private String key;
92  private String[] roundKey;
93  private String cipher;
94  private ArrayList<int[][]> sBox;
95
96  public DESDecryption() {
97      key = "";
98      roundKey = new String[16];
99      cipher = "";
100
101      sBox = new ArrayList<int[][]>();
102      sBox.add(s1);
103      sBox.add(s2);

```

```

1      sBox.add(s3);
2      sBox.add(s4);
3      sBox.add(s5);
4      sBox.add(s6);
5      sBox.add(s7);
6      sBox.add(s8);
7  }
8
9  public void reset() {
10     key = "";
11     roundKey = new String[16];
12     cipher = "";
13 }
14
15 public void setCipher(String c) {
16     String bin = "";
17     for(int i = 0; i < c.length(); i++) {
18         int hex = Integer.parseInt(c.charAt(i)+"", 16);
19         String temp = String.format("%4s", Integer.toBinaryString(hex)).replace(' ', '0');
20         bin += temp;
21     }
22     cipher = bin;
23 }
24
25 public void setKey(String k) {
26     for(int i = 0; i < k.length(); i++) {
27         key += String.format("%8s", Integer.toBinaryString(k.charAt(i))).replace(' ', '0');
28     }
29 }
30
31 public String decrypt() {
32     createSubKey();
33     String plainText = "";
34
35     String[] cipherBlock = cipher.split("(?<=\\G.{64})");
36
37     for(int block = 0; block < cipherBlock.length; block++) {
38         String init = initialPermutation(cipherBlock[block]);
39         String L0 = init.substring(0, init.length()/2);
40         String R0 = init.substring(init.length()/2, init.length());
41
42         //16 rounds inverse
43         String[] arr = {L0, R0};
44         for(int i = 15; i >= 0; i--) {
45             arr = round(arr[0], arr[1], roundKey[i]);
46         }
47
48         String tempPlainText = arr[1] + arr[0];
49         tempPlainText = permute(tempPlainText, IP1);
50         plainText += tempPlainText;
51     }
52
53     return plainText;
54 }
55
56 private String[] round(String left, String right, String rndKey) {
57     String Ln = right;
58     String Rn = xor(left, feistelCipher(right, rndKey));
59     String[] arr = {Ln, Rn};
60     return arr;
61 }
62
63 private String feistelCipher(String right, String rndKey) {
64     String res = "";
65
66     res = xor(rndKey, permute(right, exp));
67
68     String[] B = res.split("(?<=\\G.{6})");
69     String temp = "";
70     for(int i = 0; i < sBox.size(); i++) {
71         String index = B[i];
72         int[][] sbox = sBox.get(i);
73         int row = Integer.parseInt(index.charAt(0) + "" + index.charAt(index.length()-1) + ""
74             ,2);
75         int column = Integer.parseInt(index.substring(1, index.length()-1), 2);
76         temp += String.format("%4s", Integer.toBinaryString(sbox[row][column])).replace(' ', '
77             0');
78     }
79
80     res = permute(temp, P);
81
82     return res;
83 }
84
85 private String initialPermutation(String msgblock) {
86     String res = permute(msgblock, IP);
87     return res;
88 }
89
90 private void createSubKey() {
91     String K2 = permute(key, PC1);
92     String left = K2.substring(0, K2.length()/2);
93     String right = K2.substring(K2.length()/2, K2.length());
94     for(int i = 0; i < 16; i++) {
95         int shift = 2;
96         int round = i + 1;
97         if(round == 1 || round == 2 || round == 9 || round == 16) {
98             shift = 1;
99         }
100         String leftN = leftShift(left, shift);
101         String rightN = leftShift(right, shift);
102         roundKey[i] = leftN+rightN;
103         left = leftN;

```

```

1      right = rightN;
2  }
3
4      for(int i = 0; i < roundKey.length; i++) {
5          roundKey[i] = permute(roundKey[i], PC2);
6      }
7  }
8
9  private String leftShift(String text, int bit) {
10     String res = "";
11     char[] ch = new char[text.length()];
12     for(int i = 0; i < ch.length; i++) {
13         int shift = (i-bit) < 0 ? text.length()+(i-bit) : (i-bit);
14         ch[shift] = text.charAt(i);
15     }
16     res = new String(ch);
17     return res;
18 }
19
20 private String permute(String text, int[] permutationBox) {
21     String res = "";
22     for(int i = 0; i < permutationBox.length; i++) {
23         res += text.charAt(permutationBox[i]-1);
24     }
25     return res;
26 }
27
28 private String xor(String a, String b) {
29     String res = "";
30     for(int i = 0; i < a.length(); i++) {
31         int temp1 = a.charAt(i)-48;
32         int temp2 = b.charAt(i)-48;
33         int xor = temp1 ^ temp2;
34         res += xor;
35     }
36     return res;
37 }
38
39 public String binToStr(String bin) {
40     String text = "";
41     String[] temp = bin.split("(?<=\\G.{8})");
42     for(int i = 0; i < temp.length; i++) {
43         text += (char)Integer.parseInt(temp[i], 2);
44     }
45     return text;
46 }
47 }

```

Listing A.3: DataReader.java

