

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

**PASSWORD REMINDER ANALYSIS WITH SHAMIR'S
SECRET SHARING**



SAMUEL CHRISTIAN

NPM: 2011730002

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

LEMBAR PENGESAHAN

ANALISA METODE PENGINGAT *PASSWORD* DENGAN
SECRET SHARING SHAMIR

SAMUEL CHRISTIAN

NPM: 2011730002

Bandung, 12 Juni 2015

Menyetujui,

Pembimbing Tunggal

Thomas Anung Basuki, Ph.D.

Ketua Tim Penguji

Anggota Tim Penguji

Joanna Helga, M.Sc.

Pascal Alfadian, M.Com.

Mengetahui,

Ketua Program Studi

Thomas Anung Basuki, Ph.D.

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

ANALISA METODE PENGINGAT *PASSWORD* DENGAN *SECRET SHARING* SHAMIR

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 12 Juni 2015

Meterai

Samuel Christian
NPM: 2011730002

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*. 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.

Pada penelitian ini, dikembangkan mekanisme untuk mengembalikan n *password* dengan menyediakan n pertanyaan keamanan. Pengguna sistem hanya perlu menjawab sebagian dari n pertanyaan keamanan untuk mengembalikan n *password*. 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 lebih baik dalam melindungi *password*, maka dilakukan pembangunan perangkat lunak yang mengimplementasikan *secret sharing* Shamir dan pengujian terhadap perangkat lunak yang dibangun. Pengujian dengan metode survei juga dilakukan 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 dikombinasikan dengan enkripsi dan fungsi *hash*. Teknik enkripsi yang digunakan adalah *Data Encryption Standard* dan algoritma fungsi *hash* yang 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, Enkripsi, Fungsi *Hash*

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. 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.

In this research, a new mechanism is developed to retrieve n passwords by creating n security questions. This mechanism uses secret sharing Shamir to divide the each password into shares and creating n security questions for n passwords.

To find out if this developed mechanism that retrieve n passwords by providing n security questions using secret sharing Shamir performs better in protecting passwords, we develop a software which implementing secret sharing Shamir. We also have some survey tests 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 is combined with encryption technique and cryptographic hash function. Data Encryption Standard is used as encryption technique and Secure Hashing Algorithm 512 is 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, Encryption, Cryptographic Hash Function

Dipersembahkan untuk diri sendiri

KATA PENGANTAR

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Bandung, Juni 2015

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
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	8
2.3.4 Pembangunan Kunci Putaran	8
2.3.5 Putaran	10
2.3.6 Permutasi Akhir	15
2.4 Fungsi <i>Hash</i>	16
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	18
2.5.4 Fungsi Kompresi dan Putaran	19
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	Analisis Proses	33
3.1.1	Proses Penyimpanan <i>Password</i>	33
3.1.2	Proses Rekonstruksi <i>Password</i>	34
3.2	Studi Kasus	35
3.2.1	Pengenalan Kasus	35
3.2.2	Proses Penyimpanan <i>Password</i>	36
3.2.3	Proses Pengembalian <i>Password</i>	40
3.3	Pemilihan Nilai n dan k	45
3.3.1	Pemilihan Nilai k	45
3.3.2	Pemilihan Nilai n	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>	55
4.2.3	Kelas <i>EquationSolver</i>	56
4.2.4	Kelas <i>SecretSharing</i>	56
4.2.5	Kelas <i>DataEncryptionStandard</i>	58
4.2.6	Kelas <i>Encryption</i>	60
4.2.7	Kelas <i>Decryption</i>	61
4.2.8	Kelas <i>DataReader</i>	62
4.2.9	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	76
5.2.4	Analisis dan Hasil Pengujian Survei	77
5.2.5	Kesimpulan Pengujian	81
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 Pembangunan Kunci Putaran	9
2.3	Putaran dalam DES	11
2.4	Jaringan Feistel	12
2.5	Struktur Putaran dalam SHA-512	19
2.6	Masukan dan Keluaran dalam 1 Putaran SHA-512	20
2.7	Fungsi Khusus dalam 1 Putaran SHA-512	20
2.8	Proses Keseluruhan dari SHA-512	22
2.9	<i>Username</i> dan <i>Password</i>	23
2.10	<i>Password hashing</i>	24
2.11	<i>Password salting</i>	25
3.1	Proses Penyimpanan <i>Password</i>	34
3.2	Proses Rekonstruksi <i>Password</i>	35
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>	48
3.6	Rancangan Diagram Kelas	49
4.2	Kelas SHA512	51
4.1	Diagram Kelas Rinci	52
4.3	Kelas <i>Function</i>	55
4.4	Kelas <i>EquationSolver</i>	56
4.5	Kelas <i>DESEncryption</i>	57
4.6	Kelas <i>DataEncryptionStandard</i>	58
4.7	Kelas <i>Encryption</i>	60
4.8	Kelas <i>DESDecryption</i>	62
4.9	Kelas <i>DataReader</i>	62
4.10	Kelas <i>DataWriter</i>	63
4.11	Perancangan Tampilan Awal	64
4.12	Perancangan Tampilan Menyimpan <i>Password</i>	64
4.13	Perancangan Tampilan Mengembalikan <i>Password</i>	65
4.14	Perancangan Tampilan Mengembalikan <i>Password</i> (2)	65
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> (2)	70
5.7	Tampilan antarmuka untuk mengembalikan <i>password</i> (3)	71
5.8	Langkah menyimpan <i>password</i>	73
5.9	Tampilan Menjawab Pertanyaan Keamanan	74

5.10 Tampilan Menjawab Pertanyaan Keamanan Kasus Pertama	74
5.11 Hasil Pengujian Fungsional Kasus Pertama	75
5.12 Tampilan Menjawab Pertanyaan Keamanan Kasus Kedua	75
5.13 Hasil Pengujian Fungsional Kasus Kedua	76
5.14 Pengujian survei kasus 1	78
5.15 Pengujian survei kasus 2	79
5.16 Pengujian survei kasus 3	80
5.17 Pengujian survei kasus 4	81

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 $f(x)$	38
3.2	Hasil Enkripsi setiap <i>Share</i> untuk <i>Password</i> Pertama	39
3.3	Hasil Dekripsi <i>Share</i>	41
3.4	Tabel Pasangan Nilai n dan k	46
3.5	Skenario Menyimpan <i>Password</i>	47
3.6	Skenario Mengembalikan <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[1]. 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[1]. *Password* digunakan untuk mengakses sumber daya sebuah sistem. Saat pengguna sistem hendak mengakses sistem, pengguna harus 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 biasanya 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, dikembangkan mekanisme untuk mengembalikan n *password* dengan membuat n pertanyaan. Dengan menjawab sebagian pertanyaan, k pertanyaan, dari n pertanyaan, maka n *password* dapat dikembalikan. Mekanisme ini menggunakan metode *secret sharing*.

Secret sharing adalah metode membagi 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* yang membagi informasi menjadi beberapa *share* dan untuk mengembalikan informasi hanya dibutuhkan beberapa *share* saja.

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

Untuk menjaga kerahasiaan *password* dan jawaban dari setiap pertanyaan keamanan, metode *secret sharing* dikombinasikan dengan enkripsi dan fungsi *hash*. Pada penelitian ini, teknik enkripsi yang digunakan adalah *Data Encryption Standard* dan algoritma fungsi hash yang digunakan adalah *Secure Hashing Algorithm 512*.

Dalam penelitian ini, dibahas mengenai cara kerja metode *secret sharing* Shamir untuk mengembalikan *n password* dengan membuat *n* pertanyaan keamanan. Selain itu, dilakukan analisis dari kualitas pertanyaan keamanan yang dibuat dan pengaruhnya terhadap metode *secret sharing* Shamir dalam mengembalikan banyak *password*.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang sudah dibuat, maka permasalahan yang dibahas dalam penelitian ini adalah:

- Bagaimana mengembalikan banyak *password* dengan metode *secret sharing* Shamir?
- Bagaimana cara membangun perangkat lunak pengingat *password* yang mengimplementasikan metode *secret sharing* Shamir?
- Bagaimana menilai kualitas dari metode *secret sharing* Shamir lewat pertanyaan keamanan yang dibuat?

1.3 Tujuan

Berdasarkan rumusan masalah yang sudah ditetapkan, maka tujuan dari penelitian ini adalah:

- Mempelajari bagaimana metode *secret sharing* Shamir dapat mengembalikan banyak *password*.
- Membangun perangkat lunak pengingat *password* yang mengimplementasikan metode *secret sharing* Shamir.
- Melakukan pengujian terhadap perangkat lunak pengingat *password* yang dibangun.

1.4 Batasan Masalah

Batasan masalah pada penelitian ini adalah setiap pertanyaan keamanan dijawab dengan 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, *Data Encryption Standard*, *Secure Hashing Algorithm* 512, otentikasi, *password*, eliminasi Gauss-Jordan, *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 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 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[2].

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[3]:

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.

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

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

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

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)

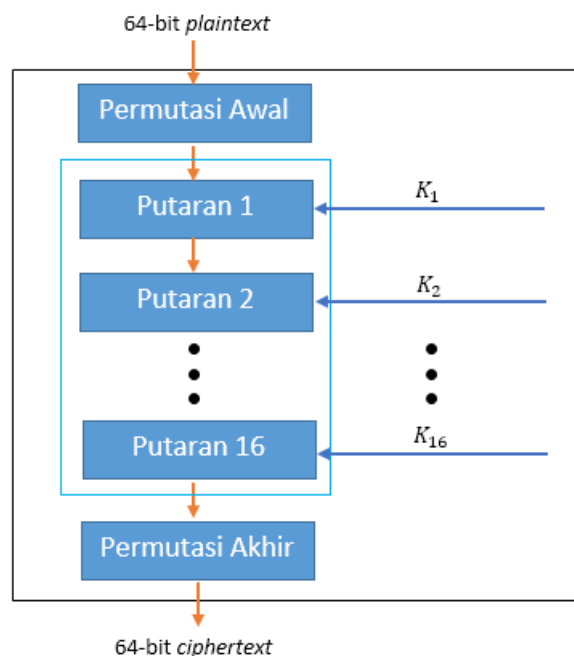
Pada bagian ini dijelaskan hal-hal mengenai *data encryption standard* dimulai dari sejarah *data encryption standard*, struktur *data encryption standard*, dan proses enkripsi *data encryption standard*.

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 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

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

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

Sebagai contoh, posisi ke-1 dari matriks mp menunjukkan angka 58. Maka, *bit* ke-58 dari masukan akan menjadi *bit* ke-1 dari keluaran. Contoh lainnya, posisi ke-62 dari matriks mp menunjukkan angka 23. Maka, *bit* ke-23 dari masukan akan menjadi *bit* ke-62 dari keluaran.

2.3.4 Pembangunan Kunci Putaran

DES menggunakan kunci dengan panjang 64-*bit*. Kunci ini perlu diubah menjadi kunci untuk setiap putaran DES dengan panjang masing-masing 48-*bit*. Proses pembangunan kunci putaran terdiri dari *parity drop*, *shift left*, dan permutasi *P-box*. Gambar 2.2 menunjukkan keseluruhan proses dari pembangunan kunci putaran. Pada bagian ini akan dijelaskan masing-masing proses dari pembangunan kunci putaran.

Parity Drop

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

Sebagai contoh, misalkan L dan R pada persamaan 2.4 dan 2.5.

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

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

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

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

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

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

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

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

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

Permutasi P -box

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

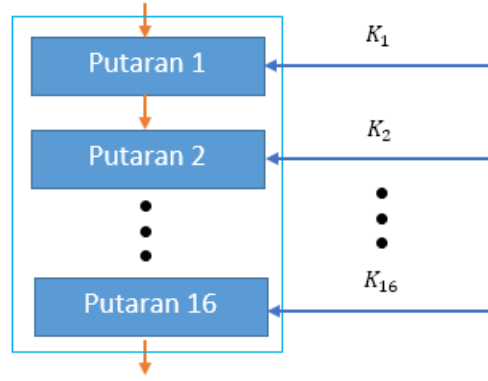
Tabel 2.3: Matriks kompresi P -box

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

Hasil keluaran dari proses ini adalah kunci putaran dengan panjang 48-bit dan siap 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 2.3 menunjukkan ilustrasi dari 16 putaran dari DES.



Gambar 2.3: 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.

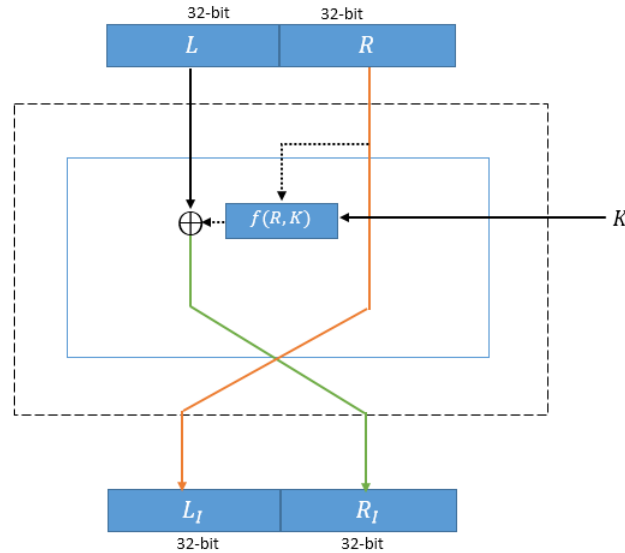
$$L_i = R_{i-1} \quad (2.10)$$

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 2.4 menunjukkan ilustrasi dari langkah-langkah yang sudah dijelaskan.



Gambar 2.4: Jaringan Feistel

Fungsi DES

Fungsi DES adalah fungsi f yang digunakan dalam jaringan Feistel pada Gambar 2.4. 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.

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.

Operasi XOR

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

Substitusi $S\text{-box}$

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

Proses substitusi terjadi sebagai berikut. Kombinasi *bit* ke-1 dan *bit* ke-6 pada blok 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, misalkan 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*.

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.

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

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

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

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

1. Preimage Resistance

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

2. Second Preimage Resistance

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

3. Collision Resistance

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

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

2.5 Secure Hashing Algorithm 512 (SHA-512)

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

2.5.1 Message Padding

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

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

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

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

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.

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

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, misalkan $i = 2$ 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, misalkan $i = 2$ dan $x = 1011$, maka hasil dari $ShL_2(1011)$ ditunjukkan pada persamaan 2.20.

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

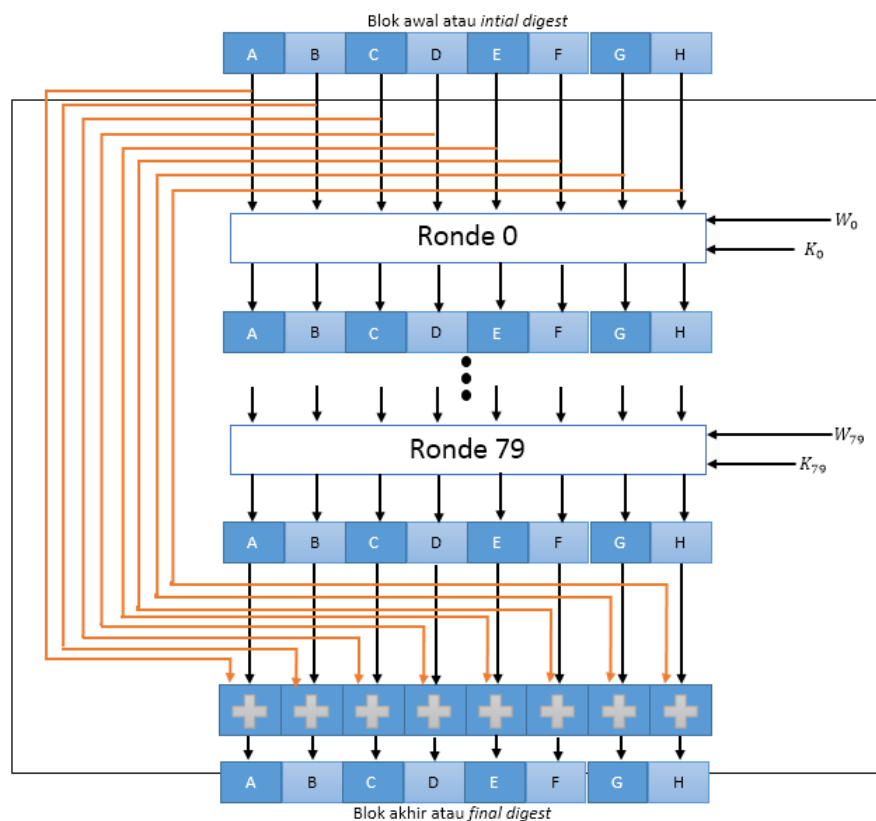
Setelah ekspansi blok *message* menjadi 80 *word* untuk setiap blok *message*, proses selanjutnya adalah putaran dari SHA-512. Proses putaran SHA-512 akan dijelaskan pada bagian selanjutnya.

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 terdiri dari 80 putaran SHA-512.

Struktur Putaran

Masukan dari putaran SHA-512 adalah blok dengan panjang 512-bit terdiri dari 8 *word* (A , B , C , D , E , F , G , dan H). Untuk putaran pertama, blok 512-bit diperoleh dari konstanta awal (A_0 sampai H_0) sedangkan untuk putaran kedua dan selanjutnya blok 512-bit diperoleh dari hasil dari putaran sebelumnya. Gambar 2.5 menunjukkan ilustrasi proses yang sudah dijelaskan.



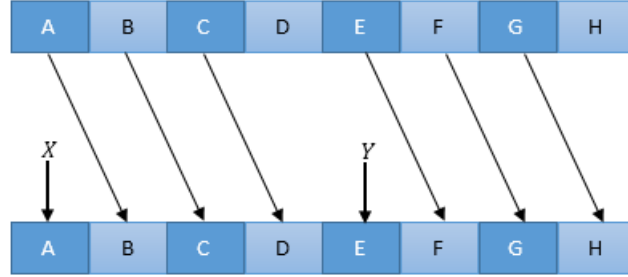
Gambar 2.5: Struktur Putaran dalam SHA-512

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

- *Word* keluaran B diperoleh dari *word* masukan A
- *Word* keluaran C diperoleh dari *word* masukan B
- *Word* keluaran D diperoleh dari *word* masukan C
- *Word* keluaran F diperoleh dari *word* masukan E
- *Word* keluaran G diperoleh dari *word* masukan F

- *Word* keluaran H diperoleh dari *word* masukan G

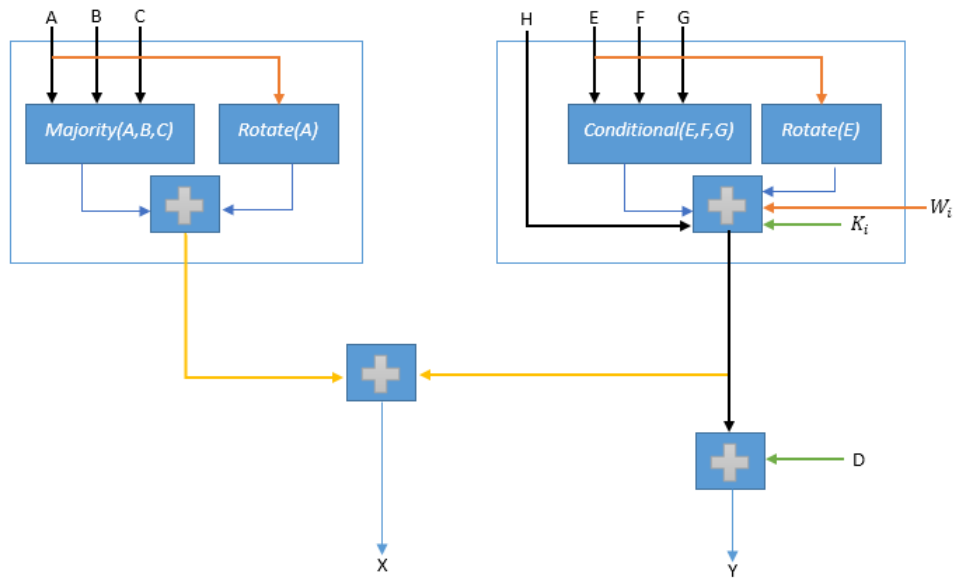
Gambar 2.6 menunjukkan ilustrasi dari masukan dan keluaran dalam 1 putaran SHA-512 untuk setiap *word*.



Gambar 2.6: Masukan dan Keluaran dalam 1 Putaran SHA-512

Untuk nilai *word* keluaran A dan E diperoleh dari *word* X dan Y . *Word* X dan Y ini diperoleh dari sebuah fungsi khusus. Gambar 2.7 menunjukkan struktur dari fungsi khusus. Berikut akan dijelaskan struktur dari fungsi khusus.

