IMPLEMENTASI ALGORITME SPECK DAN SHA-3 PADA *DATABASE* REKAM MEDIK

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Hilman Adi Kartika
NIM: 145150200111090

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI ALGORITME SPECK DAN SHA-3 PADA *DATABASE* REKAM MEDIK

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Hilman Adi Kartika
NIM: 145150200111090

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Ari Kusyanti, S.T, M.Sc	Mahendra Data, S.Kom, M.Kom
NIP. 19831228 201803 2 002	NIK. 201503 8601117 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis di sitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 2 Agustus 2018

Hilman Adi Kartika

NIM: 145150200111090

KATA PENGANTAR

Puji dan syukur peneliti panjatkan ke hadirat Allah SWT berkat rahmat dan karunia-Nya peneliti dapat menyelesaikan skripsi yang berjudul "IMPLEMENTASI ALGORITME SPECK DAN SHA-3 PADA DATABASE REKAM MEDIK". Skripsi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer (S.Kom) pada Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya Malang.

Dalam pelaksanaan dan penyusunan skripsi ini telah melibatkan banyak pihak yang sangat membantu, baik berupa bimbingan dan saran sehingga penulisan skripsi ini dapat terselesaikan. Oleh karena itu, peneliti menyampaikan rasa terima kasih sedalam-dalamnya kepada:

1. Ibu Ari Kusyanti, S.T, M.Sc selaku dosen pembimbing I yang telah bersedia membimbing, memberi dukungan, arahan, dan meluangkan waktu kepada peneliti sehingga skripsi ini dapat terselesaikan.
2. Bapak Mahendra Data, S.Kom., M.Kom selaku dosen pembimbing II yang telah bersedia membimbing, memberi dukungan, arahan, dan meluangkan waktu kepada peneliti dalam penyusunan penulisan skripsi ini.
3. Keluarga peneliti, Bapak Wijiatmono, Ibu Suhartiningsih , dan Kakak Mellisa Diah Pitaloka yang tidak pernah berhenti memberikan motivasi, doa, dan arahan kepada peneliti.
4. Teman-teman grup peneliti yaitu Rizki Maulana Akbar, Rizal Dismantoro, Mabda Amnesti Hananto, Syarif Husein, Hasan Sabiq, Rhiezky Arniansya, Bayu Febrian Putera Ammal, Helmy Rafi Nawawi, Bossarito Putro, Uis Yudha Tri Wirawan, dll. Yang telah memberikan dukungan, semangat, bantuan, dan berbagi pengalaman kepada peneliti.
5. Dan semua pihak yang telah banyak membantu dan memberi motivasi dalam penyusunan skripsi ini yang tidak bisa peneliti sebutkan namanya satu per satu.

Penulis menyadari bahwa dalam penulisan masih memilki banyak kekurangan. Karena itu, penulis mengharapkan saran dan kritik yang sifatnya membangun demi kesempurnaannya dan semoga bermanfaat bagi penulis maupun pembaca.

Malang, 2 Agustus 2018

Penulis
hilmanadi1995@gmail.com

ABSTRAK

Pada bidang kesehatan, data rekam medik merupakan data yang dilindungi dikarenakan data tersebut berisi data pribadi tentang riwayat kesehatan yang dimiliki oleh pasien maupun dokter. Namun dalam beberapa tahun terakhir, marak terjadi kasus penyerangan atau *hacking* yang menyerang pada bidang kesehatan. Dalam penelitian ini bertujuan untuk mengamankan data rekam medik pada *database*, salah satu cara mengamankan data tersebut dengan menerapkan algoritme SPECK untuk menjamin kerahasiaan data dan algoritme SHA-3 untuk menjamin keutuhan data. Dalam penerapan terhadap sistem, algoritme SPECK diterapkan pada *field* nama, alamat, keterangan rekam medik, dan gejala pada tabel pasien, sedangkan algoritme SHA-3 diterapkan pada *field password* tabel *user*. Pengujian yang dilakukan pada penelitian ini antara lain, pengujian validasi *test vector*, pengujian *performance* waktu enkripsi dan dekripsi, pengujian fungsionalitas sistem, pengujian validasi enkripsi dan dekripsi, dan pengujian non-fungsionalitas sistem. Hasil pengujian pada penelitian ini antara lain, pada pengujian *test vector ciphertext* yang dihasilkan dengan menggunakan *key* dan *plaintext* sesuai pada jurnal sesuai dengan *output ciphertext* pada *test vector*, waktu yang dihasilkan pada proses enkripsi dan dekripsi cukup cepat berkisar antara 10 – 12 *milisecond*, pada pengujian validasi enkripsi dan dekripsi hasil dari dekripsi sama dengan *plaintext* sebelum di enkripsi, hasil dari pengujian fungsionalitas sistem berjalan sesuai fungsinya, dan hasil dari pengujian non-fungsional sistem dapat berjalan dengan baik pada berbagai web-browser antara lain Microsoft edge, Mozilla firefox, Google Chrome, dan Opera.

Kata kunci: Kriptografi, SPECK, SHA-3, *Database*, Rekam medik, *Confidentiallity*, *Integrity*

ABSTRACT

In the medical field, medical record data is data protected because the data contains personal data about the medical history owned by patients and doctors. But in recent years, rampant cases of attacks or hacking that attack on the health field. In this study aims to secure data records on the database, one way to secure the data by applying the SPECK algorithm to ensure data confidentiality and SHA-3 algorithm to ensure the integrity of the data. In application to the system, the SPECK algorithm applies to the name, address, medical record, and symptom fields of the patient table, while the SHA-3 algorithm is applied to the user table's password field. Tests conducted in this study are, test vector validation testing, performance testing time of encryption and decryption, testing system functionality, testing of encryption and decryption validation, and testing system non-functionality. Test results in this study are, on test vector ciphertext tested by using key and plaintext according to the journal equal with the output ciphertext on test vector, the time generated in the process of encryption and decryption quite quickly ranged from 10 to 12 milisecond, the test the encryption and decryption validation results from the same decryption as the plaintext before the encryption, the results of the system functionality testing runs according to function, and the results of non-functional testing of the system can run well on various web-browsers such as Microsoft edge, Mozilla firefox, Google Chrome , and Opera.

Keywords : Cryptography, SPECK, SHA-3, Database, Medical Record, Confidentiallity, Integrity

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Dalam bidang kesehatan, data rekam medik merupakan data yang rahasia atau dilindungi dikarenakan data tersebut berisi tentang data pribadi tentang riwayat kesehatan pasien yang dimiliki oleh pasien maupun dokter. Data rekam medik merupakan berkas yang berisi catatan dan dokumen tentang identitas pasien, pemeriksaan, pengobatan, tindakan dan pelayanan lain yang telah diberikan kepada pasien (Menteri Kesehatan, 2008), jenis data rekam medik dapat berupa teks, gambar, suara, maupun video. Dengan berkembangnya bidang teknologi informasi dan komunikasi pencatatan data rekam medik di Indonesia yang semula menggunakan cara pencatatan manual beralih menjadi pencatatan yang bisa dilakukan dengan menggunakan teknologi informasi dan komunikasi, yaitu dengan memanfaatkan Rekam Medis Elektronik (RME). Namun dalam beberapa bulan terakhir marak terjadi kasus *hacking* yang terjadi pada berbagai bidang keilmuan salah satunya yaitu bidang kesehatan, sehingga diperlukan sebuah cara untuk melindungi data tersebut, salah satu contoh penyerangan data rekam medik yang dilakukan terjadi pada tiga wilayah Amerika Serikat (AS) dan berhasil membawa 655.000 data pasien, yang kemudian dijual oleh penyerang ke *darkweb* dengan rentang harga 100.000 dolar AS hingga 395.000 dolar AS dalam bentuk mata uang bitcoin (Yusuf, 2016).

Salah satu cara untuk mengatasi permasalahan tersebut dengan menggunakan algoritme kriptografi, masalah di atas diperbaiki oleh Firlhi Kurniawan pada jurnalnya yang berjudul *Analisis dan Implementasi SHA-1 dan SHA-3 pada Sistem Autentikasi Garuda Training Cost*, yang membahas tentang pengujian, membandingkan, analisis, dan ketahanan serangan antara algoritme *hashing* SHA-1 dan SHA3 (Kurniawan, Kusyanti dan Nurwarsito, 2017), selanjutnya penelitian yang dilakukan oleh Agung Sudrajat dan Erwin Gunadhi pada jurnalnya yang berjudul *Pengamanan Data Rekam Medis Pasien Menggunakan Kriptografi Vigenere Cipher*, yang membahas implementasi algoritme *vigenere cipher* pada data rekam medis pasien pada kolom nama dan alamat (Sudrajat dan Gunadhi, 2016), penelitian selanjutnya dilakukan oleh Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, dan Louis Wingers pada jurnalnya yang berjudul *The Simon and Speck Families of Lightweight Block Chipers* yang membahas mendemonstrasikan kinerja yang dapat dicapai dari SIMON dan SPECK dengan menawarkan kinerja tinggi pada sisi *hardware* dan *software* (Beaulieu *et al.*, 2013). Namun dalam beberapa acuan jurnal tersebut mempunyai beberapa permasalahan yaitu pada pengamanan *Autentikasi Garuda Training Cost* yang dilakukan oleh Firlhi Kurniawan hanya melakukan pengamanan dalam *integrity* data, dan pada penelitian yang dilakukan oleh Sudrajat *Pengamanan Data Rekam Medis Pasien Menggunakan Kriptografi Vigenere Cipher* pengamanan data rekam medik hanya dilakukan pada *confidentiallity* data dan menggunakan algoritme kriptografi yang lama.

Kelemahan pada beberapa penelitian sebelumnya, saya perbaiki dengan mengimplementasikan algoritme SPECK untuk menjamin *confidentiallity* dikarenakan pada SPECK mempunyai panjang *key* dan *block size* yang beragam, algoritme SPECK juga termasuk algoritme baru yang diterbitkan oleh NSA pada tahun 2013 dan algoritme SHA-3 untuk menjamin *integrity* data dikarenakan SHA-3 merupakan algoritme standar terbaru untuk *hash*. Dengan latar belakang tersebut, penulis mencoba mengimplementasikan algoritme SPECK dan SHA-3 pada *database* rekam medik.

1.2 Rumusan masalah

Berdasarkan latar belakang permasalahan di atas, rumusan masalah dari penelitian ini:

1. Bagaimana mengimplementasikan algoritme enkripsi SPECK pada *field* nama, alamat, keterangan rekam medik, gejala dan algoritme SHA-3 pada *field password database* rekam medik?
2. Bagaimana hasil validasi *plaintext* dan *ciphertext* dengan *test vector* dari algoritme SPECK?
3. Bagaimana kinerja waktu pemrosesan enkripsi dan dekripsi algoritme SPECK pada *database* rekam medik?

1.3 Tujuan

Berdasarkan permasalahan tersebut, tujuan yang ingin dicapai dari penelitian ini adalah membangun aplikasi yang dapat:

1. Mengetahui cara implementasi algoritme SPECK pada *field* nama, alamat, keterangan rekam medik, dan gejala, dan algoritme SHA-3 pada *field password database* rekam medik
2. Mengetahui hasil validasi *plaintext* dan *ciphertext* dengan *test vector* dari algoritme SPECK
3. Mengetahui hasil kinerja waktu pemrosesan enkripsi dan dekripsi algoritme SPECK pada *database* rekam medik

1.4 Manfaat

Manfaat dari penelitian ini diharapkan dapat menjadi dasar untuk pengamanan data pada *field database*. Sehingga data yang tersimpan pada *database* akan terjamin keamanannya pada aspek *confidentiality* atau kerahasiaan data dan *integrity* atau keutuhan data.