```

48 package ReaderWriter;
49
50 import java.io.BufferedReader;
51 import java.io.FileReader;
52 import java.io.IOException;
53 import java.util.ArrayList;
54
55 /**
56  *
57  * @author Samuel Christian
58  */
59 public class DataReader {
60
61     /**
62      * ATTRIBUTES
63      */
64     private String filename;
65     private ArrayList<String> content;
66
67     public DataReader(String file) {
68         filename = file;
69         content = new ArrayList<String>();
70     }
71
72     public String[] get() {
73         String[] str = new String[content.size()];
74         for(int i = 0; i < str.length; i++) {
75             str[i] = content.get(i);
76         }
77         return str;
78     }
79
80     public void read() {
81         BufferedReader br = null;
82         try {
83             String sCurrentLine;
84             br = new BufferedReader(new FileReader(filename));
85             while ((sCurrentLine = br.readLine()) != null) {
86                 content.add(sCurrentLine);
87             }
88         } catch (IOException e) {
89             } finally {
90                 try {
91                     if (br != null) {
92                         br.close();
93                     }
94                 } catch (IOException ex) {
95                     }
96             }
97         }
98     }
99 }

```

Listing A.4: DataWriter.java

```

1  package ReaderWriter;
2
3
4  import java.io.BufferedWriter;
5  import java.io.File;
6  import java.io.FileWriter;
7  import java.io.IOException;
8
9  /**
10   *
11   * @author Samuel Christian
12   */
13  public class DataWriter {
14
15      /**
16       * ATTRIBUTES
17       */
18      private String filename;
19      private String content;
20
21      public DataWriter(String name) {
22          filename = name;
23      }
24
25      public void write(String text) throws IOException {
26          content = text;
27          try {
28              File file = new File(filename);
29
30              file.createNewFile();
31
32              FileWriter fw = new FileWriter(file.getAbsolutePath(), false);
33              BufferedWriter bw = new BufferedWriter(fw);
34              bw.write(content);
35              bw.close();
36
37              System.out.println("Done");
38
39          } catch (IOException e) {
40          }
41      }
42  }

```

Listing A.5: Sha512.java

```

43 package SHA512;
44
45 import java.math.BigInteger;
46
47 /**
48  *
49  * @author Samuel Christian
50  */
51 public class Sha512 {
52
53     /**
54      * ATTRIBUTES
55      */
56
57     /**Initial values*/
58     private static final long[] INITIALS = {0x6a09e667f3bcc908L,
59                                              0xbb67ae8584caa73bL,
60                                              0x3c6ef372fe94f82bL,
61                                              0xa54ff53a5f1d36f1L,
62                                              0x510e527fade682d1L,
63                                              0x9b05688c2b3e6c1fL,
64                                              0x1f83d9abfb41bd6bL,
65                                              0x5be0cd19137e2179L};
66
67     /**Constants for 80 rounds in sha512*/
68     private static final long[] CONSTANTS =
69     {0x428a2f98d728ae22L, 0x7137449123ef65cdL, 0xb5c0fbcfec4d3b2fL,
70      0xe9b5dba58189dbbcL,
71      0x3956c25bf348b538L, 0x59f111f1b605d019L, 0x923f82a4af194f9bL,
72      0xab1c5ed5da6d8118L,
73      0xd807aa98a3030242L, 0x12835b0145706fbeL, 0x243185be4ee4b28cL,
74      0x550c7dc3d5ffb4e2L,
75      0x72be5d74f27b896fL, 0x80deb1fe3b1696b1L, 0x9bdc06a725c71235L,
76      0xc19bf174cf692694L,
77      0xe49b69c19ef14ad2L, 0xefbe4786384f25e3L, 0x0fc19dc68b8cd5b5L,
78      0x240ca1cc77ac9c65L,
79      0x2de92cf6f592b0275L, 0x4a7484aa6ea6e483L, 0x5cb0a9dcb4d1fbd4L,
80      0x76f988da831153b5L,
81      0x983e5152ee66dfabL, 0xa831c66d2db43210L, 0xb00327c898fb213fL,
82      0xbf597fc7beef0ee4L,
83      0xc6e00bf33da88fc2L, 0xd5a79147930aa725L, 0x06ca6351e003826fL,
84      0x142929670a0e6e70L,
85      0x27b70a8546d22ffcL, 0x2e1b21385c26c926L, 0x4d2c6dfc5ac42aedL,
86      0x53380d139d95b3dfL,
87      0x650a73548baf63deL, 0x766a0abb3c77b2a8L, 0x81c2c92e47edaee6L,
88      0x92722c851482353bL,
89      0xa2bfe8a14cf10364L, 0xa81a664bb4c23001L, 0xc24b8b70d0f89791L,
90      0xc76c51a30654be30L,
91      0xd192e819d6ef5218L, 0xd69906245565a910L, 0xf40e35855771202aL,
92      0x106aa07032bbd1b8L,
93      0x19a4c116b8d2d0c8L, 0x1e376c085141ab53L, 0x2748774cdf8eeb99L,
94      0x34b0bcb5e19b48a8L,
95      0x391c0cb3c5c95a63L, 0x4ed8aa4ae3418ac6L, 0x5b9cca4f7763e373L,
96      0x682e6ff3d6b2b8a3L,
97

```