Struktur Fungsi Khusus



Gambar 2.7: Fungsi Khusus dalam 1 Putaran SHA-512

Word Y pada gambar 2.7 diperoleh dari proses persamaan 2.21.

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

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

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

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

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

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

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

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

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

Kemudian, *word* X pada Gambar 2.7 diperoleh dari persamaan 2.25.

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

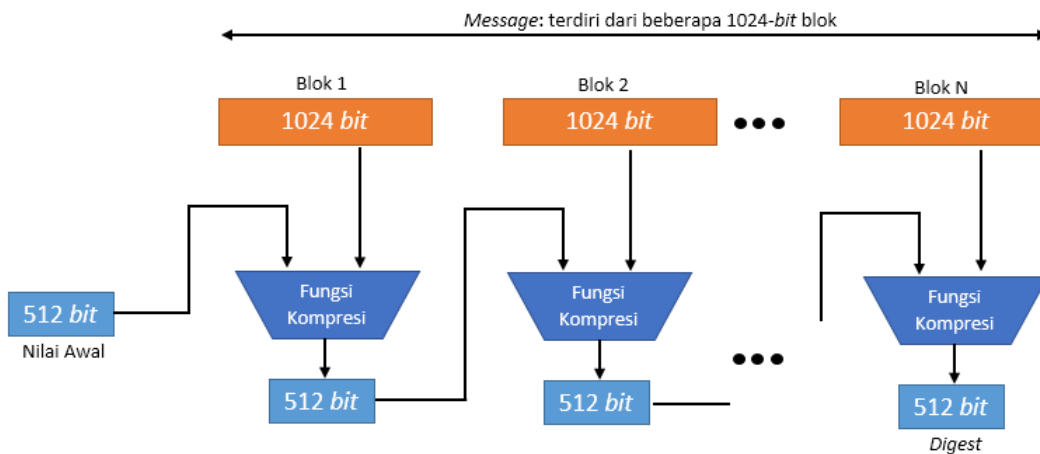
Untuk operasi *Conditional* dan *Rotate* sudah dijelaskan pada persamaan 2.23 dan 2.23. Sementara itu, untuk operasi *Majority* pada persamaan 2.25 adalah operasi *AND*, *OR* dan *XOR* dari *bit-bit* setiap *word*. Operasi *Majority* ditunjukkan pada persamaan 2.26.

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

Hasil akhir dari fungsi khusus adalah *word* X dan *word* Y . *Word* X akan menjadi *word* keluaran A dan *word* Y akan menjadi *word* keluaran E . Ilustrasi dari hasil keluaran ini ditunjukkan oleh Gambar 2.6.

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

Kemudian, hasil akhir dari proses tersebut berupa blok dengan panjang 512-bit terdiri dari 8 *word*. Blok 512-bit ini akan menjadi hasil akhir (*digest*) atau menjadi masukan untuk fungsi kompresi yang digunakan oleh blok *message* ke-2 dan seterusnya. Gambar 2.8 menunjukkan proses yang sudah dijelaskan.



Gambar 2.8: 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 [3], 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[3], 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[3], 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*.

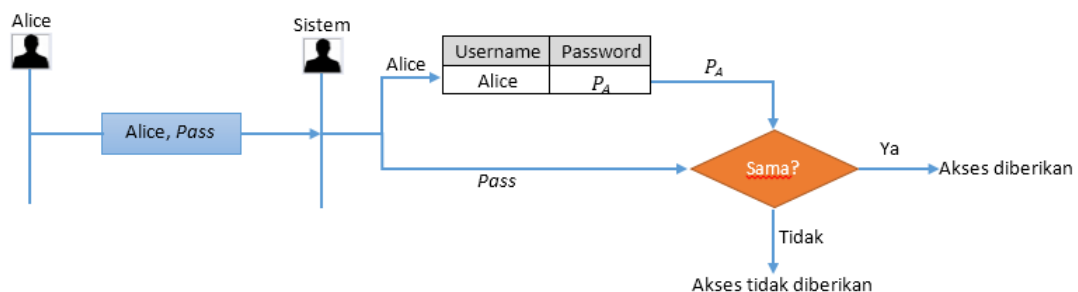
2. Password Tetap

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 memasukan *username* dan *password*.

Kemudian, saat pengguna sudah memasukan *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 2.9 menunjukkan proses yang dijelaskan.



Gambar 2.9: *Username* dan *Password*

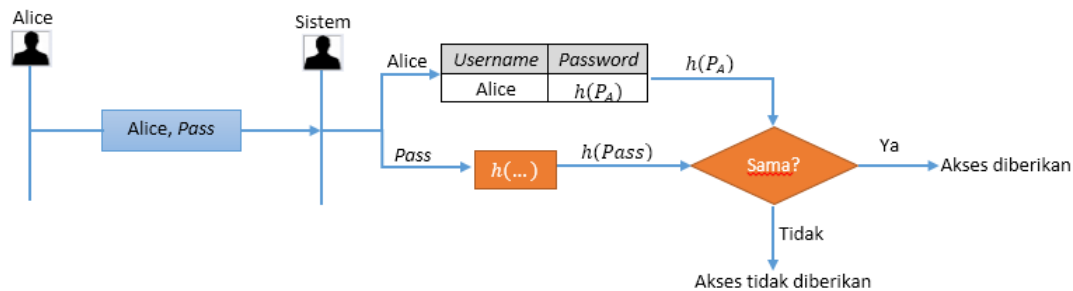
Kelebihan dari skema ini adalah skema ini mudah untuk diimplementasikan dan tidak membutuhkan proses yang rumit. Sementara itu, kekurangan dari skema ini adalah *password* yang

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 2.10 menunjukkan proses yang sudah dijelaskan.



Gambar 2.10: Password hashing

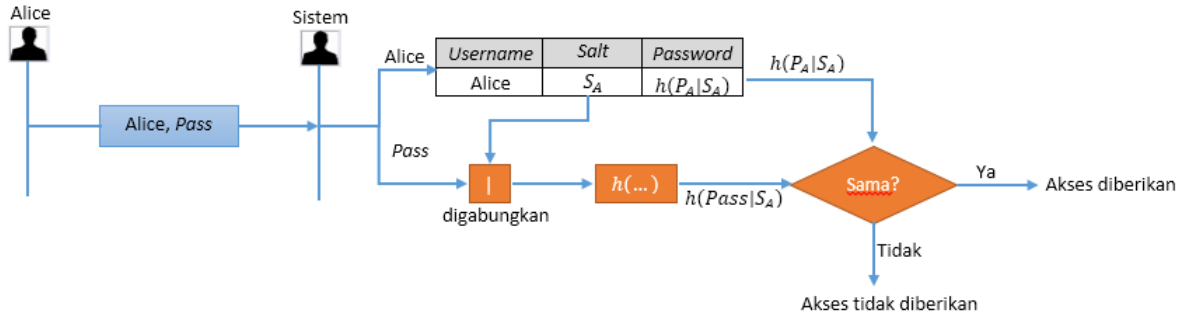
Kelebihan dari skema ini adalah walaupun *password* yang disimpan dalam tabel basis data diketahui oleh pihak yang tidak berhak, *password* tidak akan bisa dimengerti karena disimpan dalam bentuk *digest*nya. Sementara itu, *digest* tidak bisa dikembalikan ke dalam bentuk *plaintext* untuk mendapatkan *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.

Karena *salt* dibutuhkan untuk mengurangi tingkat keberhasilan *dictionary attack*, nilai *salt* akan disimpan juga dalam tabel basis data. Kemudian, saat pengguna sudah memasukkan *username* dan *password*, sistem akan menerima *password* yang dimasukan. Selanjutnya, *password* dikonkatenasi dengan *salt* yang disimpan lalu sistem akan menghitung *digest* dari hasil konkatenasi *password* dengan *salt*. Setelah itu, sistem akan membandingkan dengan *digest* yang disimpan dalam tabel basis data. Jika sesuai, pengguna akan diberikan hak akses ke sistem. Gambar 2.11 menunjukkan proses yang dijelaskan.



Gambar 2.11: *Password salting*

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

2.7 Eliminasi Gauss-Jordan

Eliminasi Gauss-Jordan adalah suatu metode untuk menyelesaikan sistem persamaan linear dengan mereduksi matriks menjadi eselon baris tereduksi[4]. Suatu matriks R dikatakan bentuk eselon baris tereduksi jika memenuhi syarat sebagai berikut[4].

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

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. Dimisalkan 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.

Langkah paling awal adalah mengubah persamaan 2.28 menjadi matriks. Konstanta dari variabel ke- i dari persamaan ke- j akan menjadi elemen matriks kolom ke- i dan baris ke- j . Sebagai contoh, variabel ke-1 dan persamaan ke-1 dari persamaan 2.28 adalah x dengan konstanta 1, maka elemen matriks kolom ke-1 baris ke-1 adalah 1. Contoh lainnya, variabel ke-2 dan persamaan ke-3 persamaan 2.28 adalah $3y$ dengan konstanta 3, maka elemen matriks kolom ke-2 baris ke-3 adalah 3. Hasil pengubahan persamaan 2.28 menjadi matriks 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.

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

Setelah operasi baris kedua, maka diperoleh Matriks 2.32 yang merupakan matriks dengan bentuk eselon baris tereduksi.

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

2.7.2 Proses Substitusi Balik

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

Sebagai contoh, dari Matriks 2.32 diperoleh hasilnya sebagai berikut.

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

Kemudian, untuk nilai y .

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

Kemudian, untuk nilai x .

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

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*'[5]. 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 dilambangkan dengan 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[5].

- 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. Dimisalkan 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.

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

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.

$$S = x_1, x_2, x_3, \dots, x_i \tag{2.41}$$

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

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

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[6]:

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. Dimisalkan bahwa probabilitas hasil keluaran setiap perulangan sukses adalah p . Sebaliknya, probabilitas hasil keluaran setiap perulangan gagal adalah $q = 1 - p$. Persamaan 2.43 untuk menghitung probabilitas x hasil keluaran yang sukses dari n percobaan.

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

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

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

2.10 Entropi

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

2.10.1 Sejarah Singkat

Istilah entropi muncul pertama kali dalam esai '*A Mathematical Theory of Communication*' pada tahun 1948. Esai ini dibuat oleh Claude E. Shannon seorang ilmuwan asal Amerika Serikat. Dalam esainya, Shannon menulis bahwa entropi adalah konsep keacakan atau suatu ketidakpastian[7]. Istilah dari entropi ini dinamakan Shannon *Entropy*.

2.10.2 Pembahasan

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

Nilai entropi yang tinggi menunjukkan bahwa informasi yang dimiliki sebuah pesan cukup tinggi. Nilai informasi yang cukup tinggi memiliki arti bahwa isi dari pesan bisa diprediksi. Sementara itu, jika nilai entropi yang rendah menunjukkan bahwa informasi yang dimiliki sebuah pesan cukup rendah. Nilai informasi yang cukup rendah memiliki arti bahwa isi dari pesan tidak bisa dengan mudah diprediksi.

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

Dari penjelasan mengenai entropi yang sudah dijelaskan, dimisalkan probabilitas kemunculan informasi x_i dalam sebuah pesan X adalah p_i . Maka, nilai entropi x_i adalah $H(x_i)$ berdasarkan probabilitas kemunculan informasi x_i yang dilambangkan oleh p_i . Persamaan 2.44 menunjukkan perhitungan dari $H(x_i)$.

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

Maka, nilai entropi pesan X untuk setiap informasi p_1, p_2, \dots, p_m ditunjukkan oleh persamaan 2.45.

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

BAB 3

ANALISIS

Pada bab ini berisi analisis terhadap teori-teori yang telah dibahas sebelumnya. Analisis akan meliputi analisis proses, studi kasus untuk penerapan metode *secret sharing* Shamir, pemilihan nilai n dan k , dan diagram-diagram mencakup diagram *use case*, diagram aktivitas, dan rancangan diagram kelas.

3.1 Analisis Proses

Pada bagian ini dijelaskan mengenai perancangan perangkat lunak yang dibangun berdasarkan pada proses-proses yang sudah dipaparkan pada bagian sebelumnya. Pada bagian sebelumnya sudah dibahas mengenai proses penyimpanan *password* dan proses rekonstruksi *password*. Pada bagian ini dibahas setiap tahapan dari proses-proses tersebut dan setiap tahapan digambarkan dalam bentuk alur proses (*flowchart*) sebelum dilakukan proses pembuatan diagram-diagram untuk membangun perangkat lunak.

3.1.1 Proses Penyimpanan *Password*

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

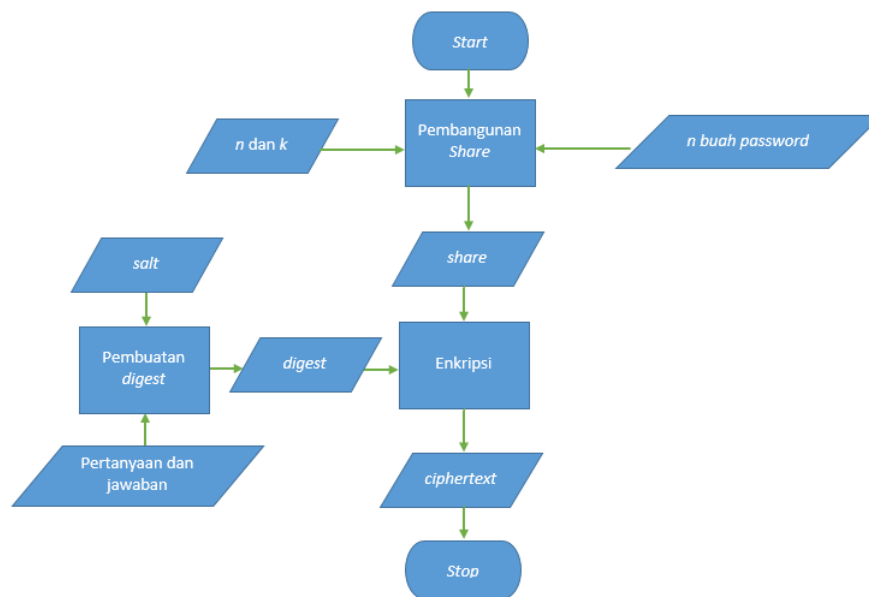
Tahapan-tahapan pada proses yang ditunjukkan oleh Gambar 3.1 adalah tahapan-tahapan untuk menyimpan *password*. Dalam penelitian ini, tahapan pembangunan *share* menggunakan metode *secret sharing* Shamir. Berikut penjelasan masing-masing tahapan:

- 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 3.3.

- 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* dikonkatenasi menjadi sebuah *string* yang akan dibuat *digest*nya. Dalam tahap ini, pembuatan *digest* menggunakan algoritma *SHA-512* (Subbab 2.5).



Gambar 3.1: Proses Penyimpanan *Password*

- Tahap Enkripsi

Pada tahap ini, *share* yang dihasilkan pada tahap pembangunan *share* dienkripsi 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.

3.1.2 Proses Rekonstruksi *Password*

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

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*:

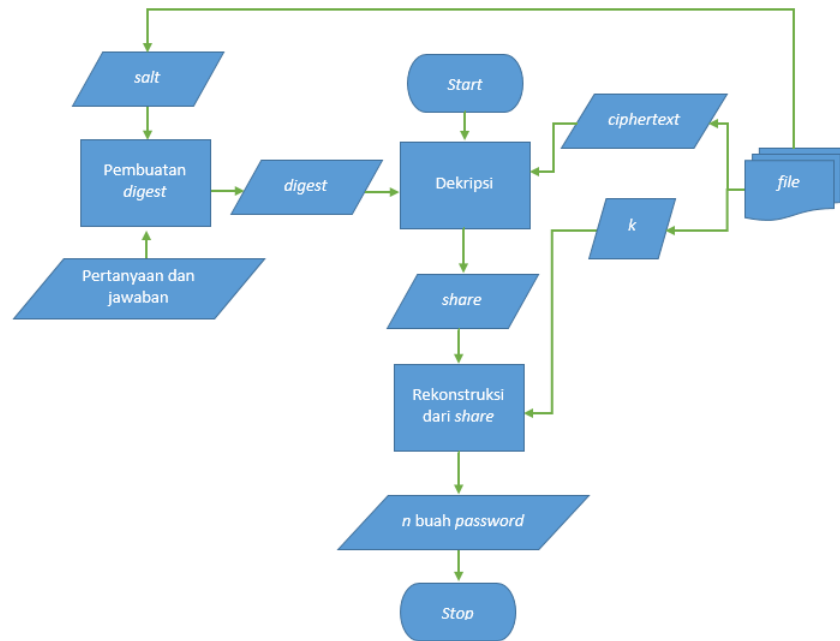
- Tahap Pembuatan *Digest*

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*

Gambar 3.2: Proses Rekonstruksi *Password*

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.

3.2 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*.

3.2.1 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, dimisalkan banyak *password* yang akan disimpan sebanyak 5 buah *password*. Setiap *password* akan diberi label p_1 , p_2 , sampai p_5 . Persamaan 3.1 sampai 3.5 menunjukkan p_1 sampai p_5 .

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

$$p_2 = password \quad (3.2)$$

$$p_3 = hello123 \quad (3.3)$$

$$p_4 = secret \quad (3.4)$$

$$p_5 = foobar \quad (3.5)$$

3.2.2 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.

Proses Pembangunan *Share*

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

1. Membagi setiap *password* p_i menjadi beberapa karakter, masing-masing karakter akan diubah menjadi nilai ASCII-nya, c_1, c_2, \dots, c_m .
2. Memilih nilai n , yaitu banyak *share* yang akan dibangun.
3. Memilih nilai k , yaitu banyak minimal pertanyaan keamanan yang harus dijawab dengan benar, dimana $0 < k \leq n$.
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.

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

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

Kembali kepada kasus pada Subbab 3.2.1, misalkan *password* yang akan dibangun *share-share*nya adalah p_1 . Langkah pertama adalah membagi p_1 menjadi beberapa karakter dan mengubah masing-masing karakter menjadi nilai ASCII-nya. Persamaan 3.7 sampai 3.13 menunjukkan langkah pertama.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Proses Enkripsi *Share*

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. Langkah-langkah proses enkripsi *share* adalah sebagai berikut.

1. Membuat n pertanyaan keamanan, q_1, q_2, \dots, q_n .
2. Menentukan jawaban dari masing-masing pertanyaan keamanan, a_1, a_2, \dots, a_n .
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 3.22 menunjukkan langkah enkripsi *share*.

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

Pada persamaan 3.22, m merupakan banyak karakter dari masing-masing *password* p_i .

Kembali kepada kasus pada Subbab 3.2.1, misalkan *password* yang akan dienkripsi *share-share*nya adalah p_1 . Langkah pertama adalah membuat n pertanyaan keamanan, karena $n = 5$ maka ada 5 pertanyaan keamanan. Setiap pertanyaan keamanan akan diberi label q_1, q_2, \dots, q_5 . Untuk kasus ini, dimisalkan pertanyaan keamanan yang dibuat adalah sebagai berikut.

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

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

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

Langkah selanjutnya adalah memilih nilai $salt$, r_s . Untuk kasus ini, misalkan $r_s = 31$.

Setelah memilih nilai $salt$, langkah selanjutnya adalah menghitung $digest$. Masing-masing dari pertanyaan keamanan akan dikonkatenasi dengan jawabannya dan r_s . Contoh hasil penghitungan $digest$, h_n , untuk setiap pertanyaan ditunjukkan pada persamaan 3.23 sampai 3.27.

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

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

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

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

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

Langkah selanjutnya adalah mengenkripsi setiap nilai $share$ yang sudah dibangun dengan $digest$ yang sudah dihitung sebagai kuncinya. Contoh hasil enkripsi setiap $share$ untuk p_1 , 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>

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

Langkah enkripsi setiap *share* ini dilakukan untuk setiap *password* p_2 sampai p_5 . Kemudian setelah proses enkripsi ini, pertanyaan keamanan, hasil enkripsi (*ciphertext*), nilai *salt*, dan nilai k akan disimpan.

3.2.3 Proses Pengembalian *Password*

Setelah *password* disimpan dalam proses Penyimpanan *Password* (Subbab 3.2.2), pada bagian ini akan dijelaskan proses bagaimana *password* bisa dikembalikan dengan menggunakan metode *secret sharing* Shamir. Proses pengembalian *password* ini dibagi menjadi 2 proses, yaitu proses dekripsi setiap *share* dan proses rekonstruksi kembali *password* dari *share-share* yang sudah didekripsi.

Proses Dekripsi *Share*

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

1. Menjawab n pertanyaan keamanan yang sebelumnya disimpan, q_1, q_2, \dots, q_n untuk 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.

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

Kembali kepada kasus yang dijelaskan pada Subbab 3.2.1, langkah pertama adalah menjawab pertanyaan keamanan yang sebelumnya disimpan. Berikut pertanyaan keamanan yang disimpan dan jawaban untuk masing-masing pertanyaan keamanan.