1.5 Batasan masalah

Pada penelitian ini terdapat beberapa batasan masalah agar tidak melenceng dari latar belakang, rumusan masalah, dan tujuan. Berikut merupakan batasan masalah pada penelitian ini :

1. Versi algoritme SPECK yang digunakan berukuran128-bit *block size* dan 128-bit *key*, dan versi algoritme SHA-3 yang digunakan yaitu 224
2. Jumlah maksimal *input* pada *database* yaitu 64-char atau 512-bit pada *field* alamat, keterangan rekam medik, dan gejala, pada *field* nama 16-char atau 128-bit
3. Data yang digunakan berupa data *dummy*
4. Sistem *website* dan *database* menggunakan *localhost*
5. *Input key* dilakukan oleh admin melalui *file* dengan format txt

1.6 Sistematika pembahasan

Sistematika penulisan dalam skripsi ini dibagi menjadi beberapa bab, yang terdiri dari:

BAB I **Pendahuluan**

Pada bab ini berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, dan manfaat penelitian dan sistematika penelitian.

BAB II **Landasan Teori**

Pada bab ini berisi tentang penelitian yang telah dilakukan dan juga berisi dasar teori tentang algoritme SPECK, SHA-3, *database*, Rekam Medik yang akan menjadi bahan sebagai referensi untuk penelitian.

BAB III **Metodologi Penelitian**

Pada bab ini berisi tentang pembahasan mengenai tahapan pengerjaan skripsi dengan judul Implementasi algoritme SHA-3 dan SPECK pada *database* rekam medik. Terdiri dari studi literatur, rekayasa kebutuhan, perancangan, implementasi dan pengujian.

BAB IV **Rekayasa Kebutuhan**

Pada bab ini berisi tentang rekayasa kebutuhan secara umum apa saja yang dibutuhkan dalam implementasi algoritme SHA-3 dan SPECK pada *database* rekam Medik. Meliputi kebutuhan perangkat keras maupun perangkat lunak.

BAB V **Perancangan dan Implementasi**

Pada bab ini berisi tentang mengenai cara untuk merancang algoritme SPECK dan SHA-3 dapat berjalan pada sistem *database* rekam medik. Sedangkan pada bagian implementasi berisi tentang menerapkan algoritme SPECK dan SHA-3 pada *database* rekam medik.

BAB VI **Pengujian**

Pada bab ini berisi tentang hasil penelitian dan pembahasan terkait Analisa dari pengujian yang telah dilakukan.

BAB VII **Penutup**

Pada bab ini berisi tentang kesimpulan yang diperoleh dari rumusan masalah berdasarkan hasil penelitian yang sudah didapat, dan saran yang dapat diberikan untuk melakukan pengembangan penelitian lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Pada sub bab dasar teori akan dijelaskan mengenai teori kriptografi yang merupakan salah satu teori untuk mengamankan data dan membahas algoritme yang akan diimplementasikan yakni algoritme SPECK dan SHA-3 dan rekam medik sebagai objek implementasi dari algoritme tersebut

2.1 Kajian Pustaka

2.1.1 Algoritme SPECK dan SIMON

Dari banyak *block cipher* ringan yang sudah ada, kebanyakan desain yang dilakukan untuk menghasilkan kinerja yang baik dalam satu *platform* dan tidak dimaksudkan untuk menghasilkan kinerja tinggi pada beberapa perangkat. Tujuan pembuatan algoritme SPECK dan SIMON *block cipher* untuk memenuhi kebutuhan dari segi keamanan, fleksibilitas, dan mudah di analisa. Dalam penelitian yang dilakukan pengujian dilakukan dengan membandingkan kinerja antara algoritme SPECK dan SIMON dengan algoritme *block cipher* yang sudah ada seperti AES, KATAN, KLEIN, PICCOLO dan TWINE, kemudian dilakukan pengujian dalam segi kinerja *hardware* yang dilakukan pada VHDL dan *synthetysized*, dan kinerja pada *software* yang dilakukan pada 8-bit *microcontrollers*. (Beaulieu *et al.*, 2013)

Hasil yang didapatkan pada kedua algoritme mempunyai kinerja yang sangat baik pada *platform software* dan *hardware*, cukup fleksibel untuk menerima berbagai implementasi pada *platform* yang diberikan, dan juga dapat dianalisis dengan teknik yang sudah ada. Namun SIMON didesain dengan kinerja optimal pada sisi *hardware*, dan SPECK pada sisi *software*. (Beaulieu *et al.*, 2013)

2.1.2 Penelitian Sebelumnya

Tabel 2.1 merupakan kajian pustaka yang digunakan pada penelitian ini :

Tabel 2. 1 Kajian Pustaka

No	Judul Penelitian	Metodologi	Hasil	Kelemahan
1	*Analisis dan Implementasi Algoritme SHA-1 dan SHA-3 pada Sistem Autentikasi Garuda Training Cost*. (Kurniawan, Kusyanti dan Nurwarsito, 2017)	Dalam penelitian ini cara yang dilakukan oleh peneliti yaitu dengan mengganti algoritme yang sudah ada sebelumnya yaitu SHA-1 dengan SHA-3 pada proses *login* dengan memasukkan	Hasil pada penelitian ini SHA-1 mempunyai nilai *avalanche effect* sebesar 100% dan SHA-3 sebesar 95%, pada pengujian	Pada jurnal tersebut pengimplementasian keamanan hanya pada sisi *integrity* data, maka diperlukan suatu algoritme enkripsi untuk mengamankan pada sisi *confidentiality* data

		username dan *password* setelah memasukkan *username* dan *password* maka akan dilakukan proses *hashing* yang kemudian akan dicocokkan dengan *hash* yang ada pada *database*.	*brute force* SHA-3 mempunyai keunggulan lebih daripada SHA-1 dalam segi waktu.	
2	*Pengamanan Data Rekam Medis Pasien Menggunakan Kriptografi Vigenere Cipher*. (Sudrajat dan Gunadhi, 2016)	Pada penelitian ini cara yang digunakan oleh peneliti yaitu dengan menerapkan algoritme enkripsi *vigenere cipher* pada *field* nama dan alamat pada *database* rekam medik.	Algoritme enkripsi berhasil diterapkan pada sistem *database*, dan *database* lebih aman dari kriptanalis	Pada penelitian ini hanya dilakukan pengamanan dalam sisi *confidentiality* data dan menggunakan algoritme enkripsi yang lama, maka diperlukan pengamanan data pada sisi *integrity* data dan menggunakan algoritme enkripsi terbaru

2.2 Dasar Teori

2.2.1 Kriptografi

Kriptografi berasal dari bahasa Yunani, *crypto* dan *graphia*. Crypto yang berarti *secret* (rahasia) dan Graphia yang mempunyai arti *writing*(tulisan). Menurut terminologinya, kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan ketika pesan dikirim dari suatu tempat ke tempat lain. (Ariyus, 2008)

Pada dasarnya kriptografi mempunyai beberapa komponen seperti: (Ariyus, 2008)

A. Enkripsi: merupakan hal yang sangat penting dalam kriptografi, yang merupakan cara pengamanan data yang dikirimkan sehingga terjaga ke rahasiannya, dengan mengubah *plaintext* menjadi *ciphertext*.
B. Dekripsi: merupakan kebalikan dari proses enkripsi. Pesan yang telah dienkripsi dikembalikan ke bentuk asalnya, yaitu dengan mengubah *ciphertext* atau hasil enkripsi menjadi *plaintext* atau text aslinya.

C. *Key* atau Kunci: digunakan untuk melakukan proses enkripsi maupun proses dekripsi. Kunci terbagi menjadi dua bagian, yaitu kunci rahasia (*private key*), dan kunci umum (*public key*).
D. *Ciphertext*: merupakan suatu pesan yang telah melalui proses enkripsi. Pesan yang ada pada teks-kode ini tidak bisa dibaca karena berupa karakter-karakter yang tidak mempunyai makna (arti).
E. *Plaintext*: merupakan teks-asli atau teks-biasa yang ditulis atau diketik di mana teks tersebut dapat dibaca dan memiliki makna. Teks-asli inilah yang diproses menggunakan algoritme kriptografi untuk dijadikan *ciphertext* (teks-kode).
F. *Pesan*: dapat berupa data atau informasi yang dikirim atau disimpan dalam media perekaman (kertas, *storage*, dsb).
G. *Cryptanalysis*: kriptanalisis bisa diartikan sebagai analisis kode atau suatu ilmu untuk mendapatkan teks-asli tanpa harus mengetahui kunci yang sah secara wajar. Jika suatu teks-atau kode berhasil diubah menjadi teks-asli tanpa menggunakan kunci yang sah. Proses tersebut dinamakan *breaking code*. Hal ini dilakukan oleh para kriptanalisis. Analisis kode juga dapat menemukan kelemahan dari suatu algoritme kriptografi dan akhirnya dapat menemukan kunci atau teks-asli dari teks-kode yang dienkripsi dengan algoritme tertentu.

2.2.2 Algoritme SPECK

Algoritme SPECK merupakan sebuah *block cipher* yang ringan yang sudah dipublikasikan oleh *National Security Agency* (NSA) pada tahun 2013. SPECK menghasilkan kinerja yang sangat baik pada *hardware* maupun *software.* SPECK termasuk ke dalam operasi ARX (*add-rotate-xor*), di mana pada operasi tersebut terdapat penjumlahan *modulo*, rotasi dengan jumlah rotasi yang pasti, dan operasi XOR (*exclusive-or*). (Beaulieu *et al.*, 2013)

Pada algoritme SPECK terdapat parameter penentuan jumlah panjang *block*, panjang *key*, panjang *word*, panjang *key* word, pergeseran bit dan jumlah *round*, seperti yang dijelaskan pada Tabel 2.2

Tabel 2. 2 Parameter algoritme SPECK

Panjang *block* (2n)	Panjang *Key* (mn)	Panjang *Word* (n)	Panjang *Key Word* (m)	Geser bit (α)	Geser bit (β)	Jumlah *round*
32	64	16	4	7	2	22
48	72	24	3	8	3	22
	96		4			23
64	96	32	3	8	3	26
	128		4			27

Tabel 2. 2 Parameter algoritme SPECK (lanjutan)

96	96	48	2	8	3	28
	144		3			29
128	128	64	2	8	3	32
	192		3			33
	256		4			34

2.2.2.1 Round Function

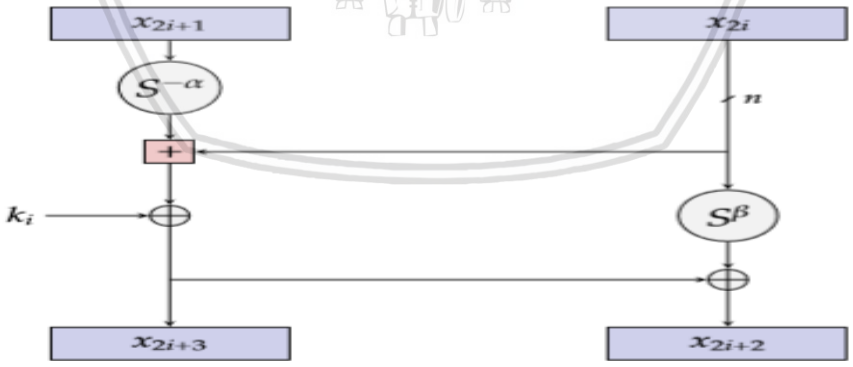
Pada proses enkripsi SPECK menggunakan operasi n-bit *word* dengan notasi operasi sebagai berikut: (Beaulieu *et al.*, 2013)

- XOR setiap bit dengan notasi $\oplus$
- Penjumlahan *modulo* 2^n dengan notasi +
- Geser bit ke kiri dan kanan dengan notasi, S^j dan S^{-j}

Pembentukan *round function* pada proses enkripsi didefinisikan dengan: (Beaulieu *et al.*, 2013)

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k) \quad (1)$$

Pada persamaan 1 $S^{-\alpha}$ merupakan pergeseran bit sejumlah α bit ke kanan pada *plain-text1* (x) dan dijumlahkan dengan *plain-text2* (y) dengan penjumlahan *additional modulo* 2^n, kemudian hasil penjumlahan di XOR dengan *key*, S^{β} melakukan pergeseran bit ke kiri sejumlah β bit pada *plain-text2* dan XOR hasil pergeseran bit dengan proses sebelumnya. Proses ini dilakukan selama sejumlah *round* yang sudah di definisikan pada tabel 2.1. pada Gambar 2.1 merupakan penjelasan proses enkripsi pada *round function*.