```

1          0x748f82ee5defb2fcL, 0x78a5636f43172f60L, 0x84c87814a1f0ab72L,
2          0x8cc702081a6439ecL,
3          0x90beffa23631e28L, 0xa4506cebde82bde9L, 0xbef9a3f7b2c67915L,
4          0xc67178f2e372532bL,
5          0xca273eceeaa26619cL, 0xd186b8c721c0c207L, 0xeda7dd6cde0eb1eL,
6          0xf57d4f7fee6ed178L,
7          0x06f067aa72176fbaL, 0x0a637dc5a2c898a6L, 0x113f9804bef90daeL,
8          0x1b710b35131c471bL,
9          0x28db77f523047d84L, 0x32caab7b40c72493L, 0x3c9ebe0a15c9bebcL,
10         0x431d67c49c100d4cL,
11         0x4cc5d4becb3e42b6L, 0x597f299cfc657e2aL, 0x5fcb6fab3ad6faecL,
12         0x6c44198c4a475817L};
13
14     private String originalMessage;
15     private String lengthMessage;
16     private String paddingMessage;
17     private String message;
18     private String[] messageBlock;
19     private String digest;
20
21     public Sha512() {
22         message = "";
23         lengthMessage = "";
24         paddingMessage = "";
25         originalMessage = "";
26         digest = "";
27     }
28
29     public void setMessage(String m) {
30         reset();
31         message = m;
32     }
33
34     public void reset() {
35         message = "";
36         lengthMessage = "";
37         paddingMessage = "";
38         originalMessage = "";
39         digest = "";
40     }
41
42     public String getDigest() {
43         String text = "";
44         for(int i = 0; i < digest.length(); i+=4) {
45             String bits = digest.substring(i, i+4);
46             text += Long.toHexString(longValue(bits));
47         }
48         return text;
49     }
50
51     public void createDigest() {
52         initialize(message);
53         String[] finalDigest = new String[INITIALS.length];
54         for(int countBlock = 0; countBlock < messageBlock.length; countBlock++) {
55             String block = messageBlock[countBlock];
56             String[] words = wordsExpansion(block);
57             String[] first = finalDigest;
58
59             //initialize first digest
60             if(countBlock == 0) {
61                 for(int i = 0; i < finalDigest.length; i++) {
62                     finalDigest[i] = pad(Long.toBinaryString(INITIALS[i]));
63                 }
64             }
65
66             //80 rounds
67             for(int i = 0; i < words.length; i++) {
68                 finalDigest = round(finalDigest, longValue(words[i]), i);
69             }
70
71             //final adding
72             for(int i = 0; i < finalDigest.length; i++) {
73                 finalDigest[i] = pad(Long.toBinaryString(longValue(finalDigest[i]) + longValue(
74                     first[i])));
75             }
76
77             for(int i = 0; i < finalDigest.length; i++) {
78                 digest += finalDigest[i] + " ";
79             }
80         }
81     }
82
83     private String[] wordsExpansion(String block) {
84         String[] words = new String[80];
85         for(int i = 0; i < 16; i++) {
86             int start = i * 64;
87             int end = start + 64;
88             words[i] = block.substring(start, end);
89         }
90         for(int i = 16; i < words.length; i++) {
91             //i-16
92             long w1 = longValue(words[i-16]);
93
94             //RotShift_1-8-7(i-15)
95             long w2 = rotShift(longValue(words[i-15]), 1, 8, 7);
96
97             //i-7
98             long w3 = longValue(words[i-7]);
99
100            //RotShift_19-61-6(i-2)
101            long w4 = rotShift(longValue(words[i-2]), 19, 61, 6);
102            long temp = w1 ^ w2 ^ w3 ^ w4;
103            words[i] = pad(Long.toBinaryString(temp));

```