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

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

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

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

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

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

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

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

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

$$D_{h_1}(c_{11}) = s_{11} = 67 \quad (3.36)$$

Kemudian, proses dekripsi diulang untuk setiap *share* dari *password* p_1 . Tabel 3.3 menunjukkan 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 .

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 masing-masing karakter dari *password* p_i berdasarkan nilai k yang disimpan. Dalam kasus Subbab 3.2.1, k yang dipilih adalah $k = 3$, maka fungsi $f(x)$ yang dibentuk memiliki derajat $k - 1$. Persamaan 3.40 menunjukkan fungsi $f(x)$ yang dibentuk.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Kolom paling kanan dari Matriks 3.50 menunjukkan nilai $f_1(x)$, $f_2(x)$, dan $f_3(x)$, sedangkan kolom lainnya menunjukkan nilai koefesien dari setiap variabel dalam $f_1(x)$, $f_2(x)$, dan $f_3(x)$. Kemudian, setiap baris akan diberi label. Baris 1 diberi label L_1 , baris 2 diberi label L_2 , dan baris 3 diberi label L_3 .

Setelah transformasi matriks, langkah selanjutnya adalah operasi setiap baris untuk memperoleh matriks bentuk eselon baris tereduksi. Operasi pertama yang dilakukan ditunjukkan oleh persamaan 3.51.

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

Operasi pertama menghasilkan Matriks 3.52.

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

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

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

Operasi kedua menghasilkan Matriks 3.54.

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

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

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

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

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

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

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

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

Proses yang sama akan dilakukan untuk karakter kedua, ketiga, sampai karakter keenam. Setelah semua karakter diperoleh, setiap karakter akan dikonkatenasi menjadi sebuah *string*. Maka, hasil

akhir dari p_1 ditunjukkan pada persamaan 3.58.

$$p_1 = 123456 \quad (3.58)$$

3.3 Pemilihan Nilai n dan k

Pengguna dapat memilih n dan k sesuai dengan kebutuhan. Pemilihan n dan k yang baik, tidak hanya dapat membuat *password* tidak akan mudah dikembalikan oleh pihak yang tidak berhak, tetapi dapat juga membuat pengguna bisa dengan mudah mengembalikan *password*[8]. Pada bagian ini, akan dijelaskan bagaimana pemilihan n dan k dapat mempengaruhi kedua hal tersebut.

3.3.1 Pemilihan Nilai k

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

Maka dengan bertambahnya nilai k dan diasumsikan e_i dari setiap pertanyaan sangat kecil, maka kemungkinan jawaban dari setiap pertanyaan akan bervariasi sehingga jawaban dari masing-masing pertanyaan keamanan tidak bisa dengan mudah ditebak atau diprediksi. Namun, nilai k yang terlalu besar juga akan menyulitkan pemilik *password* untuk mengembalikan *password* karena semakin banyak pertanyaan yang harus dijawab dengan tepat.

3.3.2 Pemilihan Nilai n

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

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

Maka, dari persamaan 3.59 untuk n pertanyaan, probabilitas dari P_i akan bertambah besar dengan bertambah besarnya nilai k . Melalui hal tersebut, dapat disimpulkan probabilitas pengguna akan berhasil mengembalikan *password*:

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

P_2 akan mempengaruhi pasangan n dan k yang harus dipilih. Jika dimisalkan $P_0 = 0.95$ dan $P_2 = 0.99998$, maka pasangan n dan k yang ideal dari hasil penghitungan menggunakan persamaan 3.60:

Tabel 3.4: Tabel Pasangan Nilai n dan k

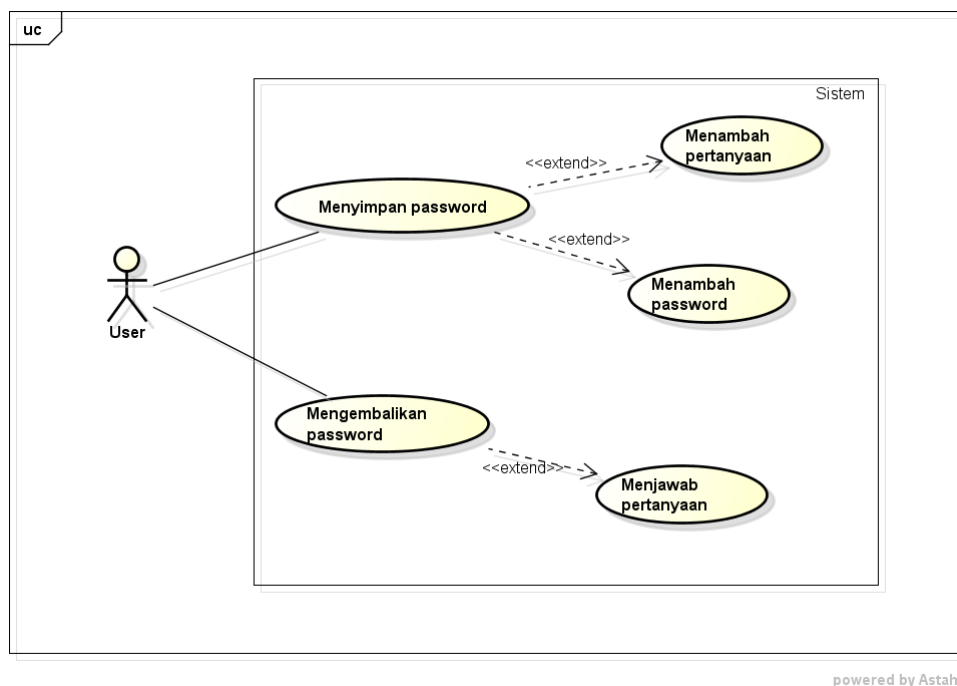
n	k	n	k	n	k
5	1	6	1	7	1..2
8	1..3	9	1..3	10	1..4
11	1..5	12	1..6	13	1..7
14	1..7	15	1..8	16	1..9
17	1..10	18	1..11	19	1..11
20	1..12	21	1..13	22	1..14
23	1..15	24	1..16	25	1..17
26	1..17	27	1..18	28	1..19
29	1..20	30	1..21

3.4 Diagram

Pada bagian ini dibuat diagram-diagram untuk perangkat lunak yang dibangun. Diagram-diagram ini dibuat berdasarkan analisis proses yang sudah dijelaskan pada Subbab 3.1.

3.4.1 Diagram Use Case

Perangkat lunak yang dibangun 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

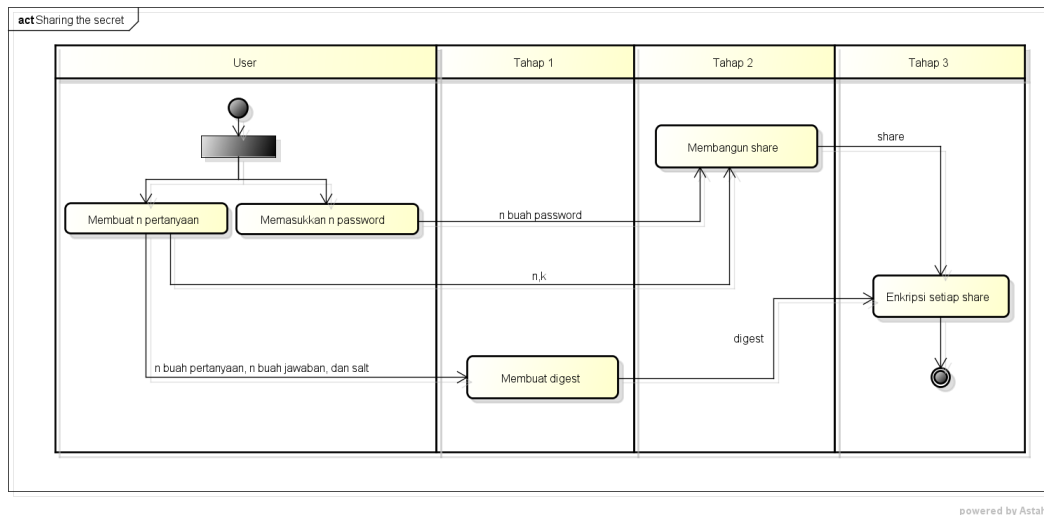
Skenario untuk mengembalikan *password* ditunjukkan oleh Tabel 3.6. Aktor dari skenario ini adalah pengguna dari sistem. Data masukan dari skenario ini adalah n jawaban dari pertanyaan keamanan yang sudah dibuat.

Tabel 3.6: Skenario Mengembalikan *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

3.4.2 Diagram Aktivitas

Perangkat lunak yang dibangun memiliki 2 proses, yaitu menyimpan *password* dan mengembalikan *password*. Urutan aktivitas yang dilakukan perangkat lunak dalam proses menyimpan *password* ditunjukkan pada diagram aktivitas 3.4.

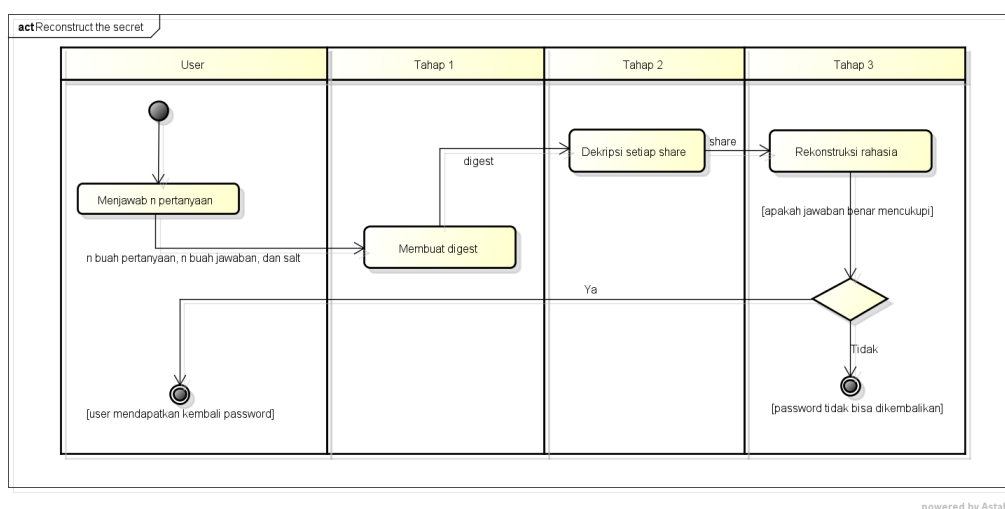


Gambar 3.4: Diagram aktivitas untuk menyimpan *password*

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

Setelah semua masukan yang dibutuhkan ada, sistem akan membuat *digest* dari konkatenasi setiap pertanyaan keamanan, jawaban, dan nilai *salt*. Sistem juga akan membangun *share* dari masukan *password*. Selanjutnya sistem akan mengenkripsi setiap *share* dari masing-masing *password* dan menggunakan *digest* dari konkatenasi setiap pertanyaan keamanan, jawaban, dan nilai *salt* sebagai kunci proses enkripsi. Setelah proses enkripsi, sistem akan menyimpan seluruh *share* yang sudah dibangun, pertanyaan yang dibuat, k , dan nilai *salt*.

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



Gambar 3.5: Diagram aktivitas untuk mengembalikan *password*

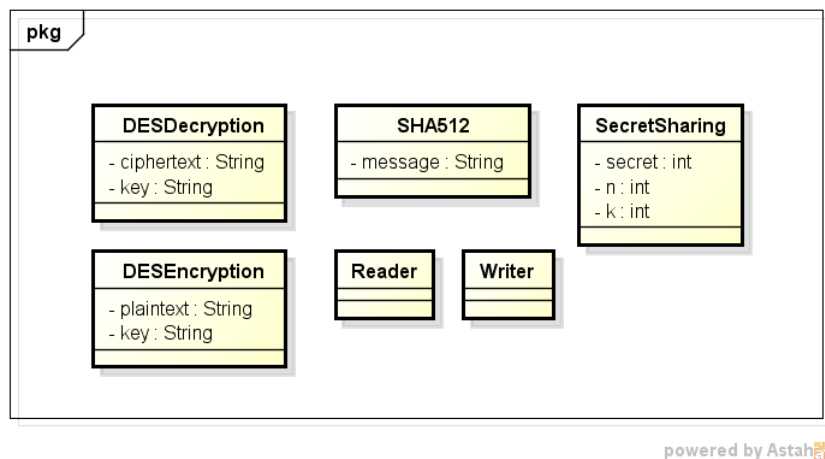
Dalam proses untuk mengembalikan *password*, pengguna akan diminta untuk menjawab n pertanyaan keamanan yang disimpan saat proses penyimpanan *password*. Sistem akan membuat *digest*

dari konkatenasi setiap pertanyaan keamanan, jawaban pertanyaan keamanan dari masukan pengguna, dan nilai *salt* yang disimpan.

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

3.4.3 Diagram Kelas

Pada bagian ini berisi rancangan diagram kelas dari perangkat lunak yang dibangun. Kelas-kelas ini merupakan rancangan kelas yang dibutuhkan untuk membangun perangkat lunak dan dibuat berdasarkan proses-proses yang sudah dijelaskan pada Subbab 3.1. Rancangan 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

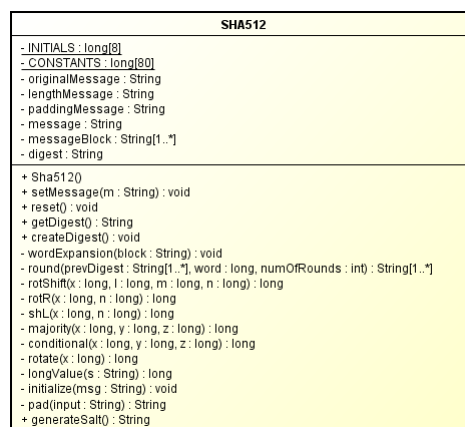
Diagram kelas rinci digunakan sebagai gambaran umum untuk setiap kelas yang ada dalam perangkat lunak yang dibangun serta keterkaitan setiap kelas. Diagram kelas rinci dapat dilihat pada Gambar 4.1. Ada perbedaan antara diagram kelas pada Gambar 4.1 dengan kelas diagram pada Bab 3. Pada diagram kelas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

4.2 Deskripsi Kelas dan Fungsi

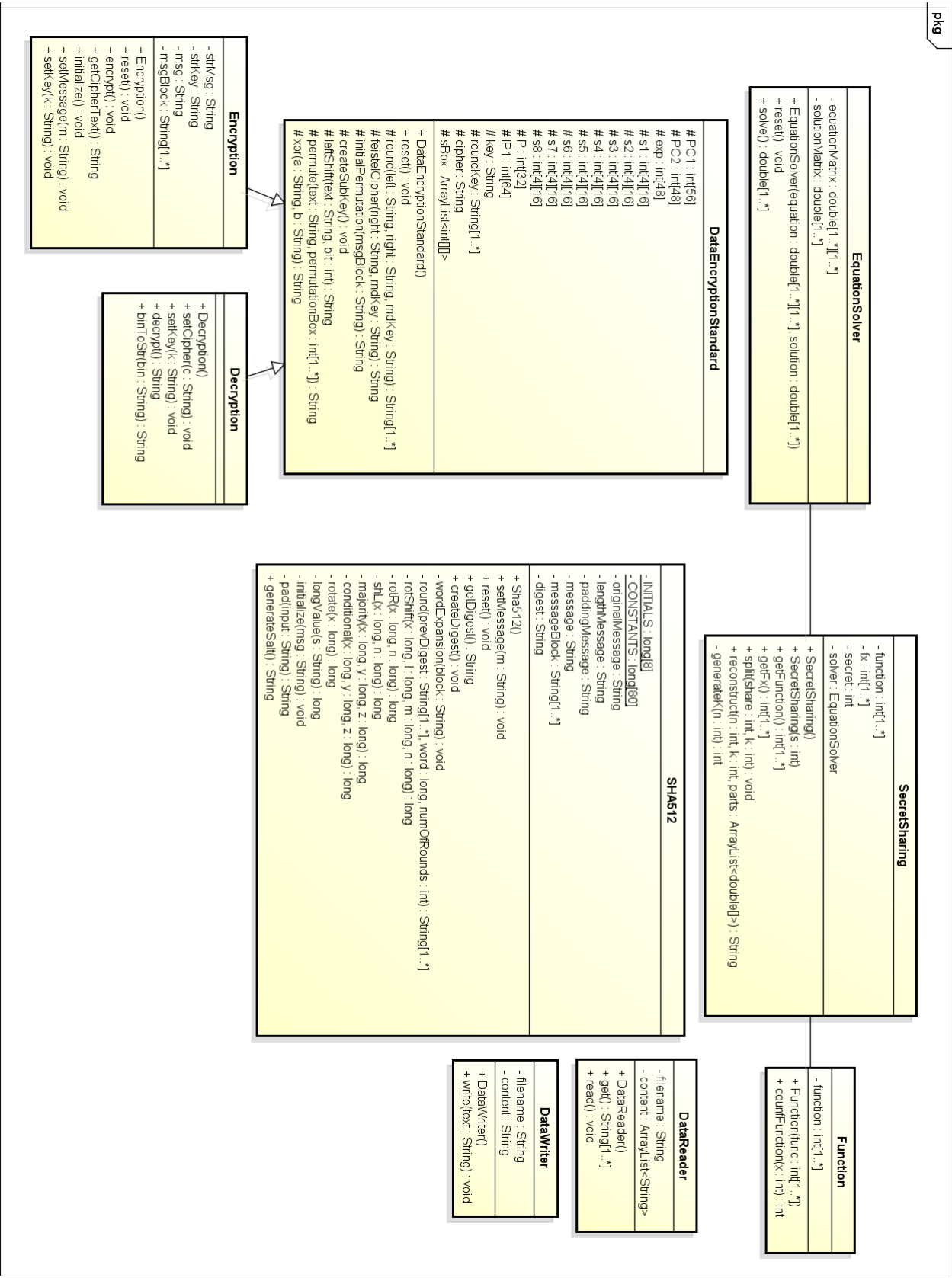
Pada bagian ini akan berisi mengenai penjelasan secara rinci masing-masing kelas. Tujuannya adalah menjelaskan peran setiap kelas dalam perangkat lunak yang dibangun.

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



Gambar 4.1: Diagram Kelas Rinci

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 *padding* 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 *padding* 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*, *pad* dan *generateSalt*. 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.

5. *void createDigest*

Fungsi yang berperan untuk membuat *digest* dari *message*. Algoritma untuk membuat *digest* ini dapat dilihat pada bagian 2.5.

6. *void wordExpansion(String block)*

Fungsi yang berperan untuk melakukan proses ekspansi blok *message*.

7. *String[] round(String[] prevDigest, long word, int numOfRounds)*

Fungsi yang berperan untuk melakukan 1 putaran dari *SHA512*. Proses yang terjadi dalam 1 putaran *SHA512* dapat dilihat pada Bab 2.5.

8. *long rotShift(long x, long m, long n)*

Fungsi yang berperan untuk melakukan operasi $RotShift_{l-m-n}(x)$ (Subbab 2.5.3).

9. *long rotR(long x, long n)*

Fungsi yang berperan untuk melakukan operasi $RotR_i(x)$ (Subbab 2.5.3).

10. *long shL(long x, long n)*

Fungsi yang berperan untuk melakukan operasi $ShL_i(x)$ (Subbab 2.5.3).

11. *long majority(long x, long y, long z)*

Fungsi yang berperan untuk melakukan operasi $Majority(x, y, z)$ (Subbab 2.5.4).

12. *long conditional(long x, long y, long z)*

Fungsi yang berperan untuk melakukan operasi $Conditional(x, y, z)$ (Subbab 2.5.4).

13. *long rotate(long x)*

Fungsi yang berperan untuk melakukan operasi $Rotate(x)$ (Subbab 2.5.4).

14. *long longValue(String s)*

Fungsi yang berperan mengubah tipe data *string* *s* menjadi tipe data *long*.

15. *void initialize(String msg)*

Fungsi yang berperan untuk melakukan proses *message padding* dan memecah menjadi blok-blok *message*.

16. *String pad(String input)*

Fungsi yang berperan untuk menambahkan *padding* angka 0 pada *string* biner *input*. 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

```

17. *String generateSalt*

Fungsi yang berguna untuk membangun nilai *salt*. Nilai *salt* yang dibangun terdiri dari 4 karakter ASCII acak. Algoritma fungsi ini ditunjukkan pada Algoritma 2.

Algorithm 2 generateSalt

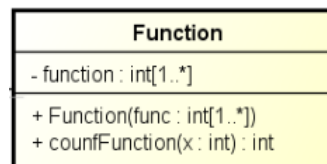
```

1: function GENERATESALT
2:    $i = 0$ 
3:   for  $i < 4$  do
4:      $temp = random((126 - 33) + 1) + 33$ 
5:      $salt += temp.toString$ 
6:   end for
7:   return  $salt$ 
8: 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.