Gambar 2. 1 *Round Function*

Sumber: (Beaulieu *et al.*, 2013)

Pembentukan *round function* pada proses dekripsi dengan cara membalik fungsi *round function* pada proses enkripsi, didefinisikan dengan: (Beaulieu *et al.*, 2013)

$$R_k^{-1}(x,y) = (S^{\alpha}((x \oplus k) - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)) \quad (2)$$

Pada persamaan 2 merupakan proses dekripsi untuk mendapatkan nilai dari *plain-text1* yaitu melakukan operasi XOR antara *plain-text1* dengan *key* dikurangi dengan hasil pergeseran bit ke kanan sejumlah β bit terhadap nilai dari *plain-text1* di XOR dengan *plain-text2*, setelah hasil bit dari proses tersebut didapat maka dilakukan pergeseran bit ke kiri sejumlah α bit, untuk mendapatkan hasil dari *plain-text2* dengan melakukan proses XOR antara *plain-text1* dengan *plain-text2* setelah *key schedule*.

2.2.2.2 Key Schedule

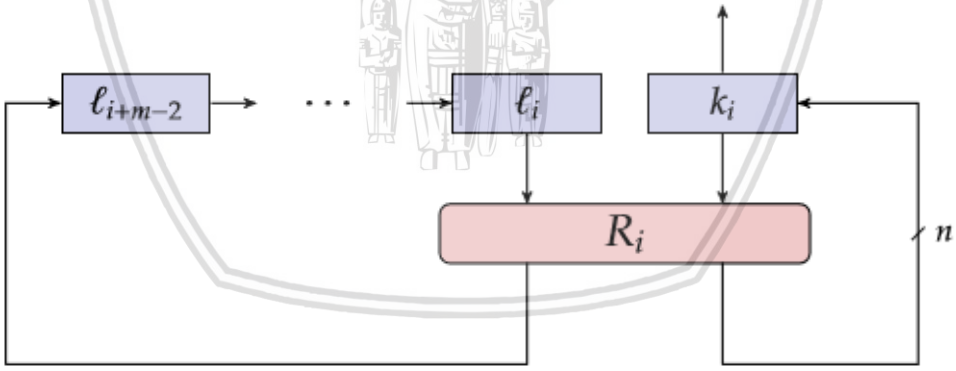
Pada proses *key schedule* pada algoritme SPECK menggunakan *round function* untuk *generate round key* (k_i), *m* digunakan sebagai nomor dari *word* pada *key*. Maka didapatkan 2 *sequences* yaitu k_i dan l_i untuk mencari nilai l_i sebagai berikut: (Beaulieu *et al.*, 2013)

$$l_{i+m-1} = (k_i + S^{-\alpha} l_i) \oplus \mathrm{I} \quad (3)$$

Pada persamaan 3 merupakan pencarian nilai l_i dengan cara k_i di *additional modulo* 2^n menggunakan dengan hasil dari pergeseran bit ke kanan sejumlah α bit. Setelah didapatkan hasilnya di XOR-kan dengan nilai i.
Untuk mencari nilai k_i sebagai berikut: (Beaulieu *et al.*, 2013)

$$k_{i+1} = S^{\beta} k_i \oplus l_{i+m-1} \quad (4)$$

Pada persamaan 4 pencarian nilai dari k_i dengan cara menggeser bit k_i sejumlah β bit ke kiri kemudian di XOR dengan hasil dari l_i, kedua proses tersebut dilakukan sejumlah *round*s sama seperti yang dilakukan pada *round function*.



Gambar 2. 2 *Key Schedule*

Sumber: (Beaulieu *et al.*, 2013)

2.2.3 Algoritme SHA-3

National Institute of Standards and Technology (NIST) merupakan laboratorium standar pengukuran, yang mempunyai misi untuk mempromosikan inovasi dan daya saing industri. Pada tahun 2006, NIST mengadakan sebuah kompetisi yang bernama *NIST hash function competition* untuk membuat standar *hash* baru yang bernama SHA-3. SHA-3 tidak ditujukan untuk menggantikan SHA-2, dikarenakan tidak ada serangan yang signifikan terhadap algoritme SHA-2. NIST bertujuan untuk memberikan algoritme alternatif yang bernama SHA-3,

dikarenakan kekawatiran terhadap algoritme sebelumnya yang berhasil ditembus seperti MD5, SHA-0, SHA-1.

Pada tahun 2012, Keccak menjadi pemenang dalam kompetisi tersebut dengan mengalahkan finalis lainnya seperti BLAKE, JH, Skein dan Grøstl. Kemudian pada tahun 2014 mempublikasikan *draft* FIPS 202 yang berisi tentang "*SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*". Pada Tahun 2015 SHA-3 diresmikan sebagai standar baru fungsi *hash* (Dworkin, 2015).

Pada algoritme SHA-3 mempunyai keluaran beragam, seperti SHA-3 dengan panjang *output* beragam mulai dari 224, 256, 384, dan 512 bit. Pada SHA 3 terdapat juga SHA *key exchange* seperti SHAKE 128 dan SHAKE 256. Dengan berbagai keluaran tersebut algoritme SHA-3 termasuk ke dalam jenis *sponge function.*

2.2.4 *Database*

Database merupakan tempat penyimpanan data, *database* juga mempunyai bahasa program sendiri yang mempunyai kemampuan tidak hanya memanipulasi data tapi juga untuk membangun aplikasi, *tool* untuk mengatur lalu lintas data yaitu bahasa SQL, dan *tool* untuk mengatur manajemen data juga tetap tersedia. Bahkan *tool* untuk manajemen data ini sudah terintegrasi dengan jaringan dan *database* bisa diatur manajemennya melalui LAN, MAN, WAN atau internet. (Kusuma, 2011)

2.2.5 Rekam Medik

Rekam medis adalah berkas yang berisi identitas, anamnesis, penentuan fisik, laboratorium, diagnosa dan tindakan medis terhadap seorang pasien yang dicatat baik secara tertulis maupun elektronik. Bila penyimpanannya secara elektronik maka dibutuhkan manajemen basis data. Pengertian rekam medis bukan hanya sekedar kegiatan pencatatan, pelayanan dan tindakan medis apa saja yang diterima pasien, selanjutnya penyimpanan berkas sampai dengan pengeluaran berkas dari tempat penyimpanan manakala diperlukan untuk kepentingan sendiri maupun untuk keperluan lainnya. (Handiwidjojo, 2009). Gambar 2.3 merupakan salah satu contoh susunan tabel pada *database* rekam medik.

	#	Name	Type	Collation	Attributes	Null	Default
☐	1	**no_rek_med**	int(11)			No	*None*
☐	2	**no_pasien**	varchar(11)	latin1_swedish_ci		Yes	*NULL*
☐	3	**nama_pasien**	varchar(25)	latin1_swedish_ci		Yes	*NULL*
☐	4	**tgl_berobat**	date			Yes	*NULL*
☐	5	**umur**	float			Yes	*NULL*
☐	6	**jenis_kelamin**	varchar(12)	latin1_swedish_ci		Yes	*NULL*
☐	7	**alamat**	varchar(30)	latin1_swedish_ci		Yes	*NULL*
☐	8	**pekerjaan**	varchar(15)	latin1_swedish_ci		Yes	*NULL*
☐	9	**nama_dokter**	varchar(25)	latin1_swedish_ci		Yes	*NULL*
☐	10	**keterangan_rek_med**	varchar(50)	latin1_swedish_ci		Yes	*NULL*
☐	11	**gejala**	varchar(50)	latin1_swedish_ci		Yes	*NULL*

Gambar 2. 3 Contoh *Database* Rekam Medik

Sumber: (Wibisono dan Munawaroh, 2012)

BAB 3 METODOLOGI PENELITIAN

Pada Bab ini membahas mengenai metode apa saja yang digunakan dalam melakukan implementasi algoritme SHA-3 dan SPECK pada *database* rekam medik. Dalam penelitian ini terdapat beberapa metode yang digunakan diawali dengan proses studi literatur sampai penarikan kesimpulan dan saran. Pada penelitian ini menggunakan metode eksperimental, partisipan yang dilibatkan pada penelitian ini hanya peneliti yang berarti dalam pengambilan data, perancangan, implementasi, dan pengujian sistem hanya dilakukan oleh peneliti. Lokasi penelitian dilakukan pada ruang bersama komputasi berbasis jaringan (KBJ) yang terletak pada gedung F lantai 9 Fakultas Ilmu Komputer (FILKOM) Universitas Brawijaya. Pengambilan data dengan menggunakan metode studi dokumen, data yang didapat pada penelitian ini diperoleh pada jurnal, laporan, dan lain-lain.

Gambar 3.1 menjelaskan tentang diagram alir dari metodologi penelitian:

Gambar 3. 1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Literatur yang digunakan untuk menunjang penelitian tentang Implementasi Algoritme SHA-3 dan SPECK pada *database* rekam medik terdapat pada sub-bab studi literatur ini. Teori-teori untuk mendukung penulisan skripsi didapat dari buku, jurnal, situs web resmi, dan penelitian sebelumnya dengan pembahasan yang sama atau berhubungan dengan skripsi. Pokok bahasan dalam

penelitian ini adalah cara untuk mengimplementasikan algoritme SPECK dan SHA-3 pada *database* rekam medik.

3.2 Rekayasa Kebutuhan

Analisis kebutuhan berisi kebutuhan apa saja yang diperlukan dalam pembuatan sistem seperti kebutuhan fungsional, non fungsional dari *hardware* maupun *software* yang akan diimplementasikan dan dijelaskan lebih detail sebagai berikut:

1. Satu Buah Laptop
2. Sistem Operasi Windows 10 Pro 64 bit
3. PyCharm Community Edition 2017.2.4
4. MySQL 10.1.31
5. Python 2.7.13
6. XAMPP 3.2.2
7. Notepad ++ v.7.5.6

3.3 Perancangan dan Implementasi

Perancangan dilakukan untuk memberikan gambaran kerja yang akan dilakukan pada penelitian ini. perancangan juga digunakan untuk mempermudah implementasi yang sesuai dengan penelitian. Teori dari daftar pustaka digabungkan dengan ilmu yang diperoleh, dan selanjutnya diimplementasikan untuk merancang serta mengembangkan algoritme SPECK dan SHA-3 pada *database* rekam medik.

Implementasi dilakukan dengan menerapkan atau merealisasikan hasil perancangan yang telah dibuat. Implementasi diawali dengan konfigurasi dan penyesuaian *database* pada *database* rekam medik, kemudian dilanjutkan dengan menerapkan enkripsi dan dekripsi algoritme SPECK pada *field database* nama, alamat, keterangan rekam medik, dan gejala , kemudian algoritme SHA-3 diterapkan pada *field password* pada *database* rekam medik.

3.4 Pengujian

Pengujian dilakukan dengan 6 tahap, yaitu pengujian *test vector*, pengujian *performance* waktu, pengujian validasi enkripsi dan dekripsi, pengujian keamanan, pengujian fungsionalitas, dan pengujian non-fungsional. Pengujian *test vector* dilakukan untuk mengetahui apakah masukan yang diperoleh telah sesuai dengan *test vector* yang dimiliki. Pengujian *performance* waktu dilakukan untuk mendapatkan waktu yang dibutuhkan sistem untuk melakukan enkripsi dan dekripsi pada *database* rekam medik. Pengujian validasi enkripsi dan dekripsi dilakukan untuk memastikan hasil setelah di dekripsi sama dengan masukan awal sebelum di enkripsi. Pengujian keamanan dilakukan untuk mengetahui apakah sistem setelah diberikan algoritme SPECK enkripsi dan dekripsi dapat

mengamankan data pada *database*. Pengujian fungsionalitas dilakukan untuk menguji apakah semua fungsi pada sistem dapat berjalan dengan baik. Pengujian non-fungsional dilakukan untuk mengetahui kinerja sistem seperti waktu respon sistem dan sistem berjalan pada semua web browser.