```

1      }
2
3      return words;
4  }
5
6  //preDigest: previous digest or initial digest
7  //prevDigest[0] -> A    prevDigest[1] -> B    prevDigest[2] -> C    prevDigest[3] -> D
8  //prevDigest[4] -> E    prevDigest[5] -> F    prevDigest[6] -> G    prevDigest[7] -> H
9  private String[] round(String[] prevDigest, long word, int numOfRounds) {
10      String[] result = new String[8];
11      result[1] = pad(prevDigest[0]); //A -> B
12      result[2] = pad(prevDigest[1]); //B -> C
13      result[3] = pad(prevDigest[2]); //C -> D
14      result[5] = pad(prevDigest[4]); //E -> F
15      result[6] = pad(prevDigest[5]); //F -> G
16      result[7] = pad(prevDigest[6]); //G -> H
17
18      //compute Y -> E (prevDigest[4])
19      //conditional(E,F,G)
20      long cond = conditional(longValue(prevDigest[4]), longValue(prevDigest[5]), longValue(
21          prevDigest[6]));
22      //rotate(E)
23      long rot_E = rotate(longValue(prevDigest[4]));
24      //addition modulo 2^64
25      //temp1 = H + cond(E,F,G) + rotate(E) + w_i + K_i
26      long temp1 = longValue(prevDigest[7]) + cond + rot_E + word + CONSTANTS[numOfRounds];
27      //Y = temp1 + D
28      long Y = temp1 + longValue(prevDigest[3]);
29      //Y -> E
30      result[4] = pad(Long.toBinaryString(Y));
31
32      //compute X -> A (prevDigest[0])
33      //majority(A,B,C)
34      long maj = majority(longValue(prevDigest[0]), longValue(prevDigest[1]), longValue(prevDigest
35          [2]));
36      //rotate(A)
37      long rot_A = rotate(longValue(prevDigest[0]));
38      //temp2 = majority(A,B,C) + rotate(A)
39      long temp2 = maj + rot_A;
40      //X = temp1 + temp2
41      long X = temp1 + temp2;
42      //X -> A
43      result[0] = pad(Long.toBinaryString(X));
44
45      return result;
46  }
47
48  /*WORD EXPANSION FUNCTIONS*/
49  //RotShift_l-m-n(x) = RotR_l(x) XOR RotR_m(x) XOR ShL_n(x)
50  private long rotShift(long x, long l, long m, long n) {
51      long res = rotR(x, l) ^ rotR(x, m) ^ shL(x, n);
52      return x;
53  }
54  //Rotate right x by n bits
55  private long rotR(long x, long n) {
56      return (((x) >> n) | ((x) << (64 - n)));
57  }
58  //Shift left x by n bits
59  private long shL(long x, long n) {
60      return x << n;
61  }
62  /*END*/
63
64  /*ROUND FUNCTIONS*/
65  //MAJORITY(x,y,z) = (x AND y) XOR (y AND z) XOR (z AND x)
66  private long majority(long x, long y, long z) {
67      return ((x & y) ^ (y & z) ^ (z & x));
68  }
69  //CONDITIONAL(x,y,z) = (x AND y) XOR (NOT x AND z)
70  private long conditional(long x, long y, long z) {
71      return ((x & y) ^ ((~x) & z));
72  }
73  //ROTATE(x) = RotR_28(x) XOR RotR_34(x) XOR RotR_29(x)
74  private long rotate(long x) {
75      long res = rotR(x, 28) ^ rotR(x, 34) ^ rotR(x, 29);
76      return res;
77  }
78  /*END*/
79
80  private long longValue(String s) {
81      return new BigInteger(s, 2).longValue();
82  }
83
84  private void initialize(String msg) {
85      for (int i = 0; i < msg.length(); i++) {
86          originalMessage += Integer.toBinaryString(msg.charAt(i));
87      }
88
89      lengthMessage = Integer.toBinaryString(originalMessage.length());
90      if (lengthMessage.length() != 128) {
91          int addBit = 128 - lengthMessage.length();
92          String add = "";
93          for (int i = 0; i < addBit; i++) {
94              add += "0";
95          }
96          lengthMessage = add + lengthMessage;
97      }
98
99      paddingMessage = "1";
100     int paddingLength = (((-originalMessage.length() - 128) % 1024 + 1024) % 1024) - 1;
101     for (int i = 0; i < paddingLength; i++) {
102         paddingMessage += "0";
103     }

```

```

1      message = originalMessage + paddingMessage + lengthMessage;
2
3      messageBlock = new String[message.length()/1024];
4      for(int i = 0; i < messageBlock.length; i++) {
5          int start = i*1024;
6          int end = start+1024;
7          messageBlock[i] = message.substring(start, end);
8      }
9  }
10
11  private String pad(String input) {
12      String res = "";
13      int add = 0;
14      if(input.length() % 8 != 0) {
15          add = 8 - (input.length()%8);
16      }
17      for(int i = 0; i < add; i++) {
18          res += "0";
19      }
20      res += input;
21      return res;
22  }
23
24 }

```

Listing A.6: Function.java

```

25
26 package ShamirSecretSharing;
27
28 /**
29  *
30  * @author Samuel Christian
31  */
32 public class Function {
33
34     /**
35      * ATTRIBUTES
36      */
37     private int[] function;
38
39     /**
40      * CONSTRUCTOR
41      */
42     public Function(int[] func) {
43         function = func;
44     }
45
46     /**
47      * OTHERS
48      */
49     /**f(x) = res
50     public int countFunction(int x) {
51         int res = 0;
52         for(int i = 0; i < function.length; i++) {
53             res += function[i] * Math.pow(x,i);
54         }
55         return res;
56     }
57 }

```

Listing A.7: EquationSolver.java

```

58
59 package ShamirSecretSharing;
60
61 /**
62  *
63  * @author Samuel Christian
64  */
65 public class EquationSolver {
66
67     /**
68      * ATTRIBUTES
69      */
70     private double[][] equationMatrix;
71     private double[] solutionMatrix;
72
73     public EquationSolver(double[][] equation, double[] solution) {
74         equationMatrix = equation;
75         solutionMatrix = solution;
76     }
77
78     public void reset() {
79         equationMatrix = new double[1][1];
80         solutionMatrix = new double[1];
81     }
82
83     public double[] solve() {
84         int n = solutionMatrix.length;
85         for(int k = 0; k < n; k++) {
86             /**set pivot row*/
87             int max = k;
88             for(int row = k+1; row < n; row++) {
89                 if(Math.abs(equationMatrix[row][k]) > Math.abs(equationMatrix[max][k])) {
90                     max = row;
91                 }
92             }
93
94             /**swap with pivot row matrix A*/

```

```

1      double[] temp = equationMatrix[max];
2      equationMatrix[max] = equationMatrix[k];
3      equationMatrix[k] = temp;
4
5      /*swap solution matrix B*/
6      double tmp = solutionMatrix[max];
7      solutionMatrix[max] = solutionMatrix[k];
8      solutionMatrix[k] = tmp;
9
10     /*set matrix into triangle matrix*/
11     for (int row = k+1; row < n; row++) {
12         double factor = equationMatrix[row][k] / equationMatrix[k][k];
13         solutionMatrix[row] -= factor * solutionMatrix[k];
14         for (int col = k; col < n; col++) {
15             equationMatrix[row][col] -= factor * equationMatrix[k][col];
16         }
17     }
18 }
19
20 /*substitution to find solution*/
21 double[] solution = new double[n];
22
23 for (int row = n - 1; row >= 0; row--) {
24     double sum = 0.0;
25     for (int j = row + 1; j < n; j++) {
26         sum += equationMatrix[row][j] * solution[j];
27     }
28     solution[row] = (solutionMatrix[row] - sum) / equationMatrix[row][row];
29 }
30 return solution;
31 }
32 }

```

Listing A.8: SecretSharing.java