Gambar 4.3: Kelas *Function*

Adapun atribut dari kelas *Function* adalah *function*. Atribut *function* berguna untuk menyimpan nilai setiap koefesien 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 3.

Algorithm 3 countFunction

```

1: function COUNTFUNCTION( $x$ )
2:   for  $i < function.length$  do
3:      $res = res + (function[i] \cdot x^i)$ 
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[1..*][1..*] equationMatrix*

Atribut yang menyimpan bentuk matriks dari persamaan linear.

2. *double[1..*] 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[1..*][1..*] equation, double[1..*] solution)*

Merupakan konstruktor dari kelas *EquationSolver* yang menerima masukan berupa matriks persamaan dan matriks solusi.

2. *void reset*

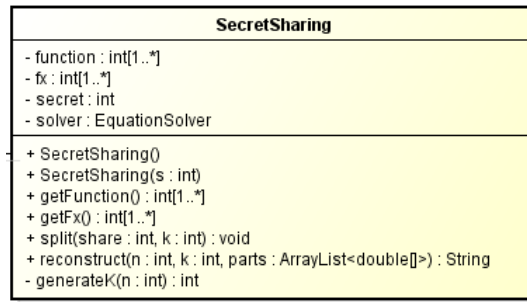
Mengembalikan nilai atribut *equationMatrix* dan *solutionMatrix* menjadi *array* kosong.

3. *double[1..*] 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.

3. *int secret*

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

4. *EquationSolver solver*

Atribut dari objek kelas *EquationSolver*.

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

1. *SecretSharing*

Merupakan konstruktor kosong dari kelas *SecretSharing*.

2. *SecretSharing(int s)*

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

3. *int[] getFunction*

Fungsi yang berguna untuk mengembalikan nilai atribut *function*.

4. *int[] getFx*

Fungsi yang berguna untuk mengembalikan nilai atribut *fx*.

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

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

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

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

7. *int generateK(int n)*

Fungsi ini berperan untuk menentukan nilai *k* yang ideal berdasarkan nilai *n*. Algoritma untuk menentukan nilai *k* ditunjukkan oleh Algoritma 4.

Algorithm 4 generateK

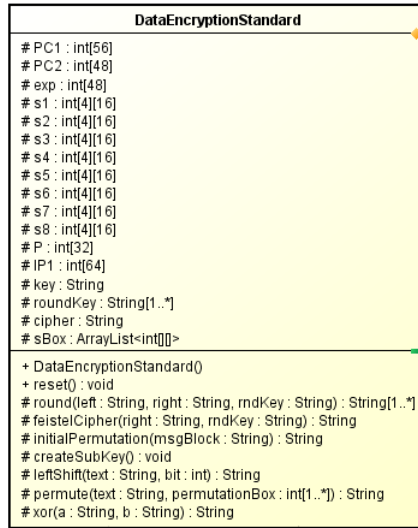
```