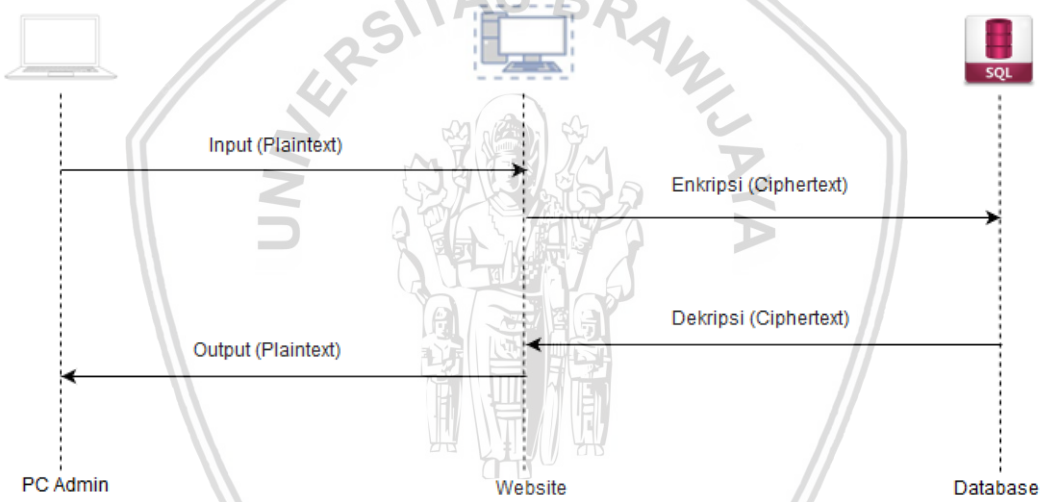
3.5 Kesimpulan

Penarikan kesimpulan dilakukan setelah semua tahapan mulai dari studi literatur, rekayasa kebutuhan, perancangan dan implementasi, pengujian serta analisis sudah selesai dilakukan dan sesuai dengan kebutuhan yang ingin dicapai. Kesimpulan diambil untuk menjawab rumusan masalah terkait dengan hasil implementasi algoritme SPECK dan SHA-3 pada *database* rekam medik. Sedangkan saran dimaksudkan untuk menyempurnakan penelitian serta dapat dijadikan pertimbangan dalam pengembangan penelitian selanjutnya.

BAB 4 REKAYASA KEBUTUHAN

4.1 Gambaran Umum Sistem

Sistem yang dibuat oleh peneliti menggunakan algoritme *block-cipher* yaitu SPECK dengan ukuran 128 bit dengan masukan *plaintext* sepanjang 128-bit dan *key* 128-bit dikarenakan data yang terdapat pada *database* cukup panjang, dan pada *hash password* dengan menggunakan API SHA-3 dengan versi 224-bit. *Database* yang digunakan dirancang dengan menggunakan Maria DB yang terdapat pada *software* XAMPP. Sistem akan melakukan enkripsi *database* dengan menggunakan algoritme SPECK pada *field* nama, alamat, keterangan rekam medik, dan gejala di mana keempat *field* tersebut masing-masing sebagai masukan untuk *plaintext*, untuk masukan *key* didapatkan dari masukan dari administrator melalui *file*. Setelah dilakukan proses enkripsi maka dihasilkan *ciphertext* yang akan digunakan untuk di *update* pada *field database* rekam medik.



Gambar 4. 1 Gambaran Umum Sistem

Penelitian dilakukan untuk menjamin *confidentiality* dengan menggunakan algoritme SPECK dan *integrity* dengan menggunakan algoritme SHA-3 data terhadap *attacker*.

4.2 Kebutuhan Sistem

Pada subbab kebutuhan sistem menjelaskan kebutuhan yang diperlukan untuk membangun sistem yaitu kebutuhan fungsional, selain itu juga berisi tentang kebutuhan *software* dan *hardware* untuk mendukung desain dan implementasi sistem.

4.2.1 Kebutuhan Fungsional

Pada penelitian ini, ada beberapa kebutuhan fungsional, yakni:

1. Algoritme SPECK harus menghasilkan *plaintext* dan *ciphertext* yang sesuai dengan *test vector* pada jurnal pencipta algoritme.
2. Algoritme SHA-3 harus menghasilkan *digest* yang sesuai dengan *test vector* pada jurnal pencipta algoritme.
3. Sistem harus bisa mengenkripsi data sebelum di *update* pada *field database*, dan hasil dekripsi harus sesuai dengan masukan *plaintext* sebelum di enkripsi.

4.2.2 Kebutuhan Non-Fungsional

Pada penelitian ini, ada beberapa kebutuhan fungsional, yakni:

1. Sistem harus dapat berjalan pada semua web browser (*Portability*).
2. Sistem harus dapat melakukan respon permintaan kurang dari 10 detik (*Performance*).

4.2.3 Kebutuhan Perangkat Keras

Pada penelitian ini membutuhkan perangkat keras, yakni:

1. Laptop, dengan spesifikasi sebagai berikut:
   - Model : Asus X455LN
   - *OS* : *Windows 10 Home (64 bit)*
   - *Processor* : *Intel (R) Core (TM) i5-4210U CPU @1.70Ghz (4 CPUs)*
   - *RAM* : 8 GB
   - Jumlah : 1
   - Fungsi : Untuk menjalankan *database* dan bahasa pemrograman

4.2.4 Kebutuhan Perangkat Lunak

Peneliti membutuhkan kurang lebih 2 perangkat lunak *free* pada penelitian ini. Berikut beberapa perangkat lunak yang digunakan beserta spesifikasinya:

1. XAMPP v3.2.2
   XAMPP digunakan untuk server yang dapat berdiri pada sistemnya sendiri (*localhost*), yang terdiri dari program Apache HTTP Server, Maria *database*, dan Penerjemah bahasa PhP dan Perl
2. Maria DB 10.1.31
   Maria DB digunakan untuk perancangan dan implementasi *database* rekam medik
3. Apache 2.4.29
   Apache 2.4.29 digunakan sebagai *web-server* untuk menyediakan layanan permintaan atau pengiriman data menggunakan *website*
4. PyCharm Community Edition 2017.2.4
   PyCharm digunakan untuk bahasa pemrograman yaitu Python
5. Notepad ++ v.7.5.6
   Notepad ++ digunakan untuk bahasa pemrograman yaitu PhP

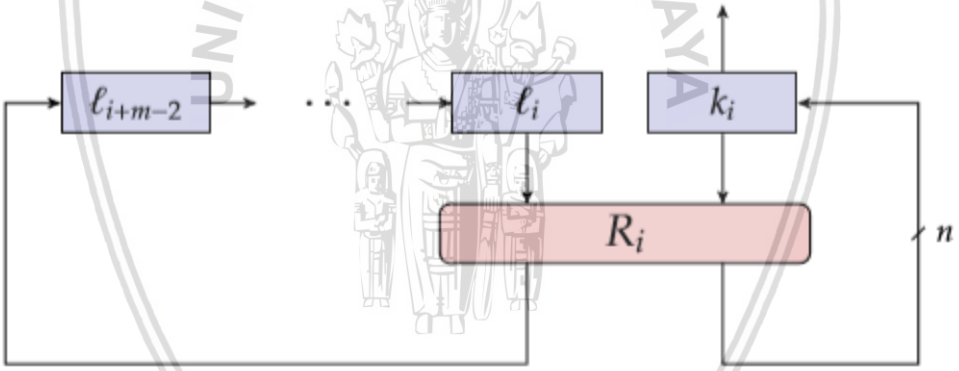
BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan

Pada bab perancangan, peneliti merancang sistem yang akan dibangun, meliputi perancangan algoritme, perancangan sistem, dan perancangan pengujian.

5.1.1 Perancangan Algoritme SPECK

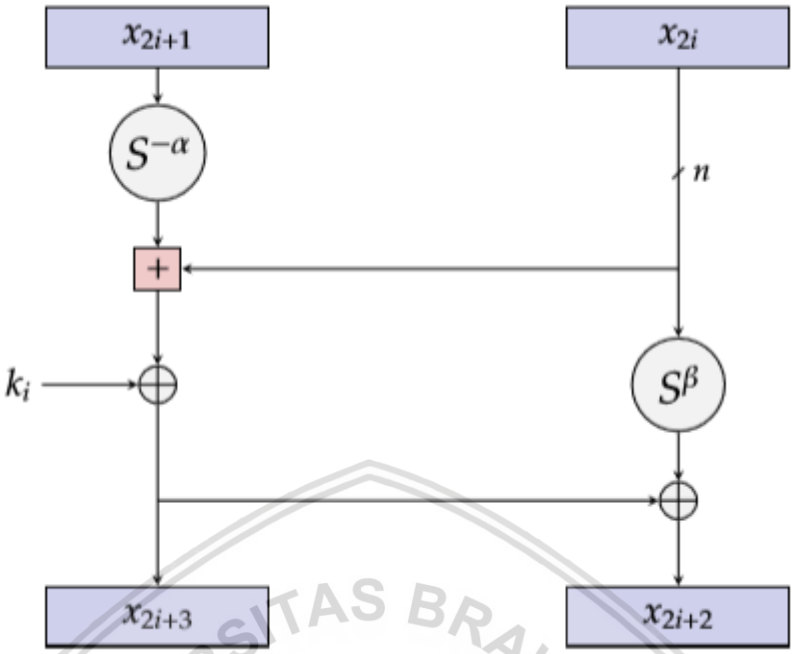
Pada algoritme SPECK terdapat dua bagian utama yaitu *block size* dan *key size*, pada *block size* jumlah bit yang masuk akan dibagi menjadi dua bagian yang sama yang disebut sebagai *word size*, dan pada bagian *key* pembagian dilakukan berdasar dari *key word*, untuk mendapatkan ukuran *key size* didapatkan dari *word size* dikalikan dengan *key word,* pada algoritme SPECK terdapat pergeseran bit sebanyak α dan β bit bergantung pada arah pergeseran bit, pergeseran bit yang dilakukan oleh algoritme SPECK yaitu pergeseran secara *bitwise* atau pergeseran setiap bit. Semua proses tersebut dilakukan sejumlah *round* sesuai versi algoritme SPECK yang sudah didefinisikan oleh *Ray Beaulieu* pada jurnalnya (Beaulieu *et al.*, 2013).



Gambar 5. 1 Proses *Key Schedule*

Sumber: (Beaulieu *et al.*, 2013)

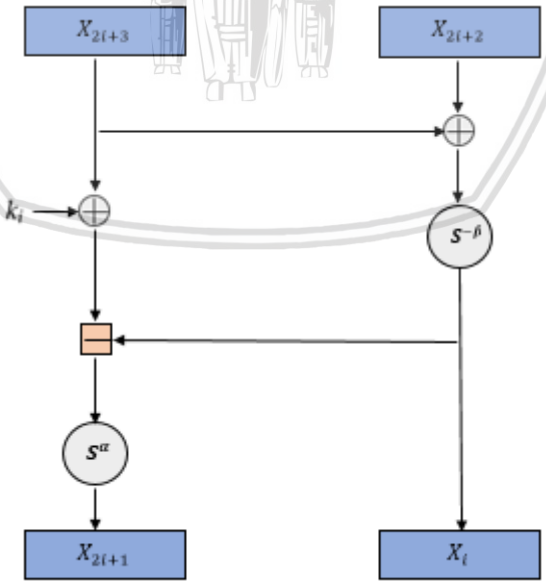
Gambar 5.1 merupakan proses pembentukan *key* atau *key_schedule*, seperti yang dijelaskan pada jurnal penelitian yang ditulis oleh *Ray Beaulieu (Beaulieu et al.*, 2013).