```

33
34 package ShamirSecretSharing;
35
36 import java.util.ArrayList;
37
38 /**
39  *
40  * @author Samuel Christian
41  */
42 public class SecretSharing {
43
44     /**
45      * ATTRIBUTES
46      */
47     private int[] function;
48     private int[] fx;
49     private int secret;
50     private EquationSolver solver;
51
52     /**
53      * CONSTRUCTOR
54      */
55     public SecretSharing(int s) {
56         secret = s;
57     }
58     public SecretSharing() {
59     }
60
61     /**
62      * ACCESSOR
63      */
64     public int[] getFunction() {
65         return function;
66     }
67     public int[] getFx() {
68         return fx;
69     }
70
71     /**
72      * OTHERS
73      */
74     public void split(int share, int k) {
75         function = new int[k];
76         function[0] = secret;
77         for (int i = 1; i < function.length; i++) {
78             function[i] = (int)(Math.random()*50) + 1;
79         }
80
81         Function fcount = new Function(function);
82         //f(0) is secret
83         fx = new int[share+1];
84         for (int i = 0; i <= share; i++) {
85             fx[i] = fcount.countFunction(i);
86         }
87     }
88
89     public String reconstruct(int n, int k, ArrayList<double[]> parts) {
90         ArrayList<double[]> functions = new ArrayList<double[]>();
91         for (int a = 1; a <= n; a++) {
92             double[] f = new double[k];
93             for (int x = 0; x < f.length; x++) {
94                 f[x] = Math.pow(a, x);
95             }
96             functions.add(f);
97         }
98
99         double[][] equation = new double[k][k];

```



```

1      int idx = 0;
2      double[] tempPasswordPart = parts.get(0);
3      for(int i = 0; i < tempPasswordPart.length; i++) {
4          if(tempPasswordPart[i] != -1 && idx < equation.length) {
5              equation[idx] = functions.get(i);
6              idx++;
7          }
8      }
9
10     int[] finalPart = new int[parts.size()];
11     for(int i = 0; i < parts.size(); i++) {
12         double[] temp = parts.get(i);
13         double[] solution = new double[k];
14         double[][] tempEq = new double[k][k];
15         for(int row = 0; row < equation.length; row++) {
16             for(int col = 0; col < equation[row].length; col++) {
17                 tempEq[row][col] = equation[row][col];
18             }
19         }
20
21         idx = 0;
22         for(int c = 0; c < temp.length; c++) {
23             if(temp[c] != -1 && idx < solution.length) {
24                 solution[idx] = temp[c];
25                 idx++;
26             }
27         }
28         solver = new EquationSolver(tempEq, solution);
29         finalPart[i] = (int)solver.solve()[0];
30     }
31
32     String forgottenPassword = "";
33     for(int i = 0; i < finalPart.length; i++) {
34         if(finalPart[i] != 0) {
35             char ch = (char)finalPart[i];
36             forgottenPassword += ch + " ";
37         }
38     }
39     return forgottenPassword;
40 }
41 }

```

Listing A.9: index.jsp

```

42  <!--
43      Document : index
44      Author   : Sam
45  -->
46
47  <%@page contentType="text/html" pageEncoding="windows-1252"%>
48  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
49      "http://www.w3.org/TR/html4/loose.dtd">
50
51  <html>
52  <head>
53      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
54      <title>Secret Sharing</title>
55      <link href="css/bootstrap.min.css" rel="stylesheet">
56      <link href="css/normalize.css" rel="stylesheet">
57      <link href="css/font-awesome.css" rel="stylesheet">
58      <link href="css/style.css" rel="stylesheet">
59      <meta http-equiv="X-UA-Compatible" content="IE=edge">
60      <meta name="viewport" content="width=device-width, initial-scale=1">
61  </head>
62  <body>
63      <div class="container_main-container">
64          <div class="main-menu-links">
65              <a href="pages/retrieve.jsp" class="btn btn-success">Retrieve Password</a>
66              or
67              <a href="pages/store.jsp" class="btn btn-info">Store Password</a>
68          </div>
69      </div>
70  </body>
71 </html>
72

```

Listing A.10: store.jsp

```

73  <!--
74      Document : store
75      Author   : Sam
76  -->
77
78  <%@page import="ReaderWriter.DataReader" %>
79  <%@page contentType="text/html" pageEncoding="windows-1252"%>
80  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
81      "http://www.w3.org/TR/html4/loose.dtd">
82
83  <html>
84  <head>
85      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
86      <title>Secret Sharing</title>
87      <link href="css/bootstrap.min.css" rel="stylesheet">
88      <link href="css/normalize.css" rel="stylesheet">
89      <link href="css/font-awesome.css" rel="stylesheet">
90      <link href="css/style.css" rel="stylesheet">
91      <meta http-equiv="X-UA-Compatible" content="IE=edge">
92      <meta name="viewport" content="width=device-width, initial-scale=1">
93  </head>
94

```