1: function GENERATEK(n)
2:    $k = n/2 + 1$ 
3:   if  $k > n$  then
4:      $k = n$ 
5:   end if
6:   return  $k$ 
7: end function

```

4.2.5 Kelas *DataEncryptionStandard*

Kelas *DataEncryptionStandard* merupakan kelas yang mengimplementasikan algoritma *Data Encryption Standard* (DES). Kelas ini akan menjadi kelas *parent* dari proses enkripsi dan dekripsi. Struktur kelas *DataEncryptionStandard* ditunjukkan oleh Gambar 4.6.



Gambar 4.6: Kelas *DataEncryptionStandard*

Kelas Encryption memiliki beberapa atribut, yaitu *PC1*, *PC2*, *IP*, *exp*, *s1*, *s2*, *s3*, *s4*, *s5*, *s6*, *s7*, *s8*, *P*, *IP1*, *key*, *roundKey*, *cipher*, dan *sBox*. 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.
7. *int[4][16] s3*
Atribut yang menyimpan matriks *s-box* ke-3.
8. *int[4][16] s4*
Atribut yang menyimpan matriks *s-box* ke-4.
9. *int[4][16] s5*
Atribut yang menyimpan matriks *s-box* ke-5.
10. *int[4][16] s6*
Atribut yang menyimpan matriks *s-box* ke-6.
11. *int[4][16] s7*
Atribut yang menyimpan matriks *s-box* ke-7.
12. *int[4][16] s8*
Atribut yang menyimpan matriks *s-box* ke-8.
13. *int[32] P*
Atribut yang menyimpan matriks permutasi pada tahap terakhir fungsi DES.
14. *int[64] IP1*
Atribut yang menyimpan matriks permutasi akhir.
15. *String key*
Atribut yang menyimpan kunci dalam bentuk *string* biner.
16. *String[] roundKey*
Atribut yang menyimpan kunci untuk setiap putaran dalam *array string* biner.
17. *String cipher*
Atribut yang menyimpan *ciphertext* hasil enkripsi dalam bentuk *string* biner.
18. *ArrayList<int[][]> sBox*
Atribut yang menyimpan seluruh matriks *s-box*.

Adapun fungsi-fungsi yang dimiliki oleh kelas *DataEncryptionStandard*, yaitu *DataEncryptionStandard*, *reset*, *round*, *feistelCipher*, *initialPermutation*, *createSubKey*, *leftShift*, *permute*, dan *xor*. Berikut penjelasan masing-masing fungsi:

1. *DataEncryptionStandard*

Merupakan konstruktor dari kelas *DataEncryptionStandard*.

2. *void reset*

Fungsi yang berperan untuk mengembalikan atribut *key* dan *cipher* menjadi *string* kosong. Fungsi ini juga mengembalikan atribut *roundKey* menjadi *array string* kosong.

3. *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.

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

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

5. *String initialPermutation(String msgBlock)*

Fungsi ini berperan untuk melakukan proses permutasi awal.

6. *void createSubKey*

Fungsi yang berperan untuk membangkitkan kunci putaran.

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

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

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

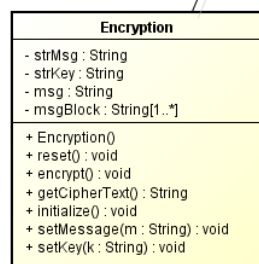
Fungsi yang berperan untuk melakukan proses permutasi.

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

Fungsi yang berperan untuk melakukan operasi XOR.

4.2.6 Kelas *Encryption*

Kelas *Encryption* merupakan kelas turunan dari kelas *DataEncryptionStandard*. Kelas ini berperan untuk melakukan proses enkripsi menggunakan algoritma *Data Encryption Standard*. Struktur kelas *Encryption* ditunjukkan oleh Gambar 4.7.



Gambar 4.7: Kelas *Encryption*

Kelas *Encryption* memiliki beberapa atribut, yaitu *strMsg*, *strKey*, *msg*, dan *msgBlock*. Berikut penjelasan masing-masing atribut:

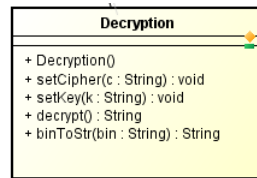
1. *String strMsg*
Atribut yang menyimpan *plaintext* dalam bentuk *string*.
2. *String strKey*
Atribut yang menyimpan kunci dalam bentuk *string*.
3. *String msg*
Atribut yang menyimpan *plaintext* dalam bentuk *string* biner.
4. *String[] msgBlock*
Atribut yang menyimpan blok-blok *plaintext* dalam bentuk *array string* biner. Setiap elemen *array* berisi *string* biner dengan panjang 64-bit.

Adapun fungsi-fungsi yang dimiliki oleh kelas *Encryption*, yaitu *Encryption*, *reset*, *encrypt*, *getCipherText*, *initialize*, *setMessage*, dan *setKey*. Berikut penjelasan masing-masing fungsi:

1. *Encryption*
Merupakan konstruktor dari kelas *Encryption*.
2. *void reset*
Fungsi yang berperan untuk mengembalikan atribut *strMsg*, *strKey*, *msg*, *key*, dan *cipher* menjadi *string* kosong. Fungsi ini juga mengembalikan atribut *roundKey* dan *msgBlock* menjadi *array string* kosong.
3. *void encrypt*
Fungsi yang berperan untuk melakukan proses enkripsi DES.
4. *String getCipherText*
Fungsi yang mengembalikan nilai atribut *cipher* dalam bentuk *string* heksadesimal.
5. *void setMessage(String m)*
Fungsi untuk menyimpan *string m* ke dalam atribut *strMsg*.
6. *void setKey(String k)*
Fungsi untuk menyimpan *string k* ke dalam atribut *strKey*.
7. *void initialize*
Fungsi yang berperan untuk melakukan *padding* terhadap *strMsg* dan memecah *strMsg* menjadi blok-blok *string*. Blok-blok *string* ini akan disimpan pada atribut *msgBlock*.

4.2.7 Kelas *Decryption*

Kelas *Decryption* merupakan kelas turunan dari kelas *DataEncryptionStandard*. Kelas ini berperan untuk melakukan proses dekripsi menggunakan algoritma *Data Encryption Standard*. Struktur kelas *Decryption* ditunjukkan pada Gambar 4.8.

Gambar 4.8: Kelas *DESDecryption*

Kelas *Decryption* tidak memiliki atribut. Seluruh atribut dari kelas ini diturunkan dari kelas *DataEncryptionStandard*. Adapun fungsi-fungsi yang dimiliki kelas *Decryption*, yaitu *Decryption*, *setCipher*, *setKey*, *decrypt*, dan *binToStr*. Berikut penjelasan masing-masing fungsi:

1. *Decryption*

Merupakan konstruktor dari kelas *Decryption*.

2. *void setCipher(String c)*

Fungsi untuk menyimpan *string c* pada atribut *cipher* dalam bentuk *string* biner.

3. *void setKey(String k)*

Fungsi untuk menyimpan *string k* pada atribut *key* dalam bentuk *string* biner.

4. *String decrypt*

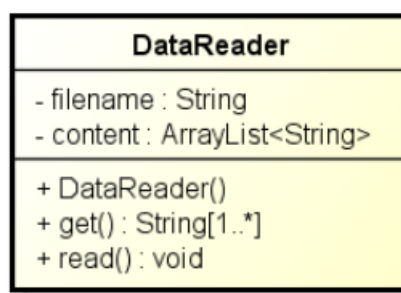
Fungsi yang berperan untuk melakukan proses dekripsi menggunakan DES.

5. *String binToStr(String bin)*

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

4.2.8 Kelas *DataReader*

Kelas *DataReader* merupakan kelas yang berperan untuk membaca berkas teks. Kelas *DataReader* ditunjukkan pada Gambar 4.9.

Gambar 4.9: 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.

2. *ArrayList<String> content*

Atribut yang menyimpan isi dari berkas teks yang dibaca.

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

1. *DataReader*

Merupakan konstruktor dari kelas *DataReader*.

2. *String[] get*

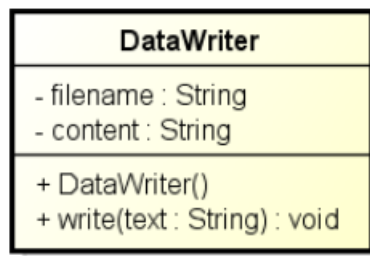
Fungsi yang berguna untuk mengembalikan atribut *content*.

3. *void read*

Fungsi yang berperan membaca berkas teks.

4.2.9 Kelas *Data Writer*

Kelas *Data Writer* adalah kelas yang berperan untuk menulis keluaran ke dalam berkas teks. Kelas *Data Writer* ditunjukkan pada Gambar 4.10.



Gambar 4.10: Kelas *Data Writer*

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 akan ditulis.

2. *String content*

Atribut yang menyimpan isi dari berkas teks yang akan ditulis.

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

1. *DataWriter*

Merupakan konstruktor dari kelas *Data Writer*.

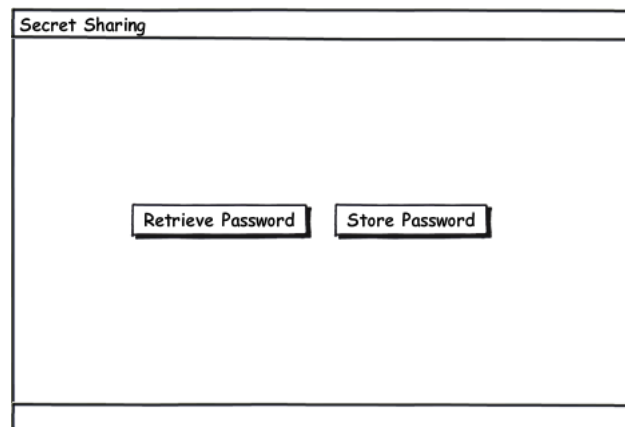
2. *write(String text)*

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*.

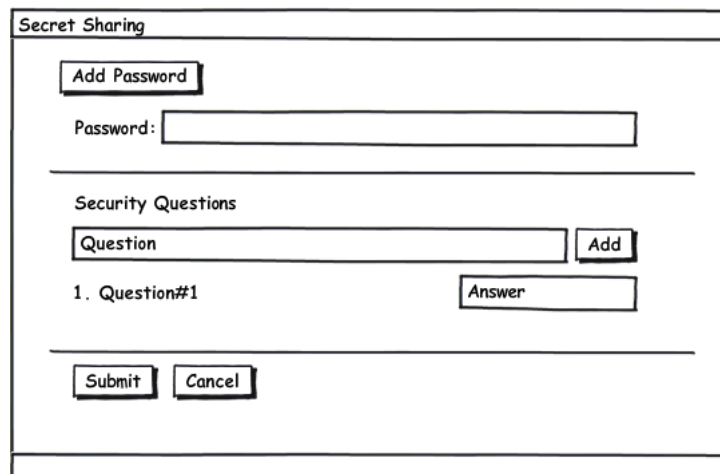
Gambar 4.11 menunjukkan tampilan awal yang akan dimunculkan pertama kali untuk memilih menyimpan *password* atau mengembalikan *password*.



The screenshot shows a window titled "Secret Sharing". Inside the window, there are two buttons: "Retrieve Password" and "Store Password". The window has a simple border and a title bar.

Gambar 4.11: Perancangan Tampilan Awal

Tampilan utama ini cukup sederhana. Dalam tampilan utama pada Gambar 4.11, 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*.



The screenshot shows a window titled "Secret Sharing". Inside the window, there is an "Add Password" button. Below it is a "Password:" label followed by a text box. Below that is a "Security Questions" section. It contains a "Question" label followed by a text box and an "Add" button. Below this is a list of questions, starting with "1. Question#1", followed by an "Answer" label and a text box. At the bottom of the window are "Submit" and "Cancel" buttons.

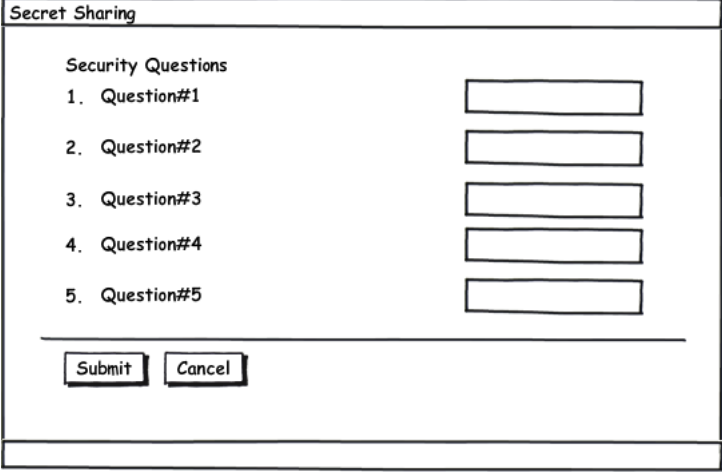
Gambar 4.12: Perancangan Tampilan Menyimpan *Password*

Pada tampilan menyimpan *password* di Gambar 4.12, 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.

Setelah mengisi seluruh pertanyaan keamanan, pengguna bisa menyimpan *password* dengan menekan tombol "*Submit*". Tombol "*Cancel*" berfungsi untuk kembali ke tampilan awal. Setelah tombol "*Submit*" ditekan, maka *password* sudah disimpan dan akan kembali ditampilkan tampilan awal.

Berikutnya adalah tampilan untuk mengembalikan *password*. Gambar 4.13 menunjukkan tampilan untuk mengembalikan *password*.

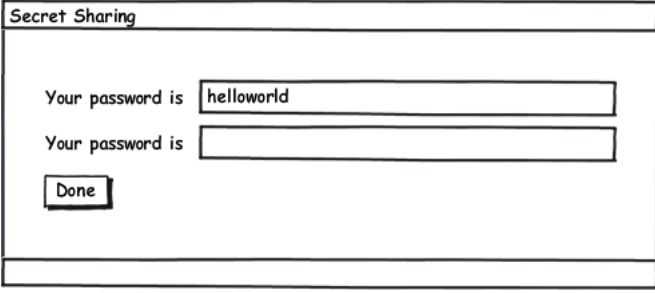


The screenshot shows a window titled "Secret Sharing". Inside, under the heading "Security Questions", there are five numbered questions: "1. Question#1", "2. Question#2", "3. Question#3", "4. Question#4", and "5. Question#5". To the right of each question is a rectangular input field. At the bottom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

Gambar 4.13: Perancangan Tampilan Mengembalikan *Password*

Pada bagian untuk mengembalikan *password*, tampilannya cukup sederhana dan pengguna hanya cukup memasukkan setiap jawaban dari pertanyaan keamanan yang sudah dibuat sebelumnya di bagian penyimpanan *password*. Pada bagian ini, pengguna bebas untuk memilih mengisi setiap pertanyaan atau tidak menjawab pertanyaan keamanan. Setelah seluruh pertanyaan sudah dijawab, pengguna dapat menekan tombol "*Submit*" yang kemudian akan menunjukkan *password* pengguna.

Gambar 4.14 menunjukkan tampilan sesudah pengguna menekan tombol "*Submit*" pada bagian di Gambar 4.13.



The screenshot shows the same "Secret Sharing" window. The first input field now contains the text "helloworld". The second input field is empty. Below the input fields, there is a button labeled "Done".

Gambar 4.14: Perancangan Tampilan Mengembalikan *Password* (2)

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

BAB 5

IMPLEMENTASI DAN PENGUJIAN

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

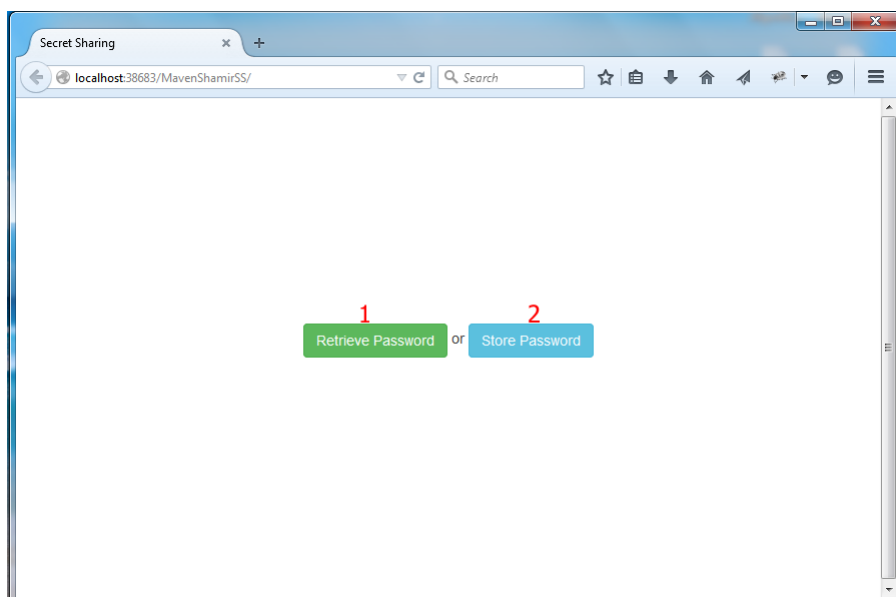
5.1 Implementasi Perangkat Lunak

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.

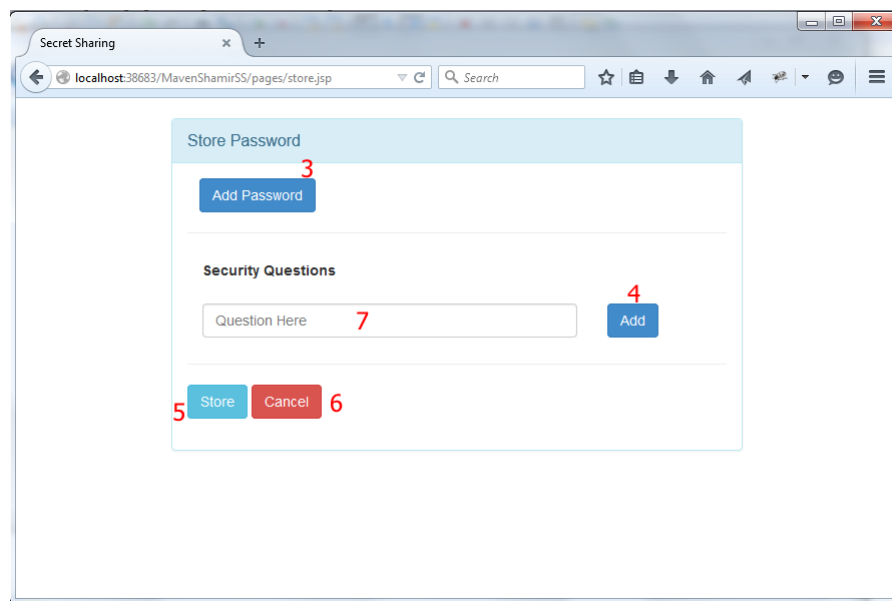
- Bagian nomor 1 merupakan tombol untuk mengembalikan *password*.
- Bagian nomor 2 merupakan tombol untuk menyimpan *password*.



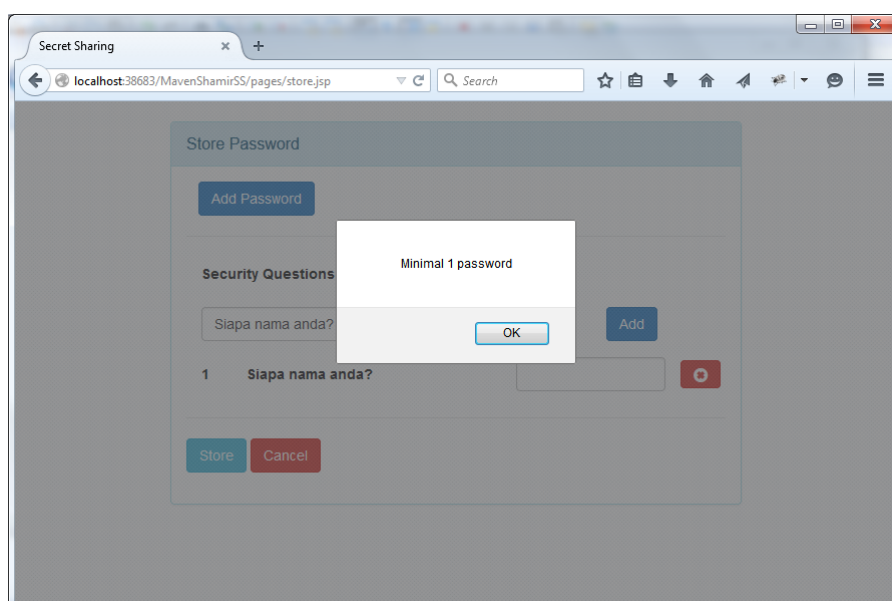
Gambar 5.1: Tampilan antarmuka awal

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

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

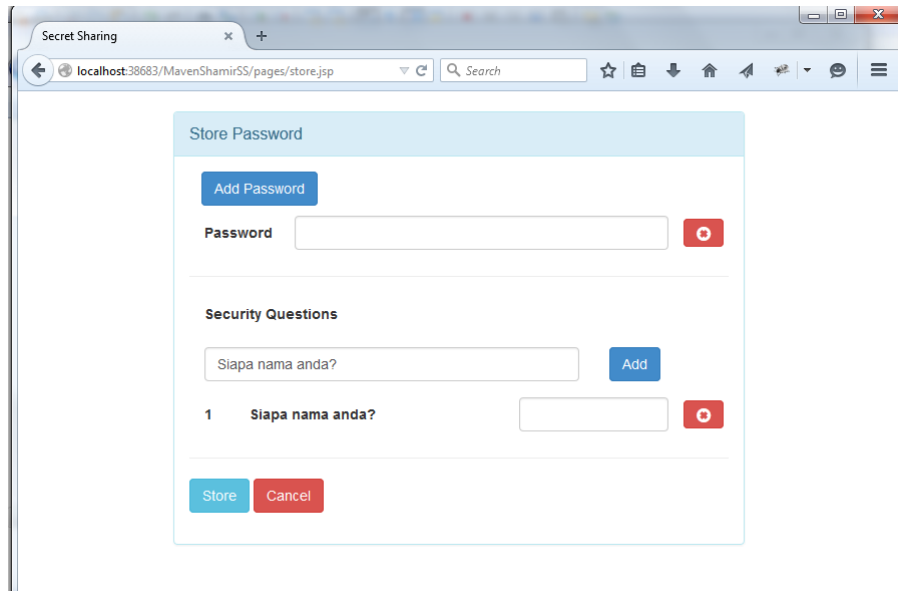


Gambar 5.2: Tampilan antarmuka untuk menyimpan *password*



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

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



Gambar 5.4: Tampilan antarmuka menambah *password*

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

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

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?

Gambar 5.5: Tampilan antarmuka untuk mengembalikan *password*

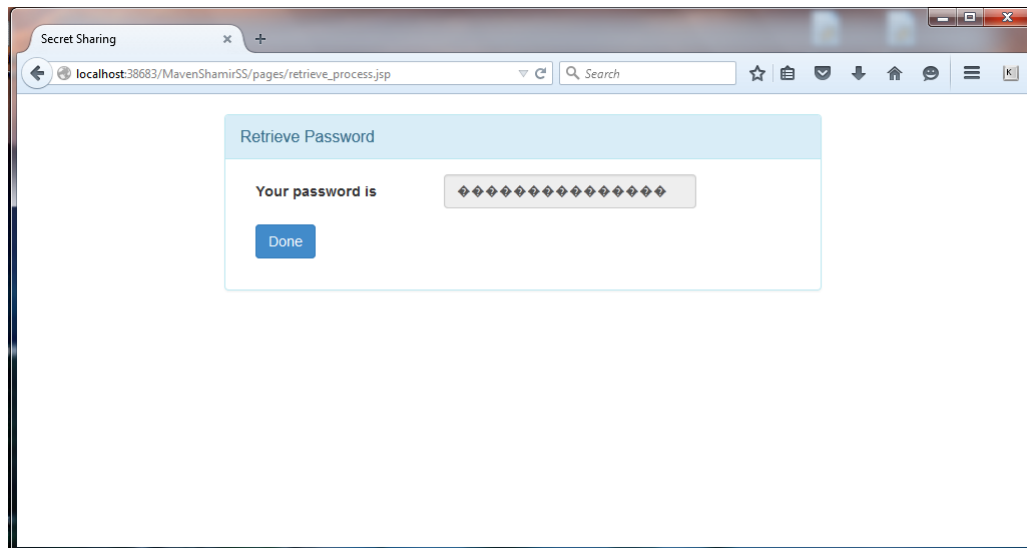
Setelah tombol "Submit" pada Gambar 5.5 ditekan, perangkat lunak akan memroses setiap pertanyaan dan jawaban. Jika banyak jawaban benar dari pertanyaan keamanan yang dijawab oleh pengguna sesuai dengan minimal banyak pertanyaan keamanan yang dijawab benar yang sudah ditentukan sebelumnya, maka tampilan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

Retrieve Password

Your password is

Gambar 5.6: Tampilan antarmuka untuk mengembalikan *password* (2)

Sedangkan, jika pertanyaan keamanan yang dijawab benar kurang dari minimal pertanyaan keamanan yang harus dijawab benar, maka perangkat lunak akan menunjukkan *password* yang tidak bisa dimengerti. Hal ini menunjukkan bahwa *password* tidak berhasil dikembalikan. Gambar 5.7 menunjukkan tampilan antarmuka perangkat lunak dengan *password* yang tidak bisa dimengerti.



Gambar 5.7: Tampilan antarmuka untuk mengembalikan *password* (3)

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.

Dalam pengujian survei ini, responden diberikan waktu tidak terbatas untuk mencoba menjawab pertanyaan keamanan yang sudah dibuat sampai *password* dapat dikembalikan. Responden pun diberikan kebebasan untuk mengakses berbagai media sebagai bantuan untuk memperoleh jawaban dari setiap pertanyaan keamanan yang dibuat. Tetapi, responden tidak boleh menanyakan jawaban dari setiap pertanyaan keamanan yang dibuat kepada pembuat pertanyaan keamanan. Dalam pengujian survei ini juga, banyak percobaan untuk mengembalikan *password* tidak dibatasi, responden boleh mencoba untuk mengembalikan password dengan menjawab pertanyaan keamanan yang dibuat sampai *password* bisa dikembalikan atau responden tidak ingin mencoba lagi.

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.

Store Password

Add Password

Password : password1

Password : password2

Password : password3

Password : password4

Password : password5

Security Questions

Question Here Add

1 Siapa nama anda? Samuel

2 Dimana kota anda lahir? Bandung

3 Apakah anda memiliki binatang peliharaan? Ya

4 Kapan anda lahir? 31 Juli 1993

5 Dimana anda berkuliah? UNPAR

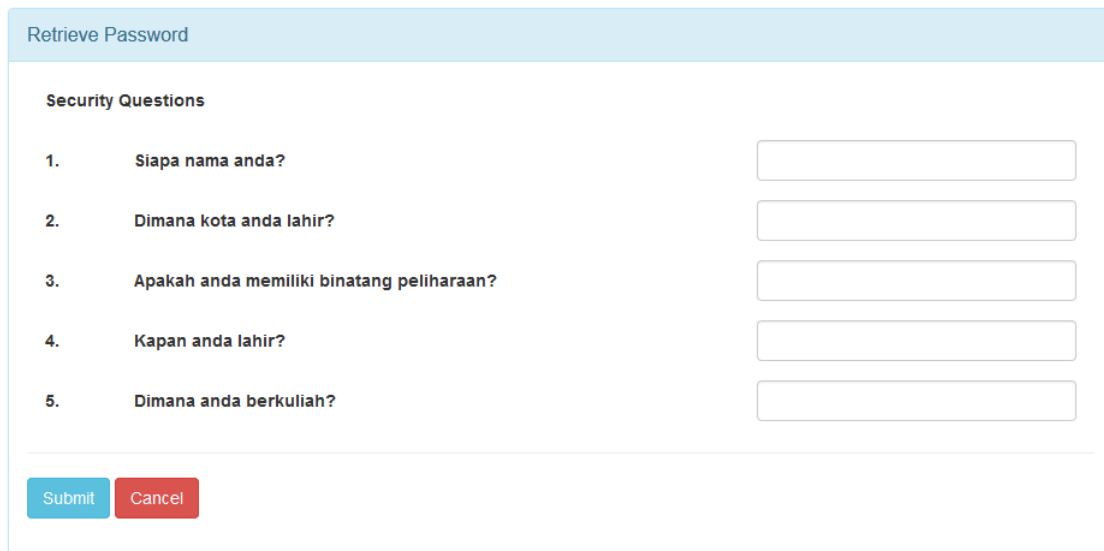
Store Cancel

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

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

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



The screenshot shows a web form titled "Retrieve Password". Under the heading "Security Questions", there are five numbered questions, each followed by an empty text input field:

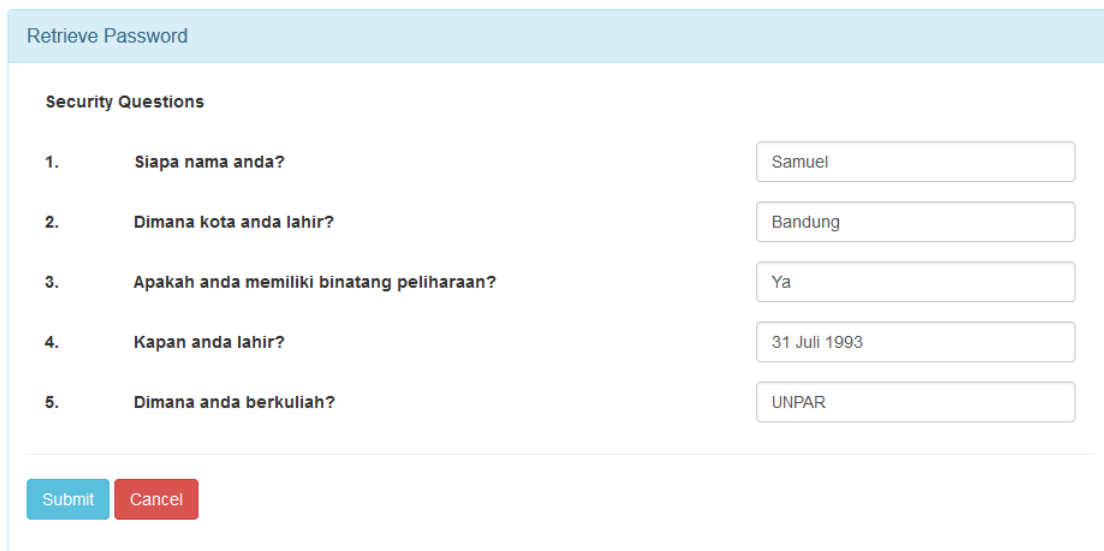
1. Siapa nama anda?
2. Dimana kota anda lahir?
3. Apakah anda memiliki binatang peliharaan?
4. Kapan anda lahir?
5. Dimana anda berkuliah?

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

Gambar 5.9: Tampilan Menjawab Pertanyaan Keamanan

Kasus 1

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



The screenshot shows the same "Retrieve Password" form as in Gambar 5.9, but now the input fields are filled with the following answers:

1. Siapa nama anda? Samuel
2. Dimana kota anda lahir? Bandung
3. Apakah anda memiliki binatang peliharaan? Ya
4. Kapan anda lahir? 31 Juli 1993
5. Dimana anda berkuliah? UNPAR


The "Submit" and "Cancel" buttons remain at the bottom.

Gambar 5.10: Tampilan Menjawab Pertanyaan Keamanan Kasus Pertama

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

Setelah tombol "Submit" pada Gambar 5.10 ditekan, perangkat lunak akan memproses setiap jawaban masing-masing pertanyaan. Dalam kasus pertama, seluruh pertanyaan bisa dijawab de-

ngan benar, maka *password* bisa dikembalikan. Gambar 5.11 menunjukkan bahwa *password* bisa dikembalikan.



Retrieve Password

Your password is password1

Your password is password2

Your password is password3

Your password is password4

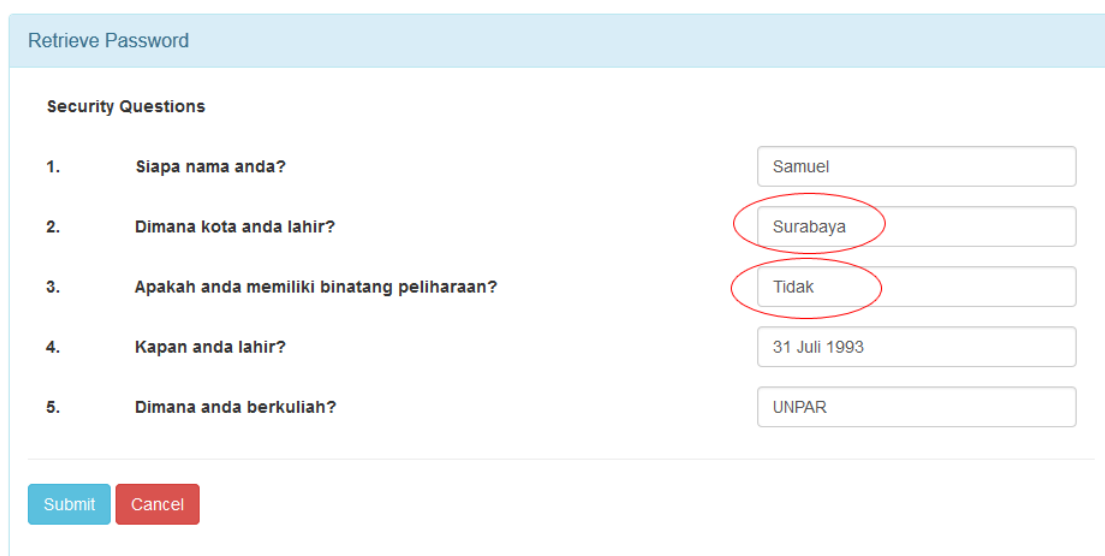
Your password is password5

Done

Gambar 5.11: Hasil Pengujian Fungsional Kasus Pertama

Kasus 2

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



Retrieve Password

Security Questions

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

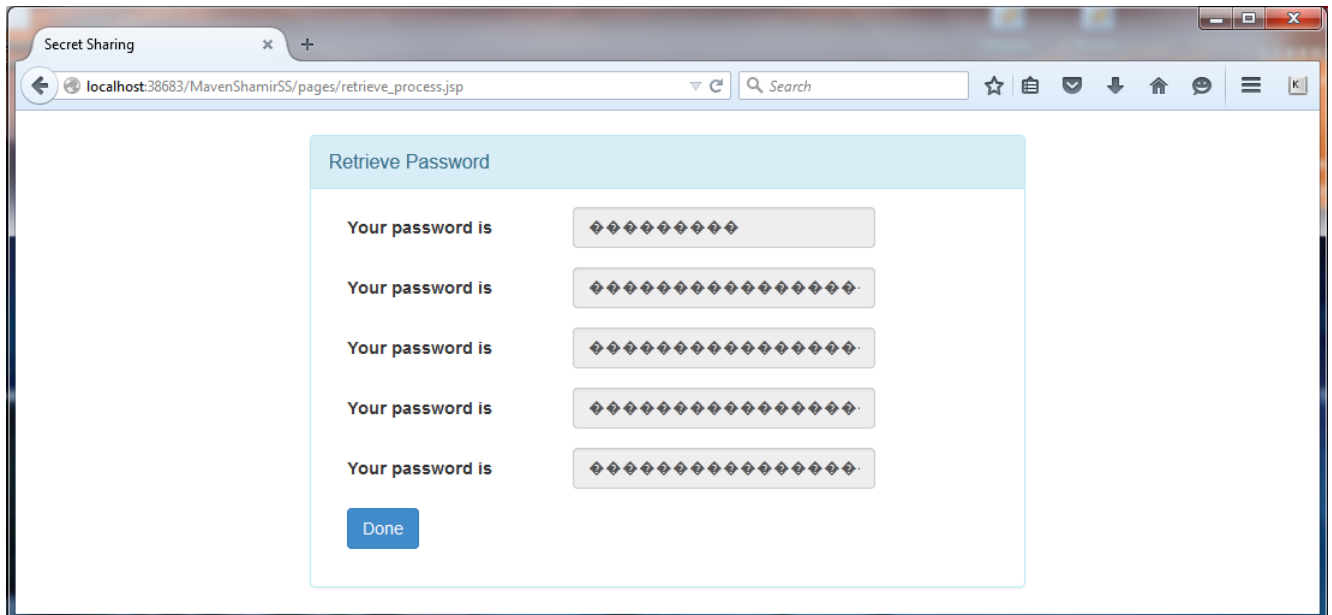
Submit Cancel

Gambar 5.12: Tampilan Menjawab Pertanyaan Keamanan Kasus Kedua

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

Langkah selanjutnya adalah memroses jawaban dari masing-masing pertanyaan dengan menekan tombol "*Submit*". Setelah tombol "*Submit*" ditekan, perangkat lunak akan menampilkan hasil

pengujian kasus kedua. Dalam kasus kedua, karena k yang dipilih $k = 4$ dan hanya 3 pertanyaan yang dijawab dengan benar, maka *password* tidak bisa dikembalikan. Gambar 5.13 menunjukkan langkah yang sudah dijelaskan.



Gambar 5.13: Hasil Pengujian Fungsional Kasus Kedua

5.2.3 Analisis Hasil Pengujian Fungsional

Pada bagian ini akan dibahas analisis dari hasil pengujian fungsional yang sudah dilakukan. Seperti yang sudah dibahas, pengujian fungsional dibagi menjadi 2 kasus, yaitu kasus *password* bisa dikembalikan dan kasus dimana *password* tidak bisa dikembalikan.

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

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

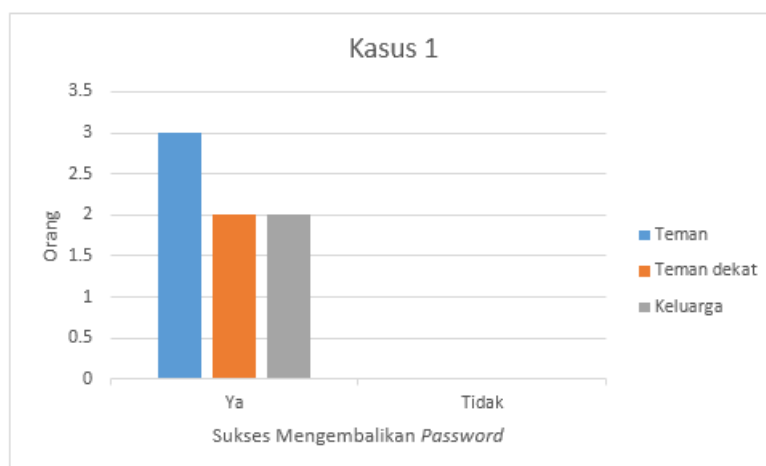
Kasus 1, 2, dan 3 memiliki 10 pertanyaan keamanan, $n = 10$, dan minimal 4 pertanyaan keamanan dijawab benar, $k = 4$. Sementara itu, untuk kasus 4 nilai n dan k ditambah. Dalam kasus 4, terdapat 15 pertanyaan keamanan, $n = 15$, dan minimal 6 pertanyaan keamanan dijawab benar, $k = 6$. Berikut tabel 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

Analisis Kasus 1

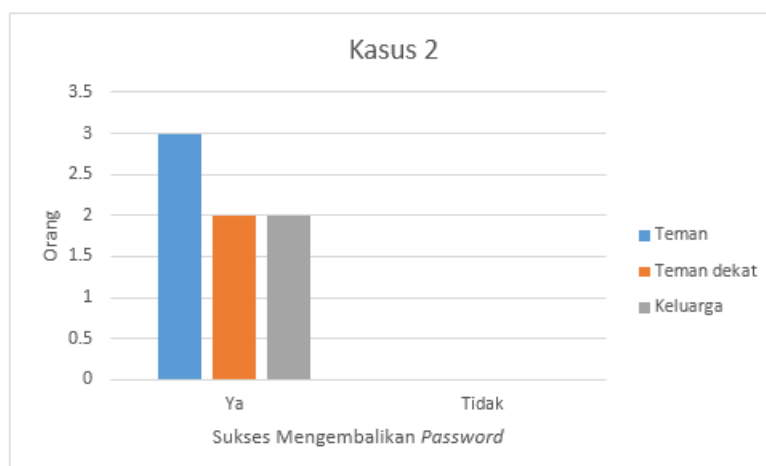
Dilihat dari grafik 5.14 untuk topik kasus 1, seluruh responden bisa berhasil mengembalikan *password*. Hal ini karena mayoritas kemungkinan jawaban dari pertanyaan keamanan adalah kemungkinan biner dengan hanya 2 kemungkinan saja (Ya atau Tidak). Pertanyaan keamanan yang kemungkinan jawabannya hanya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

Kasus 2

Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 2:

1. Pada tahun berapa anda lahir?
2. Pada tanggal berapa anda lahir?
3. Pada bulan apa anda lahir?
4. Berapa perbedaan umur anda dengan ayah anda?
5. Berapa perbedaan umur anda dengan ibu anda?
6. Berapa orang saudara anda?
7. Berapa nomor rumah tempat anda tinggal?
8. Dimana anda tinggal?
9. Apa merek kendaraan yang anda pakai?
10. Pada hari apa anda lahir?

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



Gambar 5.15: Pengujian survei kasus 2

Analisis Kasus 2

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

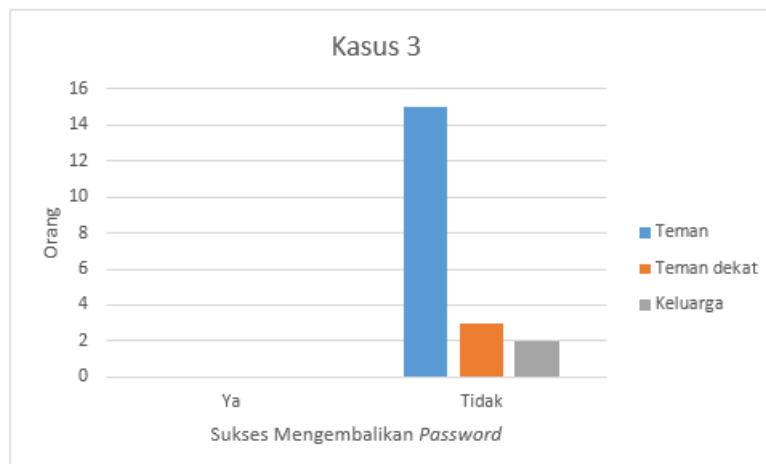
Responden dapat menjawab tanggal lahir karena hanya memiliki 30-31 kemungkinan, sedangkan untuk bulan hanya ada 12 kemungkinan, dan juga beberapa pertanyaan lain yang menyangkut angka. Dapat dilihat juga, bahwa beberapa jawaban untuk pertanyaan keamanan merupakan informasi yang sering ditunjukkan dalam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

Kasus 3

Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 3:

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

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



Gambar 5.16: Pengujian survei kasus 3

Analisis Kasus 3

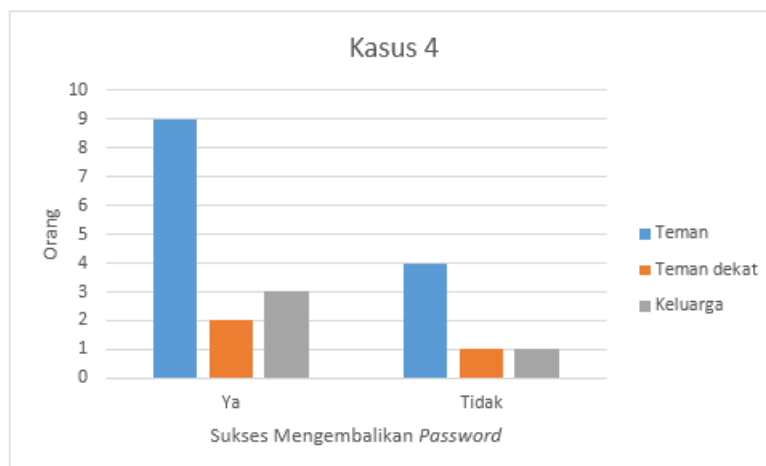
Untuk topik kasus 3, tidak ada responden yang berhasil mengembalikan *password*. Hal ini karena beberapa pertanyaan sifatnya sangat personal dan jawaban tidak bisa dengan mudah ditebak atau dicari di *internet* atau media sosial. Pertanyaan yang sifatnya sangat personal akan mempersulit untuk mengembalikan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

Kasus 4

Berikut daftar pertanyaan keamanan yang digunakan dalam kasus 4:

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

Kemudian, Grafik 5.17 menunjukkan hasil survei kasus 4.



Gambar 5.17: Pengujian survei kasus 4

Analisis Kasus 4

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

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

5.2.5 Kesimpulan Pengujian

Dari pengujian fungsional dan pengujian survei yang dilakukan maka bisa ditarik beberapa kesimpulan. Dari pengujian fungsional dapat ditarik kesimpulan bahwa pertanyaan keamanan yang dibuat sebaiknya memiliki sifat-sifat sebagai berikut.

1. Stabil

Pertanyaan keamanan tidak boleh berubah seiring berjalannya waktu.

2. Mudah diingat

Pertanyaan keamanan harus sifatnya personal sehingga mudah untuk diingat.

3. Sederhana

Pertanyaan keamanan harus sederhana tetapi sifatnya tetap personal.

Sementara itu, dari hasil pengujian survei dapat ditarik kesimpulan bahwa pertanyaan keamanan yang dibuat sebaiknya juga memiliki sifa-sifat sebagai berikut.

1. Aman

Pertanyaan keamanan harus tidak mudah ditebak dan tidak mudah diselidiki jawabannya.

2. Memiliki banyak kemungkinan jawaban

Pertanyaan keamanan tidak boleh memiliki sedikit kemungkinan jawaban karena akan mudah ditebak.

Namun, beberapa pertanyaan keamanan mungkin memiliki banyak kemungkinan jawaban dan aman sehingga tidak mudah ditebak tetapi tidak mudah diingat karena jawabannya terlalu rumit. Beberapa pertanyaan keamanan juga mungkin tidak sesuai dengan situasi atau keadaan dari pembuat pertanyaan. Sehingga, tidak ada pertanyaan keamanan yang memiliki seluruh sifat yang sudah dijelaskan di atas.

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 pengujian fungsional yang sudah dilakukan juga, dapat ditarik kesimpulan bahwa pertanyaan keamanan yang dibuat sebaiknya juga memiliki sifa-sifat sebagai berikut.

1. Stabil
2. Mudah diingat
3. Sederhana

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. Memiliki banyak kemungkinan jawaban

Dari hasil pengujian fungsional dan survei juga, dapat diketahui bahwa tidak ada pertanyaan keamanan yang memiliki seluruh sifat di atas, beberapa dari sifat di atas 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.

6.2 Saran

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

- Algoritma enkripsi yang digunakan, yaitu *data encryption standard* sebaiknya diganti dengan menggunakan algoritma enkripsi yang menggunakan panjang kunci lebih panjang dari *64-bit*.
- Metode *secret sharing* Shamir diharapkan dapat diimplementasikan tidak hanya pada perangkat lunak perorangan seperti dalam penelitian ini, tetapi bisa diimplementasikan pada sebuah sistem besar yang memiliki subsistem dan masing-masing dari subsistem ini menyimpan banyak informasi penting.
- Banyak percobaan untuk mengembalikan *password* harus dibatasi.

DAFTAR REFERENSI

- [1] M. Stamp, *Information security: principles and practice*. John Wiley & Sons, 2011.
- [2] R. Munir, *Matematika Diskrit*. Informatika Bandung, 2010.
- [3] B. A. Forouzan, *Cryptography & Network Security*. McGraw-Hill, Inc., 2007.
- [4] D. Norman and D. Wolczuk, *Introduction to Linear Algebra for Science and Engineering*. Pearson, 2012.
- [5] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [6] 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.
- [7] C. Shannon, “A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656,” *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948.
- [8] 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