Gambar 5. 2 Proses *Speck Round*

Sumber: (Beaulieu *et al.*, 2013)

Gambar 5.2 menjelaskan tentang proses *speck round* untuk menghasilkan hasil enkripsi (Beaulieu, R, *et. al*., 2013).



Gambar 5. 3 Dekrip *Speck Round*

Sumber: (Beaulieu *et al.*, 2013)

Pada Gambar 5.3 merupakan *speck round* untuk melakukan proses dekripsi, berfungsi untuk mengembalikan *ciphertext* menjadi *plaintext* semula.

```
SPECK 128/128
Key:        0f0e0d0c0b0a0908 0706050403020100
Plaintext:  6c61766975716520 7469206564616d20
Ciphertext: a65d985179783265 7860fedf5c570d18
```

Gambar 5. 4 *Test Vector*

Sumber: (Beaulieu *et al.*, 2013)

Gambar 5.4 merupakan *test vector* dari algoritme SPECK dengan Panjang *key* 128-bit dan panjang *block* 128-bit yang sudah di definisikan pada jurnal yang ditulis oleh *Ray Beaulieu* (Beaulieu *et al.*, 2013).

Proses pembentukan *key schedule* dilakukan sejumlah *round*, pada penelitian ini menggunakan algoritme SPECK dengan *key* 128bit dan *block* 128bit, maka pembentukan *key schedule* dilakukan sejumlah 32 *round* yang sudah di definisikan pada Tabel 2.2, *key* 0f0e0d0c0b0a0908 dan 0706050403020100 masing-masing dikonversi dari heksadesimal menjadi integer 1084818905618843912 dan 506097522914230528.

Pembentukan *key schedule round* pertama didapatkan dari masukan *key* ke 2 yaitu 506097522914230528, *round* kedua kunci diproses dengan menggunakan proses *round* function

Pembentukan *key round* 2:

Kunci1: 0f0e0d0c0b0a0908

Kunci2: 0706050403020100

Int Kunci1: 1084818905618843912

Int Kunci2: 506097522914230528

Masukan *key*1: 1084818905618843912

Masukan *key*2: 506097522914230528

Biner *key*1:

1111000011100000110100001100000010110000101000001001000010000

Biner *key*1 dimasukkan ke *list* 64bit:

['0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0']

Geser *key* sebanyak 8 kali ke kanan:

Geser 1:

['0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0']

Geser 2:

['0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1', '0']

Geser 3:

['0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1']

Geser 4:

['1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0']

Geser 5:

['0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0']

Geser 6:

['0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0']

Geser 7:

['0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0']

Geser 8:

['0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '1']

Hasil biner setelah dilakukan pergeseran:

0000100000001111000011100000110100001100000010110000101000001001

Int X biner setelah pergeseran: 580698326153497097

X di *additional modulo* dengan *key*2/Y (506097522914230528)

X = 1086795849067727625

Cek besar nilai X jika melebihi 2^n maka nilai dikurangi dengan 2^n+1

-

X XOR dengan k = 0

X = 1086795849067727625

Key y: 506097522914230528

Biner y:

1110000011000000101000001000000001100000010000000010000000 0

Biner *key*2 dimasukkan ke *list* 64bit:

['0', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0']

Biner Y digeser 3 kali ke kiri:

Geser1:

['0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0']

Geser2:

['0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0']

Geser3:

['0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0']

Hasil biner Y setelah dilakukan pergeseran:

0011100000110000001010000010000000011000000100000000100000000000

Int Y = 4048780183313844224

Y XOR X (1086795849067727625)

Y = 3973647328251544329

X = 1086795849067727625

Key schedule 2 = 3973647328251544329

Tabel 5. 1 Manualisasi *key schedule*

Round	Masukan X	Masukan Y	*Key Schedule*
3	1086795849067727625	3973647328251544329	17950655201463765084
4	4626410970878316565	17950655201463765084	14317489089963390601
5	1035192520405693550	14317489089963390601	94520569244079856
6	3801124081208746686	94520569244079856	13113041638675238870
7	13800311577392609366	13113041638675238870	11776184413102243070
8	917158119326679627	11776184413102243070	17659011194153809395
9	17184086614850458118	17659011194153809395	6013620206426604893
10	18158482096720636610	6013620206426604893	10220345547992904357
11	1616980946765387855	10220345547992904357	12845347822737599512
12	15919211808812513588	12845347822737599512	8448324625705747870
13	16654527133838026075	8448324625705747870	8707226797264135739
14	15070622429773744840	8707226797264135739	9733733602373445842
15	4730871150006475017	9733733602373445842	12177605352264808178
16	104007319144144510054	12177605352264808178	5220725559095716473
17	1121363729463533036	5220725559095716473	8583509032072235795
18	3783954005405374681	8583509032072235795	16764294123368103200
19	5788043934926661051	16764294123368103200	16628110658475527315
20	11814929681371882900	16628110658475527315	5580441256713131253
21	8892042571447169130	5580441256713131253	15879888747126354140
22	13253280766028207990	15879888747126354140	12690311127066449729
23	5987711397884596647	12690311127066449729	15535297226658592074
24	6300574755338850119	15535297226658592074	11685805855277634329
25	2229253949779966287	11685805855277634329	16152677378382430280
26	17387063807515269253	16152677378382430280	7446018314323557823
27	7357511529715400696	7446018314323557823	7284357927190751580
28	6898297841433085095	7284357927190751580	2652372895522723097
29	898178533758263290	2652372895522723097	4081227539235714858
30	2223535841192648675	4081227539235714858	16057933008819905284
31	2000242999015463509	16057933008819905284	14212543182292431461
32	3743897877549132355	14212543182292431461	2421186661733812543

Tabel 5.1 merupakan nilai *key schedule* setiap *round*, yang nilainya didapatkan melalui proses sama seperti *round* kedua.

Setelah proses pembentukan *key_schedule* selesai, maka dilakukan proses enkripsi dari *plaintext* menjadi *ciphertext*, pada proses enkripsi masukan *key* menggunakan nilai *key* setiap *round* yang sudah dijelaskan pada Tabel 5.1, masukan *plaintext* didapatkan dari *test vector* dalam bentuk heksadesimal yaitu 6c61766975716520 dan 7469206564616d20 yang masing-masing diubah menjadi integer 7809653424151160096 dan 8388271400802151712

Proses enkripsi *round* 1 =

Masukan PT1 = 6c61766975716520

Masukan PT2 = 7469206564616d20

Int PT1 = 7809653424151160096

Int PT2 = 8388271400802151712

ROUND **1**

Pt1 = 7809653424151160096

Pt2 = 8388271400802151712

Masuk proses speck *block*

X = pt1

X = 7809653424151160096

Bin X= 11011000110000101110110011010010111010101110001011001010010 0000

List Biner =

[‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’, ‘0’]

Mengambil nilai tiap karakter pada Bin X dimulai dari belakang dimasukkan ke dalam *list* biner

['0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0']

Geser Kanan 8x

Geser 1 :

['0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0']

Geser 2 :

['0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0', '1', '0', '0', '0']

Geser 3 :

['0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0', '1', '0', '0']

Geser 4 :

['0', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0', '1', '0']

Geser 5 :

['0', '0', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0', '1']

Geser 6 :

['1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0', '0']

Geser 7 :

['0', '1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1', '0']

Geser 8 :

['0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '1', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '1', '0', '1']

Hasil bit x setelah digeser kanan = 00100000011011000110000101111011001101001011101010111000101100101

Int bit x setelah digeser kanan:

2336349467901784421

X XOR Y(8388271400802151712)

X = 10724620868703936133

Pengecekan jika Y melebihi nilai 2^64+1 maka Y dikurangi dengan nilai 2^64 agar tidak melebihi batas

-

X = 10724620868703936133

X XOR dengan *list key*[0] = 506097522914230528

X = 10652003640443985797

Y geser kiri sebanyak 3 x

Masukan Y = 8388271400802151712

Bin Y =

1110100011010010010000001100101011001000110000101101101001000000

Y dimasukkan *list* biner :

['0', '1', '1', '1', '0', '1', '0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '0', '1', '1', '0', '0', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0']

Y geser kiri 3x :

Geser 1 :

['1', '1', '1', '0', '1', '0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '0', '1', '1', '0', '0', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0']

Geser 2 :

['1', '1', '0', '1', '0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '0', '1', '1', '0', '0', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1']

Geser 3 :

['1', '0', '1', '0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0', '1', '0', '1', '1', '0', '0', '1', '0', '0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '1']

Hasil bit Y setelah digeser kiri 3x =

10100011010010010000001100101011001000110000101101101001000
00011

Int Y = 11765938985288558851

Y XOR X(10652003640443985797)

Y = 3502261146266613382

X = 10652003640443985797

Tabel 5. 2 Manualisasi Enkripsi

Round	*Key*	X	Y
2	3973647328251544329	10652003640443985797	3502261146266613382
3	17950655201463765084	10051438997473650617	11927535118356120913
4	14317489089963390601	11070810613910766769	13086198756631555644
5	94520569244079856	7384472030637367856	14602580204519412693
6	13113041638675238870	5690199013288249761	2000143021029299983
7	11776184413102243070	2197068823826700006	13867416285273009822
8	17659011194153809395	6016837892821631175	5771282879215514673
9	6013620206426604893	4908814720496875308	14168928477211072166
10	10220345547992904357	9021751147166446072	6350807575834690766
11	12845347822737599512	16346969626781676775	2578544979115758229
12	8448324625705747870	9121420255362448151	6977039667070824382
13	8707226797264135739	37236903759067138	441122036121552369
14	9733733602373445842	10307219272881207035	10307219272881207035
15	12177605352264808178	1404411197085141147	17069271371720262406
16	5220725559095716473	14953647743013859383	12148416450501388288
17	8583509032072235795	10915003181836901027	15260204053528369830
18	16764294123368103200	11526998647646371168	126441440922855510
19	16628110658475527315	9557702157148212096	9991915174888376624
20	5580441256713131253	5070693027387425294	1373025304360913802
21	15879888747126354140	18245462372344198416	7303535335452223808
22	12690311127066449729	14286993771055114612	17050466197131240311
23	15535297226658592074	13185911616303050154	15133961284916196885
24	11685805855277634329	16037826097764365787	5666866483476972917

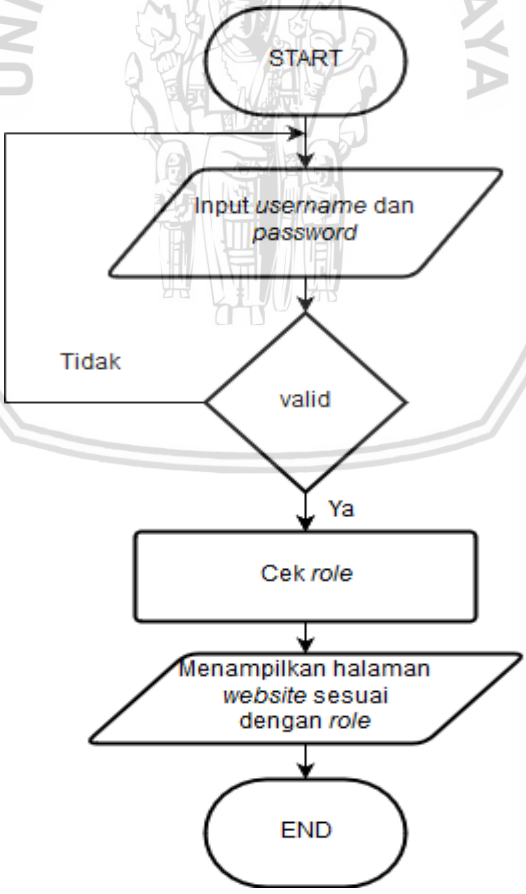
Tabel 5. 2 Manualisasi Enkripsi (lanjutan)

25	16152677378382430280	14603630590396989730	13802561923659999880
26	7446018314323557823	9584404658280936246	8747893489135413107
27	7284357927190751580	14626840139167416314	129179017315910753
28	2652372895522723097	15590040804637774137	15424141043077198385
29	4081227539235714858	3983019701258833640	9739864489950961510
30	16057933008819905284	12806835822301555376	9866432828159742340
31	14212543182292431461	18204481935023238091	13530973622405294063
32	2421186661733812543	11987905258827821669	8674213117595946264

Pada *output* akhir pada Tabel 5. 2 tepatnya pada *round* 32 dengan X = 11987905258827821669 dan Y = 8674213117595946264 yang masing-masing jika ubah menjadi heksadesimal yaitu A65D985179783265 dan 7860FEDF5C570D18, *output* heksadesimal *ciphertext* sesuai dengan *test vector*.