```

1 <body>
2 <div class="container">
3 <div class="login-box col-md-10 col-md-offset-1 col-sm-10 col-sm-offset-1">
4 <div class="panel panel-info" style="margin-top: 20px;">
5 <div class="panel-heading">
6 <div class="panel-title">Store Password</div>
7 </div>
8 <form method="POST" action="process.jsp" onsubmit="return_count()">
9 <div class="panel-body">
10 <input type="button" style="margin-left: 12px;" class="btn btn-primary
11 " id="addPassword" value="Add Password"/>
12 <hr class="pswd-hr"/>
13 <div class="form-group">
14 <label style="margin-top: 7px;" class="col-sm-7 control-label">
15 Security Questions</label>
16 </div>
17 <div style="display: none;" id="counter">1</div>
18 <div class="form-group">
19 <div class="col-sm-9"><input type="text" id="question" class="form
20 -control" placeholder="Question Here"/></div>
21 <div class="col-sm-2"><input type="button" id="add" class="btn btn
22 -primary" value="Add"/></div>
23 </div>
24 <hr class="submit-buttons-hr"/>
25 <div class="form-group">
26 <input type="submit" class="btn btn-info" value="Store"/>
27 <a class="btn btn-danger" href=" ../index.jsp">Cancel</a>
28 </div>
29 </form>
30 </div>
31 </div>
32 </div>
33 <script src=" ../js/jquery-2.1.1.js" type="text/javascript"></script>
34 <script src=" ../js/bootstrap.min.js" type="text/javascript"></script>
35 <script src=" ../js/app.js" type="text/javascript"></script>
36 </body>
37 </html>

```

Listing A.11: process.jsp

```

39 <!--
40 Document : process
41 Author : Sam
42 --%>
43
44 <%@page import="java.util.Arrays"%>
45 <%@page import="ReaderWriter,DataWriter"%>
46 <%@page import="ReaderWriter,DataReader"%>
47 <%@page import="SHA512.Sha512"%>
48 <%@page import="DES.DESEncryption"%>
49 <%@page import="ShamirSecretSharing.SecretSharing"%>
50 <%@page contentType="text/html" pageEncoding="windows-1252"%>
51 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
52 "http://www.w3.org/TR/html4/loose.dtd">
53
54 <html>
55 <head>
56 <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
57 <title>Secret Sharing</title>
58 </head>
59 <body>
60 <%
61 String[] passwords = request.getParameterValues("password");
62 String[] questions = request.getParameterValues("questions");
63 String[] answers = request.getParameterValues("answer");
64 String path = getServletContext().getRealPath("data");
65
66 int n = questions.length;
67 int k = passwords.length/2+1;
68 if(k <= 3) {
69 k = 4;
70 }
71
72 int salt = (int)(Math.random()*50) + 10;
73
74 String[] hashValue = new String[answers.length];
75 Sha512 sha = new Sha512();
76 for(int i = 0; i < hashValue.length; i++) {
77 String message = questions[i] + answers[i] + salt;
78 sha.setMessage(message);
79 sha.createDigest();
80 hashValue[i] = sha.getDigest();
81 sha.reset();
82 }
83
84 DataWriter dw = null;
85 String writeToFile = "";
86 DESEncryption e = new DESEncryption();
87 for(int count = 0; count < passwords.length; count++) {
88 String password = passwords[count];
89 String[] encrypted = new String[password.length()*n];
90
91 int secret = 0;
92 int enclidx = 0;
93 for(int i = 0; i < password.length(); i++) {
94 secret = (int)(password.charAt(i));
95 SecretSharing ss = new SecretSharing(secret);
96 ss.split(n, k);
97
98 int[] function = ss.getFunction();
99

```

```

1      int [] shares = ss.getFx();
2
3      for(int j = 1; j < shares.length; j++) {
4          e.setMessage(shares[j]+"");
5          e.setKey(hashValue[j-1]);
6          e.initialize();
7          e.encrypt();
8          encrypted[encIdx] = e.getCipherText();
9          encIdx++;
10         e.reset();
11     }
12
13
14     //save answers
15     writeToFile = "";
16     for(int i = 0; i < encrypted.length; i++) {
17         writeToFile += encrypted[i] + "\r\n";
18     }
19     dw = new DataWriter(path + "\\data_answers_" + count + ".txt");
20     dw.write(writeToFile);
21
22
23     //save questions
24     writeToFile = "";
25     for(int i = 0; i < questions.length; i++) {
26         writeToFile += questions[i] + "\r\n";
27     }
28     dw = new DataWriter(path + "\\data_questions.txt");
29     dw.write(writeToFile);
30
31     //save salt and min question
32     writeToFile = salt + "\r\n" + k;
33     dw = new DataWriter(path + "\\data_others.txt");
34     dw.write(writeToFile);
35
36     response.sendRedirect("../index.jsp");
37
38     %>
39 </body>
</html>

```

Listing A.12: retrieve.jsp