```

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

```

```

72     {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
73     {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
74     {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
75     {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
76 };
77
78 private int[][] s3 = {
79     {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
80     {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
81     {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
82     {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}
83 };
84
85 private int[][] s4 = {
86     {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
87     {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
88     {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
89     {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
90 };
91
92 private int[][] s5 = {
93     {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
94     {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
95     {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
96     {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}
97 };
98
99 private int[][] s6 = {
100     {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
101     {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
102     {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
103     {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 10, 0, 8, 13}
104 };
105
106 private int[][] s7 = {
107     {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
108     {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
109     {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
110     {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}
111 };
112
113 private int[][] s8 = {
114     {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
115     {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 10, 14, 9, 2},
116     {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
117     {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}
118 };
119
120 //straight permutation box after s-box substitution
121 private int[] P = {
122     16, 7, 20, 21,
123     29, 12, 28, 17,
124     1, 15, 23, 26,
125     5, 18, 31, 10,
126     2, 8, 24, 14,
127     32, 27, 3, 9,
128     19, 13, 30, 6,
129     22, 11, 4, 25
130 };
131
132 //final permutation
133 private int[] IP1 = {
134     40, 8, 48, 16, 56, 24, 64, 32,
135     39, 7, 47, 15, 55, 23, 63, 31,
136     38, 6, 46, 14, 54, 22, 62, 30,
137     37, 5, 45, 13, 53, 21, 61, 29,
138     36, 4, 44, 12, 52, 20, 60, 28,
139     35, 3, 43, 11, 51, 19, 59, 27,
140     34, 2, 42, 10, 50, 18, 58, 26,
141     33, 1, 41, 9, 49, 17, 57, 25
142 };
143
144 private String strMsg;
145 private String strKey;
146 private String msg;
147 private String key;
148 private String[] roundKey;
149 private String cipher;
150 private ArrayList<int[][]> sBox;
151 private String[] msgBlock;
152
153 public Encryption() {
154     strMsg = "";
155     strKey = "";
156     msg = "";
157     key = "";
158     roundKey = new String[16];
159     cipher = "";
160     msgBlock = new String[1];
161
162     sBox = new ArrayList<int[][]>();
163     sBox.add(s1);
164     sBox.add(s2);
165     sBox.add(s3);
166     sBox.add(s4);
167     sBox.add(s5);
168     sBox.add(s6);
169     sBox.add(s7);
170     sBox.add(s8);

```

```

171     }
172
173     public void reset() {
174         strMsg = "";
175         strKey = "";
176         msg = "";
177         key = "";
178         roundKey = new String[16];
179         cipher = "";
180         msgBlock = new String[1];
181     }
182
183     public void encrypt() {
184         createSubKey();
185         String tempCipher = "";
186         for(int block = 0; block < msgBlock.length; block++) {
187             String init = initialPermutation(msgBlock[block]);
188             String L0 = init.substring(0, init.length()/2);
189             String R0 = init.substring(init.length()/2, init.length());
190
191             //16 rounds
192             String[] arr = {L0, R0};
193             for(int i = 0; i < 16; i++) {
194                 arr = round(arr[0], arr[1], roundKey[i]);
195             }
196
197             tempCipher = arr[1] + arr[0];
198             tempCipher = permute(tempCipher, IP1);
199             cipher += tempCipher;
200         }
201     }
202
203     public String getCipherText() {
204         String cipherText = "";
205         String[] temp = cipher.split("(?<=\\G.{4})");
206         for(int i = 0; i < temp.length; i++) {
207             cipherText += Integer.toHexString(Integer.parseInt(temp[i], 2));
208         }
209         return cipherText;
210     }
211
212     private String[] round(String left, String right, String rndKey) {
213         String Ln = right;
214         String Rn = xor(left, feistelCipher(right, rndKey));
215         String[] arr = {Ln, Rn};
216         return arr;
217     }
218
219     private String feistelCipher(String right, String rndKey) {
220         String res = "";
221
222         res = xor(rndKey, permute(right, exp));
223
224         String[] B = res.split("(?<=\\G.{6})");
225         String temp = "";
226         for(int i = 0; i < sBox.size(); i++) {
227             String index = B[i];
228             int[][] sbox = sBox.get(i);
229             int row = Integer.parseInt(index.charAt(0) + "" + index.charAt(index.length()-1) + "", 2);
230             int column = Integer.parseInt(index.substring(1, index.length()-1), 2);
231             temp += String.format("%4s", Integer.toBinaryString(sbox[row][column])).replace(' ', '0');
232         }
233
234         res = permute(temp, P);
235
236         return res;
237     }
238
239     private String initialPermutation(String msgblock) {
240         String res = permute(msgblock, IP);
241         return res;
242     }
243
244     private void createSubKey() {
245         String K2 = permute(key, PC1);
246         String left = K2.substring(0, K2.length()/2);
247         String right = K2.substring(K2.length()/2, K2.length());
248         for(int i = 0; i < 16; i++) {
249             int shift = 2;
250             int round = i + 1;
251             if(round == 1 || round == 2 || round == 9 || round == 16) {
252                 shift = 1;
253             }
254             String leftN = leftShift(left, shift);
255             String rightN = leftShift(right, shift);
256             roundKey[i] = leftN+rightN;
257             left = leftN;
258             right = rightN;
259         }
260
261         for(int i = 0; i < roundKey.length; i++) {
262             roundKey[i] = permute(roundKey[i], PC2);
263         }
264     }
265
266     private String leftShift(String text, int bit) {
267         String res = "";
268         char[] ch = new char[text.length()];
269         for(int i = 0; i < ch.length; i++) {

```

```

270         int shift = (i-bit) < 0 ? text.length()+(i-bit) : (i-bit);
271         ch[shift] = text.charAt(i);
272     }
273     res = new String(ch);
274     return res;
275 }
276
277 public void initialize() {
278     String tempMessage = "";
279     for(int i = 0; i < strMsg.length(); i++) {
280         tempMessage += String.format("%8s", Integer.toBinaryString(strMsg.charAt(i))).replace(' ', '0')
281     }
282
283     int padLength = (64 - (tempMessage.length() % 64)) / 8;
284     String padding = "";
285     for(int i = 0; i < padLength; i++) {
286         padding += String.format("%8s", Integer.toBinaryString(32)).replace(' ', '0');
287     }
288
289     msg = tempMessage + padding;
290
291     msgBlock = msg.split("(?<=\\G.{64})");
292
293     for(int i = 0; i < strKey.length(); i++) {
294         key += String.format("%8s", Integer.toBinaryString(strKey.charAt(i))).replace(' ', '0');
295     }
296 }
297 public void setMessage(String m) {
298     strMsg = m;
299 }
300
301 public void setKey(String k) {
302     strKey = k;
303 }
304
305 private String permute(String text, int[] permutationBox) {
306     String res = "";
307     for(int i = 0; i < permutationBox.length; i++) {
308         res += text.charAt(permutationBox[i]-1);
309     }
310     return res;
311 }
312
313 private String xor(String a, String b) {
314     String res = "";
315     for(int i = 0; i < a.length(); i++) {
316         int temp1 = a.charAt(i)-48;
317         int temp2 = b.charAt(i)-48;
318         int xor = temp1 ^ temp2;
319         res += xor;
320     }
321     return res;
322 }
323 }

```

Listing A.2: DESDecryption.java

```

1
2 package DES;
3
4 import java.util.ArrayList;
5
6 /**
7  * @author Samuel Christian
8  */
9 public class DESDecryption {
10
11     /**
12      * ATTRIBUTES
13      */
14
15     //PC1 -> subkey permutation box
16     private int[] PC1 = {
17         57, 49, 41, 33, 25, 17, 9,
18         1, 58, 50, 42, 34, 26, 18,
19         10, 2, 59, 51, 43, 35, 27,
20         19, 11, 3, 60, 52, 44, 36,
21         63, 55, 47, 39, 31, 23, 15,
22         7, 62, 54, 46, 38, 30, 22,
23         14, 6, 61, 53, 45, 37, 29,
24         21, 13, 5, 28, 20, 12, 4
25     };
26
27
28     //PC2 -> subkey permutation final box
29     private int[] PC2 = {
30         14, 17, 11, 24, 1, 5,
31         3, 28, 15, 6, 21, 10,
32         23, 19, 12, 4, 26, 8,
33         16, 7, 27, 20, 13, 2,
34         41, 52, 31, 37, 47, 55,
35         30, 40, 51, 45, 33, 48,
36         44, 49, 39, 56, 34, 53,
37         46, 42, 50, 36, 29, 32
38     };
39
40     //initial permutation

```



```

41 private int[] IP = {
42     58, 50, 42, 34, 26, 18, 10, 2,
43     60, 52, 44, 36, 28, 20, 12, 4,
44     62, 54, 46, 38, 30, 22, 14, 6,
45     64, 56, 48, 40, 32, 24, 16, 8,
46     57, 49, 41, 33, 25, 17, 9, 1,
47     59, 51, 43, 35, 27, 19, 11, 3,
48     61, 53, 45, 37, 29, 21, 13, 5,
49     63, 55, 47, 39, 31, 23, 15, 7
50 };
51
52 //expansion P-box
53 private int[] exp = {
54     32, 1, 2, 3, 4, 5,
55     4, 5, 6, 7, 8, 9,
56     8, 9, 10, 11, 12, 13,
57     12, 13, 14, 15, 16, 17,
58     16, 17, 18, 19, 20, 21,
59     20, 21, 22, 23, 24, 25,
60     24, 25, 26, 27, 28, 29,
61     28, 29, 30, 31, 32, 1
62 };
63
64 //s-boxes
65 private int[][] s1 = {
66     {14, 4, 14, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
67     {0, 15, 7, 4, 14, 2, 13, 10, 3, 6, 12, 11, 9, 5, 3, 8},
68     {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},
69     {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13}
70 };
71 private int[][] s2 = {
72     {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
73     {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
74     {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
75     {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}
76 };
77 private int[][] s3 = {
78     {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
79     {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
80     {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
81     {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12}
82 };
83 private int[][] s4 = {
84     {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
85     {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
86     {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
87     {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
88 };
89 private int[][] s5 = {
90     {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
91     {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
92     {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
93     {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3}
94 };
95 private int[][] s6 = {
96     {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
97     {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
98     {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
99     {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 10, 0, 8, 13}
100 };
101 private int[][] s7 = {
102     {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
103     {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
104     {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
105     {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12}
106 };
107 private int[][] s8 = {
108     {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
109     {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 10, 14, 9, 2},
110     {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
111     {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}
112 };
113
114 //straight permutation box after s-box substitution
115 private int[] P = {
116     16, 7, 20, 21,
117     29, 12, 28, 17,
118     1, 15, 23, 26,
119     5, 18, 31, 10,
120     2, 8, 24, 14,
121     32, 27, 3, 9,
122     19, 13, 30, 6,
123     22, 11, 4, 25
124 };
125
126 //final permutation
127 private int[] IP1 = {
128     40, 8, 48, 16, 56, 24, 64, 32,
129     39, 7, 47, 15, 55, 23, 63, 31,
130     38, 6, 46, 14, 54, 22, 62, 30,
131     37, 5, 45, 13, 53, 21, 61, 29,
132     36, 4, 44, 12, 52, 20, 60, 28,
133     35, 3, 43, 11, 51, 19, 59, 27,

```