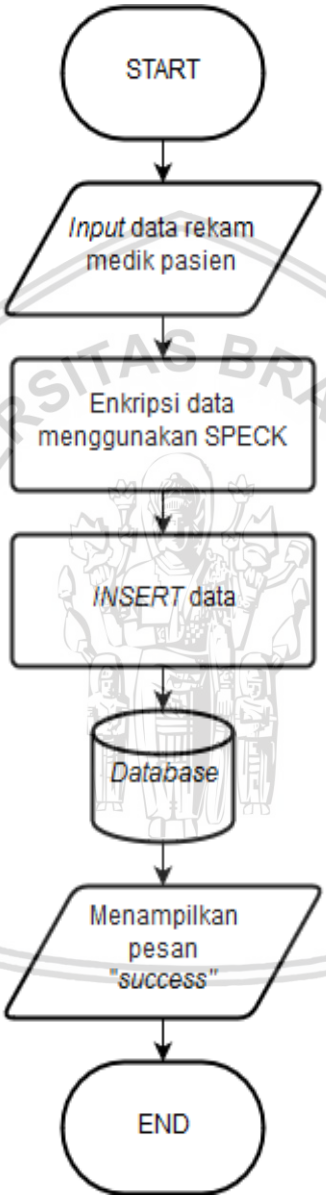
5.1.2 Perancangan Sistem

Pada perancangan sistem akan dijelaskan mengenai alur sistem secara menyeluruh dan terperinci. Berikut merupakan alur sistem:



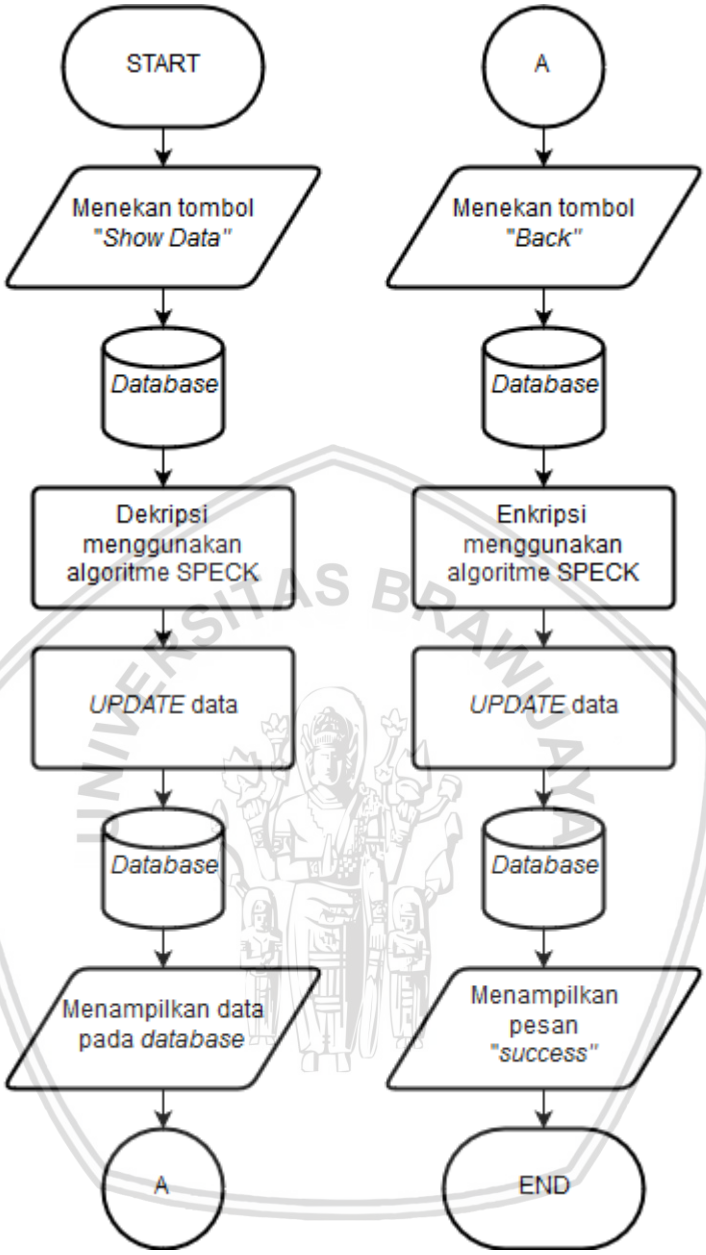
Gambar 5. 5 Alur perancangan *login*

Pada Gambar 5.5 merupakan alur dalam melakukan proses *login*, setelah pengguna melakukan *input username* dan *password* kemudian dilakukan pengecekan *username* dan *password*, pada bagian *password* dilakukan pengecekan *digest* SHA-3 224 input pengguna dengan *digest* pada *database*, kemudian dilakukan pengecekan *role* dan menampilkan halaman *website* sesuai dengan *role* pengguna.



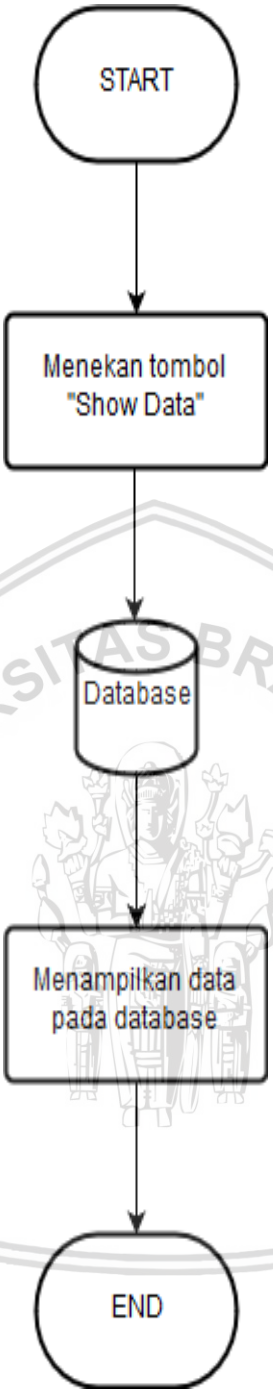
Gambar 5. 6 Alur perancangan *insert* data

Pada Gambar 5.6 menjelaskan alur perancangan proses *insert* data, pengguna melakukan *input* data rekam medik pasien, kemudian hasil input di enkripsi dengan menggunakan algoritme SPECK, setelah proses enkripsi selesai data di *insert* ke dalam *database*, setelah berhasil di *insert* pada *database* akan menampilkan pesan "*success*".



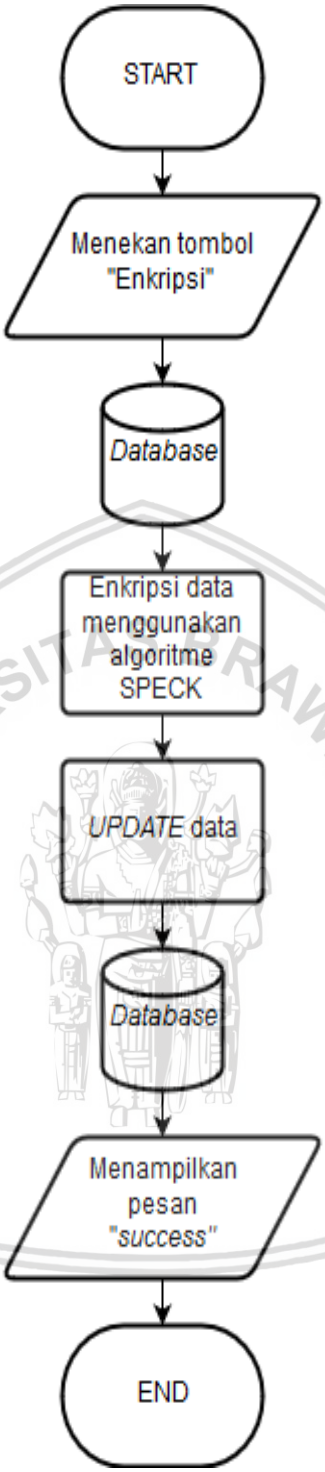
Gambar 5. 7 Alur perancangan *show* data *role* pengguna

Pada Gambar 5.7 menjelaskan tentang perancangan sistem pada saat proses s*how* data pada *role* pengguna, pengguna melakukan aksi menekan tombol "*show data"*, data pada *database* akan di dekripsi dengan menggunakan algoritme SPECK, kemudian dilakukan *update* data pada *database* dan menampilkan data yang sudah di dekripsi, lalu ketika pengguna melakukan aksi dengan menekan tombol *"back"*, data pada *database* akan di enkripsi dengan menggunakan algoritme SPECK, kemudian melakukan proses *update* data dan menampilkan pesan "*success*".



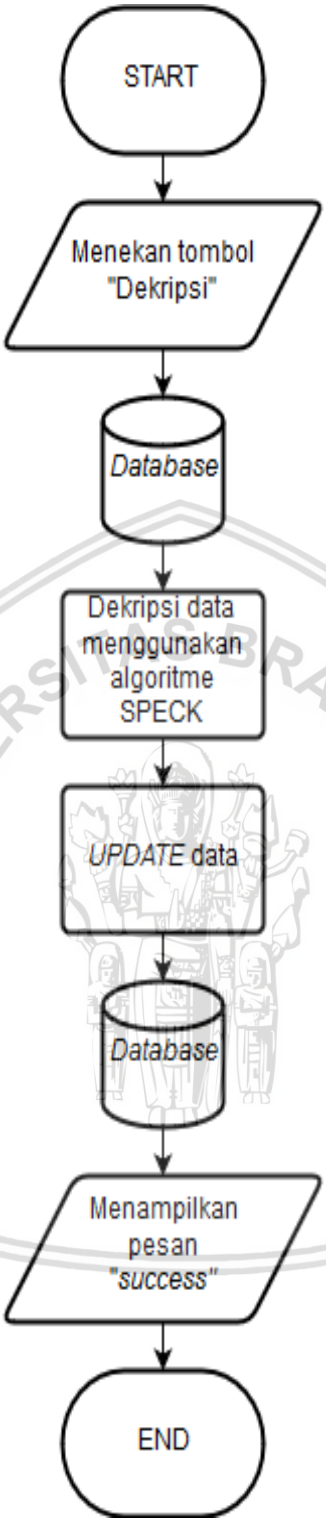
Gambar 5. 8 Alur perancangan *show* data admin

Pada Gambar 5.8 menjelaskan tentang perancangan sistem pada saat proses *show* data *role* admin, admin melakukan aksi dengan menekan tombol "*Show* Data", kemudian data diambil dari *database* dan ditampilkan pada halaman *website*.