```

40
41 <!--
42 Document : retrieve
43 Author : Sam
44 --%>
45
46 <%@page import="ReaderWriter.DataReader"%>
47 <%@page contentType="text/html" pageEncoding="windows-1252"%>
48 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
49 "http://www.w3.org/TR/html4/loose.dtd">
50
51 <html>
52 <head>
53 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
54 <title>Secret Sharing</title>
55 <link href="../css/bootstrap.min.css" rel="stylesheet">
56 <link href="../css/normalize.css" rel="stylesheet">
57 <link href="../css/font-awesome.css" rel="stylesheet">
58 <link href="../css/style.css" rel="stylesheet">
59 <meta http-equiv="X-UA-Compatible" content="IE=edge">
60 <meta name="viewport" content="width=device-width, initial-scale=1">
61 </head>
62 <body>
63 <div class="container">
64 <div class="login-box col-md-10 col-md-offset-1 col-sm-10 col-sm-offset-1">
65 <div class="panel panel-info" style="margin-top:20px;">
66 <div class="panel-heading">
67 <div class="panel-title">Retrieve Password</div>
68 </div>
69 <form action="retrieve_process.jsp" method="POST">
70 <div class="panel-body">
71 <div class="form-group">
72 <label style="margin-top:7px;" class="col-sm-7 control-label">
73 Security Questions</label>
74 </div>
75 <%
76 String path = getServletContext().getRealPath("data");
77 DataReader dr = new DataReader(path + "\\data_questions.txt");
78 dr.read();
79 String[] questions = dr.get();
80 for(int i = 0; i < questions.length; i++) {
81
82 %>
83 <div class="form-group">
84 <label for="num" style="margin-top:7px;" class="col-sm-1_
85 control-label"><%= (i+1) + ". "%></label>
86 <label for="answer" style="margin-top:7px;" class="col-sm
87 -7_ control-label"><%= questions[i] %></label>
88 <div class="col-sm-4"><input id="ans" name="answer" type="
89 text" class="form-control"/></div>
90 <input type="hidden" name="questions" value="<%= _questions
91 [i] %>"/>
92 </div>
93 <%
94 }
95 %>
96 <hr/>
97 <div class="form-group">
98 <input type="submit" class="btn btn-info" value="Submit"/>
99 <a class="btn btn-danger" href="../index.jsp">Cancel</a>
100 </div>

```

```

1 |
2 |
3 |
4 |
5 |
6 |
7 |

```

Listing A.13: retrieve_process.jsp

```

8 |
9 |
10 |
11 |
12 |
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |
21 |
22 |
23 |
24 |
25 |
26 |
27 |
28 |
29 |
30 |
31 |
32 |
33 |
34 |
35 |
36 |
37 |
38 |
39 |
40 |
41 |
42 |
43 |
44 |
45 |
46 |
47 |
48 |
49 |
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |

```

```

<!--
Document : retrieve_process
Author : Sam
-->

<%@page import="ShamirSecretSharing.SecretSharing"%>
<%@page import="ShamirSecretSharing.EquationSolver"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.io.FileNameFilter"%>
<%@page import="java.io.File"%>
<%@page import="java.util.Arrays"%>
<%@page import="DES.DESDecryption"%>
<%@page import="SHA512.Sha512"%>
<%@page import="ReaderWriter.DataReader"%>
<%@page contentType="text/html" pageEncoding="windows-1252"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Secret Sharing</title>
<link href="../css/bootstrap.min.css" rel="stylesheet">
<link href="../css/normalize.css" rel="stylesheet">
<link href="../css/font-awesome.css" rel="stylesheet">
<link href="../css/style.css" rel="stylesheet">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
<div class="container">
<div class="login-box col-md-8 col-md-offset-2 col-sm-8 col-sm-offset-2">
<div class="panel panel-info" style="margin-top: 20px;">
<div class="panel-heading">
<div class="panel-title">Retrieve Password</div>
</div>
<div class="panel-body">
<%
String path = getServletContext().getRealPath("data");
File folder = new File(path);
File[] listPassword = folder.listFiles(new FileNameFilter() {
@Override
public boolean accept(File dir, String name) {
return (name.contains("data_answers") && name.endsWith(".txt"))
&& (!name.contains("case"));
}
});

String[] answers = request.getParameterValues("answer");
String[] questions = request.getParameterValues("questions");
int n = answers.length;

DataReader dr = new DataReader(path + "\\data_others.txt");
dr.read();
int salt = Integer.parseInt(dr.get()[0]);
int k = Integer.parseInt(dr.get()[1]);

String[] hashValue = new String[answers.length];
Sha512 sha = new Sha512();
for(int i = 0; i < answers.length; i++) {
String msg = questions[i] + answers[i] + salt;
sha.setMessage(msg);
sha.createDigest();
hashValue[i] = sha.getDigest();
sha.reset();
}

int[] decryptedAnswers = new int[1];
ArrayList<double[]> passwordPart = new ArrayList<double[]>();
SecretSharing ss = new SecretSharing();
String[] passwords = new String[listPassword.length];
for(int passCount = 0; passCount < listPassword.length; passCount++) {
passwordPart = new ArrayList<double[]>();
dr = new DataReader(listPassword[passCount].getAbsolutePath());
dr.read();
String[] encryptedAnswers = dr.get();
decryptedAnswers = new int[encryptedAnswers.length];

DESDecryption d = new DESDecryption();
int count = 0;
for(int j = 0; j < encryptedAnswers.length; j++) {
d.setCipher(encryptedAnswers[j]);
d.setKey(hashValue[count]);
try {
decryptedAnswers[j] = Integer.parseInt(d.binToStr(d.
decrypt()).trim());
} catch (NumberFormatException nfe) {
decryptedAnswers[j] = -1;
}
}
d.reset();
if(count == hashValue.length-1) {

```