```

140         34, 2, 42, 10, 50, 18, 58, 26,
141         33, 1, 41, 9, 49, 17, 57, 25
142     };
143
144     private String key;
145     private String[] roundKey;
146     private String cipher;
147     private ArrayList<int[][]> sBox;
148
149     public DESDecryption() {
150         key = "";
151         roundKey = new String[16];
152         cipher = "";
153
154         sBox = new ArrayList<int[][]>();
155         sBox.add(s1);
156         sBox.add(s2);
157         sBox.add(s3);
158         sBox.add(s4);
159         sBox.add(s5);
160         sBox.add(s6);
161         sBox.add(s7);
162         sBox.add(s8);
163     }
164
165     public void reset() {
166         key = "";
167         roundKey = new String[16];
168         cipher = "";
169     }
170
171     public void setCipher(String c) {
172         String bin = "";
173         for(int i = 0; i < c.length(); i++) {
174             int hex = Integer.parseInt(c.charAt(i)+ "", 16);
175             String temp = String.format("%4s", Integer.toBinaryString(hex)).replace(' ', '0');
176             bin += temp;
177         }
178         cipher = bin;
179     }
180
181     public void setKey(String k) {
182         for(int i = 0; i < k.length(); i++) {
183             key += String.format("%8s", Integer.toBinaryString(k.charAt(i))).replace(' ', '0');
184         }
185     }
186
187     public String decrypt() {
188         createSubKey();
189         String plainText = "";
190
191         String[] cipherBlock = cipher.split("(?<=\\G.{64})");
192
193         for(int block = 0; block < cipherBlock.length; block++) {
194             String init = initialPermutation(cipherBlock[block]);
195             String L0 = init.substring(0, init.length()/2);
196             String R0 = init.substring(init.length()/2, init.length());
197
198             //16 rounds inverse
199             String[] arr = {L0, R0};
200             for(int i = 15; i >= 0; i--) {
201                 arr = round(arr[0], arr[1], roundKey[i]);
202             }
203
204             String tempPlainText = arr[1] + arr[0];
205             tempPlainText = permute(tempPlainText, IP1);
206             plainText += tempPlainText;
207         }
208
209         return plainText;
210     }
211
212     private String[] round(String left, String right, String rndKey) {
213         String Ln = right;
214         String Rn = xor(left, feistelCipher(right, rndKey));
215         String[] arr = {Ln, Rn};
216         return arr;
217     }
218
219     private String feistelCipher(String right, String rndKey) {
220         String res = "";
221
222         res = xor(rndKey, permute(right, exp));
223
224         String[] B = res.split("(?<=\\G.{6})");
225         String temp = "";
226         for(int i = 0; i < sBox.size(); i++) {
227             String index = B[i];
228             int[][] sbox = sBox.get(i);
229             int row = Integer.parseInt(index.charAt(0) + "" + index.charAt(index.length()-1) + "", 2);
230             int column = Integer.parseInt(index.substring(1, index.length()-1), 2);
231             temp += String.format("%4s", Integer.toBinaryString(sbox[row][column])).replace(' ', '0');
232         }
233
234         res = permute(temp, P);
235
236         return res;
237     }
238

```

```

239 private String initialPermutation(String msgblock) {
240     String res = permute(msgblock, IP);
241     return res;
242 }
243
244 private void createSubKey() {
245     String K2 = permute(key, PC1);
246     String left = K2.substring(0, K2.length()/2);
247     String right = K2.substring(K2.length()/2, K2.length());
248     for(int i = 0; i < 16; i++) {
249         int shift = 2;
250         int round = i + 1;
251         if(round == 1 || round == 2 || round == 9 || round == 16) {
252             shift = 1;
253         }
254         String leftN = leftShift(left, shift);
255         String rightN = leftShift(right, shift);
256         roundKey[i] = leftN+rightN;
257         left = leftN;
258         right = rightN;
259     }
260
261     for(int i = 0; i < roundKey.length; i++) {
262         roundKey[i] = permute(roundKey[i], PC2);
263     }
264 }
265
266 private String leftShift(String text, int bit) {
267     String res = "";
268     char[] ch = new char[text.length()];
269     for(int i = 0; i < ch.length; i++) {
270         int shift = (i-bit) < 0 ? text.length()+(i-bit) : (i-bit);
271         ch[shift] = text.charAt(i);
272     }
273     res = new String(ch);
274     return res;
275 }
276
277 private String permute(String text, int[] permutationBox) {
278     String res = "";
279     for(int i = 0; i < permutationBox.length; i++) {
280         res += text.charAt(permutationBox[i]-1);
281     }
282     return res;
283 }
284
285 private String xor(String a, String b) {
286     String res = "";
287     for(int i = 0; i < a.length(); i++) {
288         int temp1 = a.charAt(i)-48;
289         int temp2 = b.charAt(i)-48;
290         int xor = temp1 ^ temp2;
291         res += xor;
292     }
293     return res;
294 }
295
296 public String binToStr(String bin) {
297     String text = "";
298     String[] temp = bin.split("(?<=\\G.{8})");
299     for(int i = 0; i < temp.length; i++) {
300         text += (char) Integer.parseInt(temp[i], 2);
301     }
302     return text;
303 }
304 }

```

Listing A.3: DataReader.java

```

1 package ReaderWriter;
2
3
4 import java.io.BufferedReader;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.ArrayList;
8
9 /**
10  *
11  * @author Samuel Christian
12  */
13 public class DataReader {
14
15     /**
16      * ATTRIBUTES
17      */
18     private String filename;
19     private ArrayList<String> content;
20
21     public DataReader(String file) {
22         filename = file;
23         content = new ArrayList<String>();
24     }
25
26     public String[] get() {
27         String[] str = new String[content.size()];
28         for(int i = 0; i < str.length; i++) {
29             str[i] = content.get(i);

```

```

30     }
31     return str;
32 }
33
34 public void read() {
35     BufferedReader br = null;
36     try {
37         String sCurrentLine;
38         br = new BufferedReader(new FileReader(filename));
39         while ((sCurrentLine = br.readLine()) != null) {
40             content.add(sCurrentLine);
41         }
42     } catch (IOException e) {
43     } finally {
44         try {
45             if (br != null) {
46                 br.close();
47             }
48         } catch (IOException ex) {
49         }
50     }
51 }
52 }

```

Listing A.4: DataWriter.java

```

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

```

Listing A.5: Sha512.java

[illegible]

```

26  /*Constants for 80 rounds in sha512*/
27  private static final long[] CONSTANTS =
28      {0x428a2f98d728ae22L, 0x7137449123ef65cdL, 0xb5c0fbcfec4d3b2fL, 0
        xe9b5dba58189dbbcL,
29      0x3956c25b348b538L, 0x59f111f1b605d019L, 0x923f82a4af194f9bL, 0
        xab1c5ed5da6d8118L,
30      0xd807aa98a3030242L, 0x12835b0145706fbeL, 0x243185be4ee4b28cL, 0
        x550c7dc3d5ffb4e2L,
31      0x72be5d74f27b896fL, 0x80deb1fe3b1696b1L, 0x9bdc06a725c71235L, 0
        xc19bf174cf692694L,
32      0xe49b69c19ef14ad2L, 0xefbe4786384f25e3L, 0x0fc19dc68b8cd5b5L, 0
        x240ca1cc77ac9c65L,
33      0x2de92c6f592b0275L, 0x4a7484aa6ea6e483L, 0x5cb0a9dcbd41fbd4L, 0
        x76f988da831153b5L,
34      0x983e5152ee66dfabL, 0xa831c66d2db43210L, 0xb00327c898fb213fL, 0
        xbf597fc7beef0ee4L,
35      0xc6e00bf33da88fc2L, 0xd5a79147930aa725L, 0x06ca6351e003826fL, 0
        x142929670a0e6e70L,
36      0x27b70a8546d22ffcL, 0x2e1b21385c26c926L, 0x4d2c6dfc5ac42aedL, 0
        x53380d139d95b3dfL,
37      0x650a73548baf63deL, 0x766a0abb3c77b2a8L, 0x81c2c92e47edaee6L, 0
        x92722c851482353bL,
38      0xa2bfe8a14cf10364L, 0xa81a664bbca42300L, 0xc24b8b70d0f89791L, 0
        xc76c51a30654be30L,
39      0xd192e819d6ef5218L, 0xd69906245565a910L, 0xf40e35855771202aL, 0
        x106aa07032bbd1b8L,
40      0x19a4c116b8d2d0c8L, 0x1e376c085141ab53L, 0x2748774cdf8eeb99L, 0
        x34b0bc5e19b48a8L,
41      0x391c0cb3c5c95a63L, 0x4ed8aa4ae3418acbL, 0x5b9cca4f7763e373L, 0
        x682e6ff3d6b2b8a3L,
42      0x748f82ee5defb2fcL, 0x78a5636f43172f60L, 0x84c87814a1f0ab72L, 0
        x8cc702081a6439ecL,
43      0x90bffffa23631e28L, 0xa4506cebd82bde9L, 0xbef9a3f7b2c67915L, 0
        xc67178f2e372532bL,
44      0xca273ceea26619cL, 0xd186b8c721c0c207L, 0xeda7dd6cde0eb1eL, 0
        xf57d4f7fee6ed178L,
45      0x06f067aa72176fbaL, 0x0a637dc5a2c898a6L, 0x113f9804bef90daeL, 0
        x1b710b35131c471bL,
46      0x28db77f523047d84L, 0x32caab7b40c72493L, 0x3c9ebe0a15c9babcL, 0
        x431d67c49c100d4cL,
47      0x4cc5d4becb3e42b6L, 0x597f299cfc657e2aL, 0x5fcb6fab3ad6faecL, 0
        x6c44198c4a475817L };
48
49  private String originalMessage;
50  private String lengthMessage;
51  private String paddingMessage;
52  private String message;
53  private String[] messageBlock;
54  private String digest;
55
56  public Sha512() {
57      message = "";
58      lengthMessage = "";
59      paddingMessage = "";
60      originalMessage = "";
61      digest = "";
62  }
63
64  public void setMessage(String m) {
65      reset();
66      message = m;
67  }
68
69  public void reset() {
70      message = "";
71      lengthMessage = "";
72      paddingMessage = "";
73      originalMessage = "";
74      digest = "";
75  }
76
77  public String getDigest() {
78      String text = "";
79      for(int i = 0; i < digest.length(); i+=4) {
80          String bits = digest.substring(i, i+4);
81          text += Long.toHexString(longValue(bits));
82      }
83      return text;
84  }
85
86  public void createDigest() {
87      initialize(message);
88      String[] finalDigest = new String[INITIALS.length];
89      for(int countBlock = 0; countBlock < messageBlock.length; countBlock++) {
90          String block = messageBlock[countBlock];
91          String[] words = wordsExpansion(block);
92          String[] first = finalDigest;
93
94          //initialize first digest
95          if(countBlock == 0) {
96              for(int i = 0; i < finalDigest.length; i++) {
97                  finalDigest[i] = pad(Long.toBinaryString(INITIALS[i]));
98              }
99          }
100
101          //80 rounds
102          for(int i = 0; i < words.length; i++) {
103              finalDigest = round(finalDigest, longValue(words[i]), i);
104          }

```

```

105         //final adding
106         for(int i = 0; i < finalDigest.length; i++) {
107             finalDigest[i] = pad(Long.toBinaryString(longValue(finalDigest[i]) + longValue(first[i])))
108         }
109     }
110 }
111 for(int i = 0; i < finalDigest.length; i++) {
112     digest += finalDigest[i] + " ";
113 }
114 }
115
116 private String[] wordsExpansion(String block) {
117     String[] words = new String[80];
118     for(int i = 0; i < 16; i++) {
119         int start = i * 64;
120         int end = start + 64;
121         words[i] = block.substring(start, end);
122     }
123     for(int i = 16; i < words.length; i++) {
124         //i-16
125         long w1 = longValue(words[i-16]);
126
127         //RotShift_1-8-7(i-15)
128         long w2 = rotShift(longValue(words[i-15]), 1, 8, 7);
129
130         //i-7
131         long w3 = longValue(words[i-7]);
132
133         //RotShift_19-61-6(i-2)
134         long w4 = rotShift(longValue(words[i-2]), 19, 61, 6);
135
136         long temp = w1 ^ w2 ^ w3 ^ w4;
137         words[i] = pad(Long.toBinaryString(temp));
138     }
139 }
140
141 return words;
142 }
143
144 //preDigest: previous digest or initial digest
145 //prevDigest[0] -> A    prevDigest[1] -> B    prevDigest[2] -> C    prevDigest[3] -> D
146 //prevDigest[4] -> E    prevDigest[5] -> F    prevDigest[6] -> G    prevDigest[7] -> H
147 private String[] round(String[] prevDigest, long word, int numOfRounds) {
148     String[] result = new String[8];
149     result[1] = pad(prevDigest[0]); //A -> B
150     result[2] = pad(prevDigest[1]); //B -> C
151     result[3] = pad(prevDigest[2]); //C -> D
152     result[5] = pad(prevDigest[4]); //E -> F
153     result[6] = pad(prevDigest[5]); //F -> G
154     result[7] = pad(prevDigest[6]); //G -> H
155
156     //compute Y -> E (prevDigest[4])
157     //conditional(E,F,G)
158     long cond = conditional(longValue(prevDigest[4]), longValue(prevDigest[5]), longValue(prevDigest[6]));
159     //rotate(E)
160     long rot_E = rotate(longValue(prevDigest[4]));
161     //addition modulo 2^64
162     //temp1 = H + cond(E,F,G) + rotate(E) + w_i + K_i
163     long temp1 = longValue(prevDigest[7]) + cond + rot_E + word + CONSTANTS[numOfRounds];
164     //Y = temp1 + D
165     long Y = temp1 + longValue(prevDigest[3]);
166     //Y -> E
167     result[4] = pad(Long.toBinaryString(Y));
168
169     //compute X -> A (prevDigest[0])
170     //majority(A,B,C)
171     long maj = majority(longValue(prevDigest[0]), longValue(prevDigest[1]), longValue(prevDigest[2]));
172     //rotate(A)
173     long rot_A = rotate(longValue(prevDigest[0]));
174     //temp2 = majority(A,B,C) + rotate(A)
175     long temp2 = maj + rot_A;
176     //X = temp1 + temp2
177     long X = temp1 + temp2;
178     //X -> A
179     result[0] = pad(Long.toBinaryString(X));
180 }
181
182 return result;
183 }
184
185 /*WORD EXPANSION FUNCTIONS*/
186 //RotShift_l-m-n(x) = RotR_l(x) XOR RotR_m(x) XOR ShL_n(x)
187 private long rotShift(long x, long l, long m, long n) {
188     long res = rotR(x, l) ^ rotR(x, m) ^ shL(x, n);
189     return x;
190 }
191 //Rotate right x by n bits
192 private long rotR(long x, long n) {
193     return (((x) >>> n) | ((x) << (64 - n)));
194 }
195 //Shift left x by n bits
196 private long shL(long x, long n) {
197     return x << n;
198 }
199
200 /*ROUND FUNCTIONS*/
201 //MAJORITY(x,y,z) = (x AND y) XOR (y AND z) XOR (z AND x)
202 private long majority(long x, long y, long z) {

```

```

202     return ((x & y) ^ (y & z) ^ (z & x));
203 }
204 //CONDITIONAL(x,y,z) = (x AND y) XOR (NOT x AND z)
205 private long conditional(long x, long y, long z) {
206     return ((x & y) ^ ((~x) & z));
207 }
208 //ROTATE(x) = RotR_28(x) XOR RotR_34(x) XOR RotR_29(x)
209 private long rotate(long x) {
210     long res = rotR(x,28) ^ rotR(x,34) ^ rotR(x,29);
211     return res;
212 }
213 /*END*/
214
215 private long longValue(String s) {
216     return new BigInteger(s,2).longValue();
217 }
218
219 private void initialize(String msg) {
220     for(int i = 0; i < msg.length(); i++) {
221         originalMessage += Integer.toBinaryString(msg.charAt(i));
222     }
223
224     lengthMessage = Integer.toBinaryString(originalMessage.length());
225     if(lengthMessage.length() != 128) {
226         int addBit = 128 - lengthMessage.length();
227         String add = "";
228         for(int i = 0; i < addBit; i++) {
229             add += "0";
230         }
231         lengthMessage = add + lengthMessage;
232     }
233
234     paddingMessage = "1";
235     int paddingLength = (((-originalMessage.length() - 128) % 1024 + 1024) % 1024) - 1;
236     for(int i = 0; i < paddingLength; i++) {
237         paddingMessage += "0";
238     }
239
240     message = originalMessage + paddingMessage + lengthMessage;
241
242     messageBlock = new String[message.length()/1024];
243     for(int i = 0; i < messageBlock.length; i++) {
244         int start = i*1024;
245         int end = start+1024;
246         messageBlock[i] = message.substring(start, end);
247     }
248 }
249
250 private String pad(String input) {
251     String res = "";
252     int add = 0;
253     if(input.length() % 8 != 0) {
254         add = 8 - (input.length() % 8);
255     }
256     for(int i = 0; i < add; i++) {
257         res += "0";
258     }
259     res += input;
260     return res;
261 }
262 }

```

Listing A.6: Function.java

```

1
2 package ShamirSecretSharing;
3
4 /**
5  *
6  * @author Samuel Christian
7  */
8 public class Function {
9
10     /**
11     * ATTRIBUTES
12     */
13     private int[] function;
14
15     /**
16     * CONSTRUCTOR
17     */
18     public Function(int[] func) {
19         function = func;
20     }
21
22     /**
23     * OTHERS
24     */
25     //f(x) = res
26     public int countFunction(int x) {
27         int res = 0;
28         for(int i = 0; i < function.length; i++) {
29             res += function[i] * Math.pow(x,i);
30         }
31         return res;
32     }
33 }

```

Listing A.7: EquationSolver.java

```

1 package ShamirSecretSharing;
2
3 /**
4  *
5  * @author Samuel Christian
6  */
7 public class EquationSolver {
8
9     /**
10      * ATTRIBUTES
11      */
12     private double[][] equationMatrix;
13     private double[] solutionMatrix;
14
15     public EquationSolver(double[][] equation, double[] solution) {
16         equationMatrix = equation;
17         solutionMatrix = solution;
18     }
19
20     public void reset() {
21         equationMatrix = new double[1][1];
22         solutionMatrix = new double[1];
23     }
24
25     public double[] solve() {
26         int n = solutionMatrix.length;
27         for (int k = 0; k < n; k++) {
28             /*set pivot row*/
29             int max = k;
30             for (int row = k+1; row < n; row++) {
31                 if (Math.abs(equationMatrix[row][k]) > Math.abs(equationMatrix[max][k])) {
32                     max = row;
33                 }
34             }
35
36             /*swap with pivot row matrix A*/
37             double[] temp = equationMatrix[max];
38             equationMatrix[max] = equationMatrix[k];
39             equationMatrix[k] = temp;
40
41             /*swap solution matrix B*/
42             double tmp = solutionMatrix[max];
43             solutionMatrix[max] = solutionMatrix[k];
44             solutionMatrix[k] = tmp;
45
46             /*set matrix into triangle matrix*/
47             for (int row = k+1; row < n; row++) {
48                 double factor = equationMatrix[row][k] / equationMatrix[k][k];
49                 solutionMatrix[row] -= factor * solutionMatrix[k];
50                 for (int col = k; col < n; col++) {
51                     equationMatrix[row][col] -= factor * equationMatrix[k][col];
52                 }
53             }
54         }
55
56         /*substitution to find solution*/
57         double[] solution = new double[n];
58
59         for (int row = n - 1; row >= 0; row--) {
60             double sum = 0.0;
61             for (int j = row + 1; j < n; j++) {
62                 sum += equationMatrix[row][j] * solution[j];
63             }
64             solution[row] = (solutionMatrix[row] - sum) / equationMatrix[row][row];
65         }
66         return solution;
67     }
68 }
69 }

```

Listing A.8: SecretSharing.java

```

1 package ShamirSecretSharing;
2
3 import java.util.ArrayList;
4
5 /**
6  *
7  * @author Samuel Christian
8  */
9 public class SecretSharing {
10
11     /**
12      * ATTRIBUTES
13      */
14     private int[] function;
15     private int[] fx;
16     private int secret;
17     private EquationSolver solver;
18
19     /**
20      * CONSTRUCTOR
21      */
22     public SecretSharing(int s) {
23         secret = s;
24     }
25 }

```