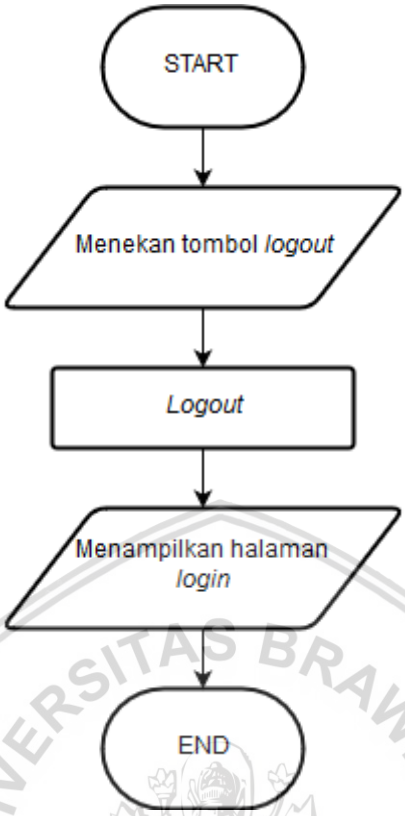
Gambar 5. 9 Alur perancangan enkripsi admin

Pada Gambar 5.9 menjelaskan tentang perancangan sistem pada saat proses enkripsi *role* admin, admin melakukan aksi dengan menekan tombol “enkripsi”, data diambil dari *database* dan dilakukan proses enkripsi dengan menggunakan algoritme SPECK, kemudian dilakukan *update* data pada *database* dan menampilkan pesan ”*success”*.



Gambar 5. 10 Alur perancangan dekripsi admin

Pada Gambar 5.10 menjelaskan tentang perancangan sistem pada saat proses dekripsi *role* admin, admin melakukan aksi dengan menekan tombol “Dekripsi”, data diambil dari *database* dan dilakukan proses dekripsi, kemudian dilakukan *update* data pada *database* dan menampilkan pesan “*success”*.



Gambar 5. 11 Alur perancangan *logout*

Pada Gambar 5.11 menjelaskan tentang perancangan sistem pada saat proses *logout*, pengguna melakukan aksi dengan menekan tombol "*logout*", kemudian sistem akan melakukan aksi *logout* dan menampilkan halaman *login*.

5.1.3 Perancangan Pengujian

Pada perancangan pengujian akan dijelaskan mengenai beberapa pengujian yang akan dilakukan terhadap sistem dengan menggunakan lima skenario pengujian, yakni pengujian validasi *test vector,* pengujian fungsionalitas, p pengujian validasi enkripsi dan dekripsi, pengujian waktu enkripsi dan dekripsi, dan pengujian keamanan dengan mengakses database melalui *command line interface* (CLI) . Berikut penjelasan lebih lanjut terhadap beberapa pengujian yang telah disebutkan:

1. Pengujian validasi *test vector*

   Pengujian ini dilakukan untuk memastikan benar atau tidaknya hasil *plaintext* dan *ciphertext* pada algoritme SPECK dengan *test vector* pada jurnal, dan pada algoritme SHA-3 dilakukan pengecekan *digest* yang dihasilkan dengan *test vector* pada jurnal.

2. Pengujian Fungsionalitas

   Pengujian fungsionalitas bertujuan untuk pengujian apakah semua fitur pada sistem dapat berjalan sesuai dengan fungsinya.

3. Pengujian Non-fungsional

   Pengujian non-fungsional bertujuan untuk pengujian *behavior* di luar fungsi sistem.

4. Pengujian validasi enkripsi dan dekripsi

   Pengujian validasi bertujuan untuk apakah data asli yang sudah dienkripsi ketika di dekripsi akan sama seperti data asli sebelum dienkripsi.

5. Pengujian waktu enkripsi dan dekripsi

   Pengujian waktu pada proses enkripsi dan dekripsi bertujuan untuk mendapatkan berapa waktu yang dibutuhkan saat melakukan proses enkripsi dan berapa waktu yang dibutuhkan saat melakukan proses dekripsi.

6. Pengujian Keamanan

   Pengujian keamanan bertujuan untuk mengetahui data pada *database*, apabila data belum di enkripsi maka data tersebut dapat dibaca, dan apabila data sudah di enkripsi maka data tidak bisa dibaca atau berupa *hexadecimal*.

5.2 Implementasi

Pada subbab implementasi akan dijelaskan bagaimana peneliti mengimplementasikan algoritme dan sistem dengan subbab perancangan sebagai acuan.

5.2.1 Algoritme SPECK

5.2.1.1 Pergeseran bit ke kiri

Algoritme1 : *Source code* pergeseran bit ke kiri

```
1   def geserkiri(value, shift):
2           tmp_bin = bin(value)[2:]
3           biner = ['0'] * 64
4           list = 63
5           for i in reversed(tmp_bin):
6               biner[list] = i
7               list -= 1
8           for i in range(shift):
9               biner.append(biner.pop(0))
10          result = ''.join(biner)
11          return int(result, 2)
```

Algoritme 1 merupakan proses pergeseran *circular* bit sejumlah *shift* ke kiri, masukan yang masuk ke dalam *method* akan diubah menjadi biner kemudian setiap nilai biner yang ada akan di masukkan ke dalam *array* sejumlah 64 *list* dimulai dari nilai 0 sampai 63, proses pembacaan bit dimulai dari biner yang paling belakang, kemudian setiap nilai biner dari masukan akan dimasukkan ke dalam *array*, setelah semua nilai sudah berada dalam *array*, maka pergeseran dilakukan sejumlah *shift* ke kanan dengan pop nilai *list* terakhir *array* kemudian di *append* ke depan.

5.2.1.2 Pergeseran bit ke kanan

Algoritme 2 : *Source code* pergeseran bit ke kanan

```
1  def geserkanan(value, shift):
2          tmp_bin = bin(value)[2:]
3          biner = ['0'] * 64
4          list = 63
5          for i in reversed(tmp_bin):
6              biner[list] = i
7              list -= 1
8          for i in range(shift):
9              biner.insert(0, biner.pop())
           result = ''.join(biner)
           return int(result, 2)
```

Algoritme 2 merupakan proses pergeseran *circular* bit sejumlah *shift* ke kiri, masukan akan diubah menjadi biner kemudian setiap nilai biner akan di simpan pada *array* sejumlah 64 dengan *list* dimulai dari 0 sampai 63, pembacaan nilai biner masukan dimulai dari belakang dan akan dimasukkan ke dalam *array*, setelah semua nilai sudah di dalam *array*, maka nilai *array* digeser sejumlah *shift* ke kanan dengan pop nilai *list* terakhir *array* kemudian di insert ke *list* 0 atau paling depan.

Algoritme 3 : *Source code key schedule*

```
1  def speck_key(key, key_schedule):
2          a = key[0]
3          b = key[1]
4          key_schedule[0] = b
5          for i in range(ROUNDS-1):
6              tmp_res = speck_block(a, b, i)
7              a = tmp_res[0]
8              b = tmp_res[1]
9              key_schedule[i+1] = b
10         return key_schedule
```

Algoritme 3 merupakan proses pada *key-schedule* pada algoritme SPECK, b merupakan variabel yang akan dikeluarkan sebagai nilai *key* yang nantinya akan di proses pada *round function*, nilai *key* ke 2 diset sebagai *key* pertama dan nilai *key* pertama di set sebagai nilai *key* kedua, kemudian diproses ke dalam *round function* yang nantinya keluaran nilai kedua setelah proses *round function* akan dimasukkan ke dalam variabel *key_schedule*, proses tersebut dilakukan sejumlah 32 *round* pada SPECK 128-bit dengan *key* 128-bit sesuai dengan jurnal yang sudah didefinisikan oleh *Ray Beaulieu* (Beaulieu, R, *et. al*., 2013). hasil.

5.2.1.3 *Round Function*

Algoritme 4: *Source code round function*

```
1   def speck_block(x, y, k):
2           x = geserkanan(x, 8)
3           x += y
4           while x > 18446744073709551615:
5               x -= (18446744073709551615 + 1)
6           x ^= k
7           y = geserkiri(y, 3)
8           y ^= x
9           result = [x, y]
10          return result
```

Algoritme 4 merupakan proses pada *round function* pada algoritme SPECK dengan masukan parameter x, y, dan k. nilai x merupakan *plaintext*1, nilai y *plaintext* 2, dan nilai k merupakan nilai *key* yang didapat dari nilai *key_schedule*, nilai x akan digeser ke kanan sejumlah 8 kali kemudian hasilnya akan di *additional modulo* dengan nilai y, hasilnya akan disimpan pada variabel x. kemudian dilakukan perulangan untuk pengecekan besar nilai x, jika nilai x melebihi 2 pangkat n maka nilai x akan dikurangi dengan nilai 2 pangkat n – 1, kemudian nilai disimpan ke variabel x, nilai x akan di XOR dengan nilai *list* pada variabel *key_schedule*, proses selanjutnya nilai y digeser ke kiri sejumlah 3x, setelah proses pergeseran nilai y akan di XOR kan dengan nilai x, hasil akan disimpan ke dalam variabel *result* yang berisi nilai x dan y.

5.2.1.4 Dekrip *Round Function*

Algoritme 5 : *Source code* dekrip *round function*

```
1   def dekripsi_speck_block(x, y, k):
2           y ^= x
3           y = geserkanan(y, 3)
4           x ^= k
5           x -= y
6           while x < 0:
7               x += (18446744073709551615 + 1)
8           x = geserkiri(x, 8)
9           result = [x, y]
10          return result
```

Algoritme 5 merupakan proses *round function* untuk melakukan dekripsi, pada *method* ini mempunyai parameter x, y, dan k. nilai x merupakan *ciphertext*1, nilai y merupakan *ciphertext*2, dan k merupakan nilai *key* yang didapatkan dari *list* pada *key_schedule*. Nilai y akan di XOR kan dengan nilai x kemudian hasilnya di geser ke kanan sejumlah 3x hasil akan disimpan ke dalam variabel y, proses selanjutnya nilai x di XOR kan dengan nilai *list* pada *key_schedule* sesuai *round* yang berjalan, hasilnya akan di kurangi dengan nilai y, kemudian dilakukan perulangan untuk pengecekan nilai x, jika nilai x kurang dari 0 maka akan ditambah dengan nilai 2 pangkat n +1, kemudian hasilnya dilakukan pergeseran ke kiri sejumlah 8x, hasil dari *method* ini akan disimpan pada variabel *result* dengan *output* x dan y.

5.2.1.5 Enkripsi

Algoritme 6 : *Source code* Enkripsi

```
1  def proses_enkripsi(plaintext, key_schedule, ciphertext):
2          ciphertext[0] = plaintext[0]
3          ciphertext[1] = plaintext[1]
4          for i in range(ROUNDS):
5              tmp_res = speck_block(ciphertext[0], ciphertext[1],
6  key_schedule[i])
7              ciphertext[0] = tmp_res[0]
8              ciphertext[1] = tmp_res[1]
9          return ciphertext
```

Algoritme 6 merupakan *method* untuk proses enkripsi dengan parameter masukan *plaintext*, *keyschedule*, dan *ciphertext*, pertama nilai dari *plaintext* 1 disimpan pada *ciphertext*[0] dan *plaintext*2 pada *ciphertext*[1] kemudian dilakukan perulangan sejumlah *round*, proses enkripsi dilakukan dengan memanggil *method round* function atau speck_block hasilnya x akan disimpan ke dalam *ciphertext*[0] dan y ke dalam *ciphertext*[1], hasil dari *method* ini akan dikeluarkan pada variabel *ciphertext*.