```

1         count = 0;
2     } else {
3         count++;
4     }
5 }
6
7     for(int j = 0; j < decryptedAnswers.length; ) {
8         double[] temp = new double[n];
9         for(int idx = 0; idx < n; idx++) {
10             temp[idx] = decryptedAnswers[j];
11             j++;
12         }
13         passwordPart.add(temp);
14     }
15
16     passwords[passCount] = ss.reconstruct(n, k, passwordPart);
17 }
18
19
20 for(int i = 0; i < passwords.length; i++) {
21
22     <% if(passwords[i] == "") { %>
23         <div class="form-group">
24             <div class="failed-notification">Password cannot be retrieved<
25             /div>
26         </div>
27         <% i = passwords.length; %>
28     } else { %>
29         <div class="form-group">
30             <label class="col-sm-4" style="margin-top:7px;">Your password
31             is</label>
32             <div class="col-sm-6"><input class="form-control" type="text"
33             value="<%=passwords[i]_%" disabled/></div>
34         </div>
35     } %>
36 } %>
37 <div class="form-group">
38     <div class="col-sm-4">
39         <a href=" ../index.jsp" class="btn btn-primary">Done</a>
40     </div>
41 </div>
42 </div>
43 </div>
44 </div>
45 </div>
46 </body>
47 </html>

```

Listing A.14: app.js

```

48
49 jQuery(document).ready(function() {
50     jQuery("#addQuestions").click(function() {
51         var obj = jQuery("#selectQuestions_option:selected");
52         var num = jQuery("#counter").text();
53         if(jQuery("#selectQuestions").has('option').length > 0) {
54             var element = "<div class='form-group'>" + "<label style='margin-top:7px; ' class='col-
55             sm-1'>" + num + "</label>"
56             + "<label style='margin-top:7px; ' class='col-sm-7'>" + obj.text() + "</label>" +
57             "<div class='col-sm-4'><input id='quest' _name='answer' _type='text' _class='form-
58             control_questions' /></div>"
59             + "</div><input _name='questions' _type='hidden' _value='" + obj.text() + "' />";
60             jQuery(".submit-buttons-hr").before(element);
61         }
62         jQuery("#selectQuestions_option:selected").remove();
63         num++;
64         jQuery("#counter").text(num);
65     });
66
67     jQuery("#add").click(function() {
68         var question = jQuery("#question").val();
69         var num = jQuery("#counter").text();
70         if(question == "") {
71             alert("Empty");
72         } else {
73             var element = "<div class='form-group'>"
74             + "<label style='margin-top:7px; ' class='col-sm-1'>" + num + "</label>"
75             + "<label style='margin-top:7px; ' class='col-sm-6'>" + question + "</label>"
76             + "<div class='col-sm-4'><input id='quest' _name='answer' _type='text' _class='form-
77             control_questions' /></div>"
78             + "<a class='btn btn-danger' _onclick='removeDiv(this)' _style='margin-top:3px;'>"
79             + "<span class='glyphicon glyphicon-remove-sign' _aria-hidden='true'></span>"
80             + "</a>"
81             + "<input _name='questions' _type='hidden' _value='" + question + "' />"
82             + "</div>";
83             jQuery(".submit-buttons-hr").before(element);
84             num++;
85             jQuery("#counter").text(num);
86         }
87     });
88
89     jQuery("#addPassword").click(function() {
90         var element = "<div class='form-group' _style='margin-top:10px;'>"
91         + "<label for='pswd' _style='margin-top:7px; ' class='col-sm-1 control-label'>Password</
92         label>"
93         + "<label style='margin-top:7px; ' class='col-sm-1'></label>"
94         + "<div class='col-sm-9'><input id='pswd' _name='password' _type='password' _class='form-
95         control' _required='required' /></div>"
96         + "<a class='btn btn-danger' _onclick='removeDiv(this)' _style='margin-top:3px;'>"
97         + "<span class='glyphicon glyphicon-remove-sign' _aria-hidden='true'></span>"
98         + "</a></div>";
99         jQuery(".pswd-hr").before(element);

```

```

1  });
2  });
3
4  function check() {
5      var share = jQuery(".questions").length;
6      var k = +jQuery(".k-value").val();
7      if(share < k) {
8          return false;
9      } else {
10         return true;
11     }
12 }
13
14 function removeDiv(e) {
15     jQuery(e).closest('div').remove();
16 }
17
18 function count() {
19     var password = jQuery("input#pswd").length;
20     if(password <= 0) {
21         alert("Minimal 1 password");
22         return false;
23     }
24     return true;
25 }

```

Listing A.15: style.css

```

26
27 .main-container {
28     margin: 0px auto;
29     height: 500px;
30 }
31 .main-menu-links {
32     margin: 0px auto;
33     margin-top: 45vh;
34     height: 200px;
35     width: 600px;
36     text-align: center;
37 }
38 tr,td {
39     padding: 10px;
40 }
41
42 .password-input {
43     width: 350px;
44 }
45
46 label {
47     cursor: pointer !important;
48 }
49
50 .form-group {
51     margin-bottom: 10px !important;
52     height: 40px;
53 }
54 select.form-control {
55     width: 90px;
56     margin-bottom: 10px !important;
57 }
58
59 .failed-notification {
60     background-color: #D9534F;
61     color: #FFF;
62     font-weight: bold;
63     padding: 10px;
64     border-radius: 5px;
65     margin-left: 10px;
66     margin-right: 10px;
67 }

```