```

25 |     }
26 |     public SecretSharing() {
27 |     }
28 |
29 |     /**
30 |      * ACCESSOR
31 |      */
32 |     public int[] getFunction() {
33 |         return function;
34 |     }
35 |     public int[] getFx() {
36 |         return fx;
37 |     }
38 |
39 |     /**
40 |      * OTHERS
41 |      */
42 |     public void split(int share) {
43 |         int k = generateK(share);
44 |         function = new int[k];
45 |         function[0] = secret;
46 |         for(int i = 1; i < function.length; i++) {
47 |             function[i] = (int)(Math.random()*50) + 1;
48 |         }
49 |
50 |         Function fcount = new Function(function);
51 |         //f(0) is secret
52 |         fx = new int[share+1];
53 |         for(int i = 0; i <= share; i++) {
54 |             fx[i] = fcount.countFunction(i);
55 |         }
56 |     }
57 |
58 |     private int generateK(int n) {
59 |         int k = n/2 + 1;
60 |         if(k > n) {
61 |             k = n;
62 |         }
63 |         return k;
64 |     }
65 |
66 |     public String reconstruct(int n, int k, ArrayList<double[]> parts) {
67 |         ArrayList<double[]> functions = new ArrayList<double[]>();
68 |         for(int a = 1; a <= n; a++) {
69 |             double[] f = new double[k];
70 |             for(int x = 0; x < f.length; x++) {
71 |                 f[x] = Math.pow(a, x);
72 |             }
73 |             functions.add(f);
74 |         }
75 |
76 |         double[][] equation = new double[k][k];
77 |         int idx = 0;
78 |         double[] tempPasswordPart = parts.get(0);
79 |         for(int i = 0; i < tempPasswordPart.length; i++) {
80 |             if(tempPasswordPart[i] != -1 && idx < equation.length) {
81 |                 equation[idx] = functions.get(i);
82 |                 idx++;
83 |             }
84 |         }
85 |
86 |         int[] finalPart = new int[parts.size()];
87 |         for(int i = 0; i < parts.size(); i++) {
88 |             double[] temp = parts.get(i);
89 |             double[] solution = new double[k];
90 |             double[][] tempEq = new double[k][k];
91 |             for(int row = 0; row < equation.length; row++) {
92 |                 for(int col = 0; col < equation[row].length; col++) {
93 |                     tempEq[row][col] = equation[row][col];
94 |                 }
95 |             }
96 |
97 |             idx = 0;
98 |             for(int c = 0; c < temp.length; c++) {
99 |                 if(temp[c] != -1 && idx < solution.length) {
100 |                     solution[idx] = temp[c];
101 |                     idx++;
102 |                 }
103 |             }
104 |             solver = new EquationSolver(tempEq, solution);
105 |             finalPart[i] = (int)solver.solve()[0];
106 |         }
107 |
108 |         String forgottenPassword = "";
109 |         for(int i = 0; i < finalPart.length; i++) {
110 |             char ch = (char)finalPart[i];
111 |             forgottenPassword += ch + "";
112 |         }
113 |         return forgottenPassword;
114 |     }
115 | }

```

Listing A.9: index.jsp

```

1 |
2 | <%--
3 | Document : index
4 | Author : Sam

```

```

5 <!--%>
6
7 <%@page contentType="text/html" pageEncoding="windows-1252"%>
8 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
9 "http://www.w3.org/TR/html4/loose.dtd">
10
11 <html>
12 <head>
13 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
14 <title>Secret Sharing</title>
15 <link href="css/bootstrap.min.css" rel="stylesheet">
16 <link href="css/normalize.css" rel="stylesheet">
17 <link href="css/font-awesome.css" rel="stylesheet">
18 <link href="css/style.css" rel="stylesheet">
19 <meta http-equiv="X-UA-Compatible" content="IE=edge">
20 <meta name="viewport" content="width=device-width, initial-scale=1">
21 </head>
22 <body>
23 <div class="container_main-container">
24 <div class="main-menu-links">
25 <a href="pages/retrieve.jsp" class="btn btn-success">Retrieve Password</a>
26 or
27 <a href="pages/store.jsp" class="btn btn-info">Store Password</a>
28 </div>
29 </div>
30 </body>
31 </html>

```

Listing A.10: store.jsp

```

1
2 <!--
3 Document : store
4 Author : Sam
5 --%>
6
7 <%@page import="ReaderWriter.DataReader" %>
8 <%@page contentType="text/html" pageEncoding="windows-1252"%>
9 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
10 "http://www.w3.org/TR/html4/loose.dtd">
11
12 <html>
13 <head>
14 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15 <title>Secret Sharing</title>
16 <link href="css/bootstrap.min.css" rel="stylesheet">
17 <link href="css/normalize.css" rel="stylesheet">
18 <link href="css/font-awesome.css" rel="stylesheet">
19 <link href="css/style.css" rel="stylesheet">
20 <meta http-equiv="X-UA-Compatible" content="IE=edge">
21 <meta name="viewport" content="width=device-width, initial-scale=1">
22 </head>
23 <body>
24 <div class="container">
25 <div class="login-box col-md-10 col-md-offset-1 col-sm-10 col-sm-offset-1">
26 <div class="panel panel-info" style="margin-top: 20px;">
27 <div class="panel-heading">
28 <div class="panel-title">Store Password</div>
29 </div>
30 <form method="POST" action="process.jsp" onsubmit="return count()">
31 <div class="panel-body">
32 <input type="button" style="margin-left: 12px;" class="btn btn-primary" id="
33 addPassword" value="Add Password"/>
34 <hr class="pswd-hr"/>
35 <div class="form-group">
36 <label style="margin-top: 7px;" class="col-sm-7 control-label">Security
37 Questions</label>
38 </div>
39 <div style="display: none;" id="counter">1</div>
40 <div class="form-group">
41 <div class="col-sm-9"><input type="text" id="question" class="form-control
42 placeholder="Question Here"/></div>
43 <div class="col-sm-2"><input type="button" id="add" class="btn btn-primary
44 value="Add"/></div>
45 </div>
46 <hr class="submit-buttons-hr"/>
47 <div class="form-group">
48 <input type="submit" class="btn btn-info" value="Store"/>
49 <a class="btn btn-danger" href="index.jsp">Cancel</a>
50 </div>
51 </form>
52 </div>
53 </div>
54 </div>
55 <script src="js/jquery-2.1.1.js" type="text/javascript"></script>
56 <script src="js/bootstrap.min.js" type="text/javascript"></script>
57 <script src="js/app.js" type="text/javascript"></script>
58 </body>
59 </html>

```

Listing A.11: process.jsp

```

1
2 <!--
3 Document : process

```

```

4      Author      : Sam
5  --%>
6
7  <%@page import="java.util.Arrays"%>
8  <%@page import="ReaderWriter.DataWriter"%>
9  <%@page import="ReaderWriter.DataReader"%>
10 <%@page import="SHA512.Sha512"%>
11 <%@page import="DES.DESEncryption"%>
12 <%@page import="ShamirSecretSharing.SecretSharing"%>
13 <%@page contentType="text/html" pageEncoding="windows-1252"%>
14 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
15     "http://www.w3.org/TR/html4/loose.dtd">
16
17 <html>
18     <head>
19         <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
20         <title>Secret Sharing</title>
21     </head>
22     <body>
23         <%
24             String[] passwords = request.getParameterValues("password");
25             String[] questions = request.getParameterValues("questions");
26             String[] answers = request.getParameterValues("answer");
27             String path = getServletContext().getRealPath("data");
28
29             int n = questions.length;
30             int k = passwords.length/2+1;
31             if(k <= 3) {
32                 k = 4;
33             }
34
35             int salt = (int)(Math.random()*50) + 10;
36
37             String[] hashValue = new String[answers.length];
38             Sha512 sha = new Sha512();
39             for(int i = 0; i < hashValue.length; i++) {
40                 String message = questions[i] + answers[i] + salt;
41                 sha.setMessage(message);
42                 sha.createDigest();
43                 hashValue[i] = sha.getDigest();
44                 sha.reset();
45             }
46
47             DataWriter dw = null;
48             String writeToFile = "";
49             DESEncryption e = new DESEncryption();
50             for(int count = 0; count < passwords.length; count++) {
51                 String password = passwords[count];
52                 String[] encrypted = new String[password.length()*n];
53
54                 int secret = 0;
55                 int encIdx = 0;
56                 for(int i = 0; i < password.length(); i++) {
57                     secret = (int)(password.charAt(i));
58                     SecretSharing ss = new SecretSharing(secret);
59                     ss.split(n, k);
60
61                     int[] function = ss.getFunction();
62                     int[] shares = ss.getFx();
63
64                     for(int j = 1; j < shares.length; j++) {
65                         e.setMessage(shares[j]+"");
66                         e.setKey(hashValue[j-1]);
67                         e.initialize();
68                         e.encrypt();
69                         encrypted[encIdx] = e.getCipherText();
70                         encIdx++;
71                         e.reset();
72                     }
73                 }
74
75                 //save answers
76                 writeToFile = "";
77                 for(int i = 0; i < encrypted.length; i++) {
78                     writeToFile += encrypted[i] + "\r\n";
79                 }
80                 dw = new DataWriter(path + "\\data_answers_" + count + ".txt");
81                 dw.write(writeToFile);
82             }
83
84             //save questions
85             writeToFile = "";
86             for(int i = 0; i < questions.length; i++) {
87                 writeToFile += questions[i] + "\r\n";
88             }
89             dw = new DataWriter(path + "\\data_questions.txt");
90             dw.write(writeToFile);
91
92             //save salt and min question
93             writeToFile = salt + "\r\n" + k;
94             dw = new DataWriter(path + "\\data_others.txt");
95             dw.write(writeToFile);
96
97             response.sendRedirect("../index.jsp");
98         %>
99     </body>
100 </html>

```

Listing A.12: retrieve.jsp

```

1  <!--
2  Document   : retrieve
3  Author    : Sam
4  -->
5
6
7  <%@page import="ReaderWriter.DataReader"%>
8  <%@page contentType="text/html" pageEncoding="windows-1252"%>
9  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
10     "http://www.w3.org/TR/html4/loose.dtd">
11
12 <html>
13 <head>
14 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15 <title>Secret Sharing</title>
16 <link href="../../css/bootstrap.min.css" rel="stylesheet">
17 <link href="../../css/normalize.css" rel="stylesheet">
18 <link href="../../css/font-awesome.css" rel="stylesheet">
19 <link href="../../css/style.css" rel="stylesheet">
20 <meta http-equiv="X-UA-Compatible" content="IE=edge">
21 <meta name="viewport" content="width=device-width, initial-scale=1">
22 </head>
23 <body>
24 <div class="container">
25 <div class="login-box col-md-10 col-md-offset-1 col-sm-10 col-sm-offset-1">
26 <div class="panel panel-info" style="margin-top: 20px;">
27 <div class="panel-heading">
28 <div class="panel-title">Retrieve Password</div>
29 </div>
30 <form action="retrieve_process.jsp" method="POST">
31 <div class="panel-body">
32 <div class="form-group">
33 <label style="margin-top: 7px;" class="col-sm-7 control-label">Security
34   Questions</label>
35 </div>
36 <%
37   String path = getServletContext().getRealPath("data");
38   DataReader dr = new DataReader(path + "\\data_questions.txt");
39   dr.read();
40   String[] questions = dr.get();
41   for(int i = 0; i < questions.length; i++) {
42     %>
43     <div class="form-group">
44       <label for="num" style="margin-top: 7px;" class="col-sm-1 control-label"><%= (i+1) + ". " %></label>
45       <label for="answer" style="margin-top: 7px;" class="col-sm-7 control-label"><%= questions[i] %></label>
46       <div class="col-sm-4"><input id="ans" name="answer" type="text"
47         class="form-control"/></div>
48       <input type="hidden" name="questions" value="<%= _questions[i] _%>"/
49     </div>
50   }
51   %>
52 </hr>
53 <div class="form-group">
54 <input type="submit" class="btn btn-info" value="Submit"/>
55 <a class="btn btn-danger" href="../../index.jsp">Cancel</a>
56 </div>
57 </form>
58 </div>
59 </div>
60 </body>
61 </html>

```

Listing A.13: retrieve_process.jsp

```

1  <!--
2  Document   : retrieve_process
3  Author    : Sam
4  -->
5
6
7  <%@page import="ShamirSecretSharing.SecretSharing"%>
8  <%@page import="ShamirSecretSharing.EquationSolver"%>
9  <%@page import="java.util.ArrayList"%>
10 <%@page import="java.io.File" filenameFilter"%>
11 <%@page import="java.io.File"%>
12 <%@page import="java.util.Arrays"%>
13 <%@page import="DES.DESDecryption"%>
14 <%@page import="SHA512.Sha512"%>
15 <%@page import="ReaderWriter.DataReader"%>
16 <%@page contentType="text/html" pageEncoding="windows-1252"%>
17 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
18     "http://www.w3.org/TR/html4/loose.dtd">
19
20 <html>
21 <head>
22 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
23 <title>Secret Sharing</title>
24 <link href="../../css/bootstrap.min.css" rel="stylesheet">
25 <link href="../../css/normalize.css" rel="stylesheet">
26 <link href="../../css/font-awesome.css" rel="stylesheet">

```

```

27     <link href="../../css/style.css" rel="stylesheet">
28     <meta http-equiv="X-UA-Compatible" content="IE=edge">
29     <meta name="viewport" content="width=device-width, initial-scale=1">
30 </head>
31 <body>
32     <div class="container">
33         <div class="login-box col-md-8 col-md-offset-2 col-sm-8 col-sm-offset-2">
34             <div class="panel panel-info" style="margin-top:20px;">
35                 <div class="panel-heading">
36                     <div class="panel-title">Retrieve Password</div>
37                 </div>
38                 <div class="panel-body">
39                     <%
40                         String path = getServletContext().getRealPath("data");
41                         File folder = new File(path);
42                         File[] listPassword = folder.listFiles(new FilenameFilter() {
43                             @Override
44                             public boolean accept(File dir, String name) {
45                                 return (name.contains("data_answers") && name.endsWith(".txt")) && (!
46                                     name.contains("case"));
47                             }
48                         });
49                         String[] answers = request.getParameterValues("answer");
50                         String[] questions = request.getParameterValues("questions");
51                         int n = answers.length;
52
53                         DataReader dr = new DataReader(path + "\\data_others.txt");
54                         dr.read();
55                         int salt = Integer.parseInt(dr.get()[0]);
56                         int k = Integer.parseInt(dr.get()[1]);
57
58                         String[] hashValue = new String[answers.length];
59                         Sha512 sha = new Sha512();
60                         for(int i = 0; i < answers.length; i++) {
61                             String msg = questions[i] + answers[i] + salt;
62                             sha.setMessage(msg);
63                             sha.createDigest();
64                             hashValue[i] = sha.getDigest();
65                             sha.reset();
66                         }
67
68                         int[] decryptedAnswers = new int[1];
69                         ArrayList<double[]> passwordPart = new ArrayList<double[]>();
70                         SecretSharing ss = new SecretSharing();
71                         String[] passwords = new String[listPassword.length];
72                         for(int passCount = 0; passCount < listPassword.length; passCount++) {
73                             passwordPart = new ArrayList<double[]>();
74                             dr = new DataReader(listPassword[passCount].getAbsolutePath());
75                             dr.read();
76                             String[] encryptedAnswers = dr.get();
77                             decryptedAnswers = new int[encryptedAnswers.length];
78
79                             DESDecryption d = new DESDecryption();
80                             int count = 0;
81                             for(int j = 0; j < encryptedAnswers.length; j++) {
82                                 d.setCipher(encryptedAnswers[j]);
83                                 d.setKey(hashValue[count]);
84                                 try {
85                                     decryptedAnswers[j] = Integer.parseInt(d.binToStr(d.decrypt()).trim());
86                                 } catch (NumberFormatException nfe) {
87                                     decryptedAnswers[j] = -1;
88                                 }
89                                 d.reset();
90                                 if(count == hashValue.length-1) {
91                                     count = 0;
92                                 } else {
93                                     count++;
94                                 }
95                             }
96
97                             for(int j = 0; j < decryptedAnswers.length; ) {
98                                 double[] temp = new double[n];
99                                 for(int idx = 0; idx < n; idx++) {
100                                     temp[idx] = decryptedAnswers[j];
101                                     j++;
102                                 }
103                                 passwordPart.add(temp);
104                             }
105
106                             passwords[passCount] = ss.reconstruct(n, k, passwordPart);
107                         }
108                     <%>
109                     <%>
110                     for(int i = 0; i < passwords.length; i++) {
111                         <%>
112                         <% if(passwords[i] == "") { %>
113                             <div class="form-group">
114                                 <div class="failed-notification">Password cannot be retrieved</div>
115                             </div>
116                             <% i = passwords.length; %>
117                         <% } else { %>
118                             <div class="form-group">
119                                 <label class="col-sm-4" style="margin-top:7px;">Your password is</label>
120                                 <div class="col-sm-6"><input class="form-control" type="text" value="
121                                     <%= passwords[i]-%>" disabled/></div>

```

```

122         <% } %>
123     <% } %>
124     <div class="form-group">
125         <div class="col-sm-4">
126             <a href="../index.jsp" class="btn btn-primary">Done</a>
127         </div>
128     </div>
129 </div>
130 </div>
131 </div>
132 </div>
133 </body>
134 </html>

```

Listing A.14: app.js

```

1  jQuery(document).ready(function() {
2      jQuery("#addQuestions").click(function() {
3          var obj = jQuery("#selectQuestions_option:selected");
4          var num = +jQuery("#counter").text();
5          if(jQuery("#selectQuestions").has('option').length > 0) {
6              var element = "<div class='form-group'>"
7                  + num + "</label>"
8                  + "<label style='margin-top:7px;' class='col-sm-7'>" + obj.text() + "</label>" +
9                  "<div class='col-sm-4'><input id='quest' _name='answer' _type='text' _class='form-control' _"
10                     + "</div><input name='questions' _type='hidden' _value='" + obj.text() + "'/>";
11              jQuery(".submit-buttons-hr").before(element);
12          }
13          jQuery("#selectQuestions_option:selected").remove();
14          num++;
15          jQuery("#counter").text(num);
16      });
17
18      jQuery("#add").click(function() {
19          var question = jQuery("#question").val();
20          var num = +jQuery("#counter").text();
21          if(question == "") {
22              alert("Empty");
23          } else {
24              var element = "<div class='form-group'>"
25                  + "<label style='margin-top:7px;' class='col-sm-1'>" + num + "</label>"
26                  + "<label style='margin-top:7px;' class='col-sm-6'>" + question + "</label>"
27                  + "<div class='col-sm-4'><input id='quest' _name='answer' _type='text' _class='form-control' _"
28                     + "</div><input name='questions' _type='hidden' _value='" + question + "'/>"
29                  + "<a class='btn btn-danger' onclick='removeDiv(this)' _style='margin-top:3px;'>"
30                  + "<span class='glyphicon glyphicon-remove-sign' _aria-hidden='true'></span>"
31                  + "</a>"
32                  + "<input name='questions' _type='hidden' _value='" + question + "'/>"
33                  + "</div>";
34              jQuery(".submit-buttons-hr").before(element);
35              num++;
36              jQuery("#counter").text(num);
37          }
38      });
39
40      jQuery("#addPassword").click(function() {
41          var element = "<div class='form-group' _style='margin-top:10px;'>"
42                  + "<label for='pswd' _style='margin-top:7px;' class='col-sm-1 control-label'>Password</label>"
43                  + "<label style='margin-top:7px;' class='col-sm-1'></label>"
44                  + "<div class='col-sm-9'><input id='pswd' _name='password' _type='password' _class='form-control' _"
45                     + "</div><input type='password' _required='required'/></div>"
46                  + "<a class='btn btn-danger' onclick='removeDiv(this)' _style='margin-top:3px;'>"
47                  + "<span class='glyphicon glyphicon-remove-sign' _aria-hidden='true'></span>"
48                  + "</a></div>";
49              jQuery(".pswd-hr").before(element);
50          });
51
52      function check() {
53          var share = jQuery(".questions").length;
54          var k = +jQuery(".k-value").val();
55          if(share < k) {
56              return false;
57          } else {
58              return true;
59          }
60      }
61
62      function removeDiv(e) {
63          jQuery(e).closest('div').remove();
64      }
65
66      function count() {
67          var password = jQuery("input#pswd").length;
68          if(password <= 0) {
69              alert("Minimal 1 password");
70              return false;
71          }
72          return true;
73      }

```

Listing A.15: style.css

```
2 | .main-container {
3 |     margin: 0px auto;
4 |     height: 500px;
5 | }
6 | .main-menu-links {
7 |     margin: 0px auto;
8 |     margin-top: 45vh;
9 |     height: 200px;
10 |    width: 600px;
11 |    text-align: center;
12 | }
13 | tr,td {
14 |     padding: 10px;
15 | }
16 |
17 | .password-input {
18 |     width: 350px;
19 | }
20 |
21 | label {
22 |     cursor: pointer !important;
23 | }
24 |
25 | .form-group {
26 |     margin-bottom: 10px !important;
27 |     height: 40px;
28 | }
29 | select.form-control {
30 |     width: 90px;
31 |     margin-bottom: 10px !important;
32 | }
33 |
34 | .failed-notification {
35 |     background-color: #D9534F;
36 |     color: #FFF;
37 |     font-weight: bold;
38 |     padding: 10px;
39 |     border-radius: 5px;
40 |     margin-left: 10px;
41 |     margin-right: 10px;
42 | }
```