5.2.1.6 Dekripsi

Algoritme 7 : *Source code* Dekripsi

```
1  def proses_dekripsi(ciphertext, key_schedule, decrypted):
2          decrypted[0] = ciphertext[0]
3          decrypted[1] = ciphertext[1]
4          for i in reversed(range(ROUNDS)):
5              tmp_res = dekripsi_speck_block(decrypted[0],
6  decrypted[1], key_schedule[i])
7              decrypted[0] = tmp_res[0]
8              decrypted[1] = tmp_res[1]
9          return decrypted
```

Algoritme 7 merupakan *method* untuk proses dekripsi dengan parameter masukan *ciphertext*, *keyschedule*, dan decrypted, pertama nilai dari *plaintext* 1 disimpan pada *decrypted*[0] dan *plaintext*2 pada *decrypted*[1] kemudian dilakukan perulangan sejumlah *reverse round* atau dimulai dari *round* paling terakhir, proses dekripsi dilakukan dengan memanggil *method* dekripsi *round function* atau dekripsi_speck_*block* hasilnya x akan disimpan ke dalam *decrypted*[0] dan y ke dalam *decrypted*[1], hasil dari *method* ini akan dikeluarkan pada variabel *decrypted*.

5.2.1.7 Main

Algoritme 8 : *Source code* Main

```
1  def runprogram(pl1,pl2,k1,k2,tnt):
2          testvector = enkrip(7809653424151160096,
3  8388271400802151712, 1084818905618843912, 506097522914230528)
```

Algoritme 8 merupakan *method* untuk main untuk melakukan *output* proses enkripsi maupun dekripsi dengan set nilai *plaintext*1, *plaintext*2, *key*1, dan *key*2.

5.2.2 Implementasi pada *Database* Rekam Medik

Algoritme 9 : *Source code* pemanggilan fungsi SHA-3 pada proses *autentifikasi*

```
1   <?php
2   session_start();
3   include "config.php";
4
5   $username = $_POST['username'];
6   $password = hash('sha3-224',$_POST['password']);
7
8   $cek = mysqli_query($con, "SELECT * FROM user WHERE username =
9   '$username' AND password = '$password'");
10  $ada = mysqli_num_rows($cek);
11  $data = mysqli_fetch_assoc($cek);
12
13  if ($ada > 0) {
14     $_SESSION['username'] = $username;
15     $_SESSION['role'] = $data["role"];
16     if($data["role"] == "1"){
17        header("location:../pilih.php");
18     }else if($data["role"]=="2"){
19        header("location:../pilihbiasa.php");
20     }else {
21        header("location:../list.php");
22     }
23  }else {
24     header("location:../list.php");
25  }
```

Algoritme 9 menjelaskan tentang implementasi algoritme SHA-3 224 pada *password* saat proses autentifikasi yang nantinya *password* yang dimasukan oleh *user* akan diubah menjadi *digest* SHA-3 dan dicocokkan dengan *digest* SHA-3 pada *database*. Algoritme SHA-3 224 menggunakan API seperti yang sudah dijelaskan sebelumnya, kemudian setelah pengecekan *hash password* sesuai pada *database*, kemudian dilakukan pengecekan *user role* untuk menentukan *user* biasa atau admin.

Algoritme 10 : *Source code* pemanggilan fungsi enkripsi algoritme SPECK

```
1   <?php
2   session_start();
3   include "config.php";
4
5   $result = shell_exec("python enkrips.py");
6
7   if($result){
8      header("location:../pilih.php?success=true");
9   }else{
10     echo "fail";
11  }
12  ?>
```

Algoritme 10 menjelaskan tentang mengimplementasikan algoritme SPECK enkripsi pada *role* admin dan *user* biasa, pada *role* admin diberikan menu enkripsi manual pada tampilan halaman pilih.php dan pada *user* biasa enkripsi dilakukan pada saat *user* menekan tombol "*back"* setelah *showdata* pada *database*.

Algoritme 11 : *Source code* pemanggilan fungsi dekripsi algoritme SPECK

```
1   <?php
2   session_start();
3   include "config.php";
4
5   $result = shell_exec("python dekrips.py");
6
7   if($result){
8      header("location:../pilih.php?success=true");
9   }else{
10     echo "fail";
11  }
12  ?>
```

Algoritme 11 menjelaskan tentang mengimplementasikan algoritme SPECK dekripsi pada *role* admin dan *user* biasa, pada *role* admin diberikan menu dekripsi manual pada tampilan halaman pilih.php dan pada *user* biasa enkripsi dilakukan pada saat *user* menekan *showdata* pada tampilan pilihbiasa.php.

BAB 6 PENGUJIAN

Pengujian pada penelitian ini terdiri dari pengujian *test vectors, performance* waktu, validasi enkripsi dan dekripsi, keamanan, fungsional dan pengujian non-fungsional.

6.1 Pengujian *Test Vector*

Pada pengujian *test vector* digunakan untuk memastikan bahwa algoritme SPECK yang di buat oleh peneliti memiliki *output ciphertext* yang sama dengan algoritme SPECK yang buat oleh penciptanya. Dan pengujian *test vector* dilakukan juga pada algoritme SHA-3 yang menggunakan *library* dari php dan python yang digunakan oleh peneliti dengan mencocokkan *output digest text* dari *library* dengan yang telah dibuat oleh penciptanya. Jika *ciphertext* dari algoritme SPECK dan *digest text* dari SHA-3 sama maka *code* pada penelitian ini valid.

6.1.1 Algoritme SPECK

Test vector algoritme SPECK

Tabel 6. 1 *Test vector* dari pencipta algoritme SPECK

Key	0f0e0d0c0b0a0908	0706050403020100
Plaintext	6c61766975716520	7469206564616d20
Ciphertext	a65d985179783265	7860fedf5c570d18

Sumber : (Beaulieu *et al.*, 2013)

Hasil *ciphertext* algoritme SPECK yang dibuat oleh peneliti

```
PT1: 6c61766975716520
PT2: 7469206564616d20
key 1 : 0f0e0d0c0b0a0908
key 2 : 0706050403020100
Hasil Enkripsi PT1 : a65d985179783265
Hasil Enkripsi PT2 : 7860fedf5c570d18
```

Gambar 6. 1 Hasil *test vector* algoritme SPECK dari Peneliti

Dari hasil pengujian *test vector* pada Gambar 6.1 dapat dibuktikan bahwa hasil *ciphertext* yang digunakan oleh peneliti memiliki hasil yang sama pada *ciphertext* yang diterbitkan oleh pencipta algoritme pada Tabel 6.1, sehingga program yang ditulis oleh peneliti mempunyai hasil yang bersifat valid.

Proses pembentukan *key* dan dapat menghasilkan *ciphertext* di atas pada setiap *round* nya dijelaskan pada halaman 23 - 31.

6.1.2 Algoritme SHA-3

Test vector algoritme SHA-3

a) Pesan: “abc”
SHA-3: e642824c3f8cf24a d09234ee7d3c766f c9a3a5168d0c94ad 73b46fdf
b) Pesan: “” (*string* kosong)
SHA-3: 6b4e03423667dbb7 3b6e15454f0eb1ab d4597f9a1b078e3f 5b5a6bc7
c) Pesan: "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
SHA-3: 8a24108b154ada21 c9fd5574494479ba 5c7e7ab76ef264ea d0fcce33

Hasil *digest text* algoritme *library* SHA-3 pada php yang digunakan oleh peneliti

```
test vector 'abc' :
e642824c3f8cf24ad09234ee7d3c766fc9a3a5168d0c94ad73b46fdf
test vector 'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq' :
8a24108b154ada21c9fd5574494479ba5c7e7ab76ef264ead0fcce33
test vector '' :
6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7
```

Gambar 6. 2 Hasil *test vector library* PhP algoritme SHA-3

Dari hasil *test vector library* PhP yang digunakan peneliti Gambar 6.2 mempunyai hasil yang sama dengan *test vector* SHA-3 yang mengacu pada situs *https://www.di-mgt.com.au/sha_testvectors.html*. Sehingga *library* SHA-3 pada PhP yang digunakan peneliti bersifat valid.

Hasil *digest text* algoritme *library* SHA-3 pada python yang digunakan oleh peneliti

```
test vector 'abc' :
e642824c3f8cf24ad09234ee7d3c766fc9a3a5168d0c94ad73b46fdf
test vector 'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq' :
8a24108b154ada21c9fd5574494479ba5c7e7ab76ef264ead0fcce33
test vector '' :
6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7
```

Gambar 6. 3 Hasil *test vector library* python algoritme SHA-3

Dari hasil *test vector library* python Gambar 6.3 yang digunakan peneliti mempunyai hasil yang sama dengan *test vector* SHA-3 yang mengacu pada situs *https://www.di-mgt.com.au/sha_testvectors.html*. Sehingga *library* SHA-3 pada python yang digunakan peneliti bersifat valid.

6.2 Pengujian *performance* Waktu

Pengujian *performance* waktu yang dibutuhkan oleh sistem dalam melakukan proses enkripsi dan dekripsi pada keempat *field database* yaitu nama, alamat, keterangan rekam medik, dan gejala. Pengujian *performance* waktu dilakukan sebanyak 40 kali dengan menggunakan python *timeit modul* untuk mendapatkan waktu tempuh, data uji dilakukan dengan masukan 1-char, *full*-char, data asli dari jurnal, dan data kosong. Pengujian seluruh hasil waktu kemudian diambil rata-ratanya dalam waktu *milisecond*.

6.2.1 Pengujian waktu enkripsi ketika *field database* berisi data kosong

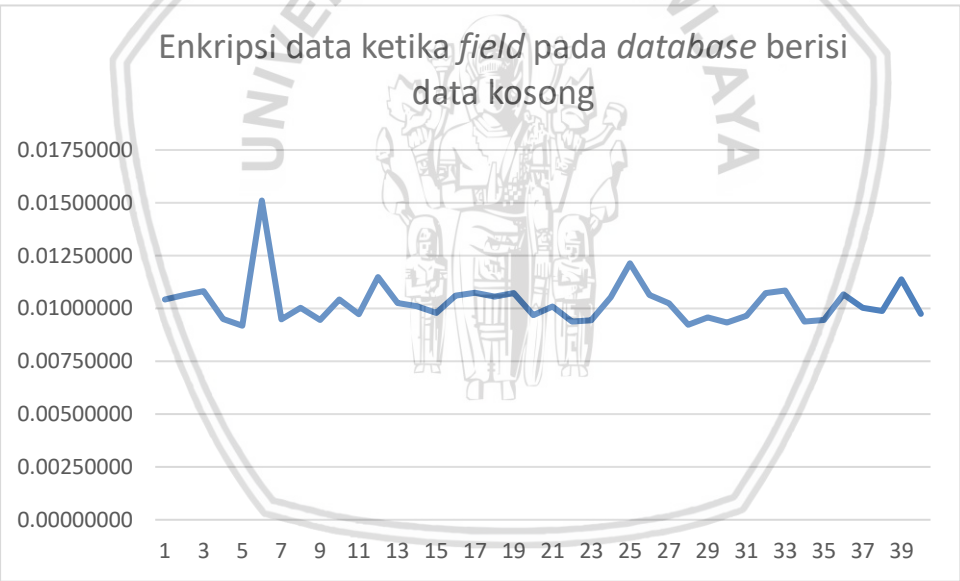
Tabel 6. 2 Pengujian waktu enkripsi ketika *field database* berisi data kosong

Percobaan	Kosong Semua Enkrip (s)
1	0.01042878
2	0.01063277
3	0.01081239
4	0.00950419
5	0.00918516
6	0.01510903
7	0.00948580
8	0.01003919
9	0.00945758
10	0.01043306
11	0.00973299
12	0.01148680
13	0.01025430
14	0.01010847
15	0.00979671
16	0.01060497
17	0.01073712
18	0.01056563
19	0.01072600
20	0.00968124
21	0.01009307
22	0.00937333
23	0.00943448
24	0.01052500
25	0.01213170
26	0.01064004
27	0.01024660
28	0.00922707
29	0.00958117
30	0.00933997

Tabel 6. 2 Pengujian waktu enkripsi ketika *field database* berisi data kosong (lanjutan)

31	0.00965301
32	0.01072771
33	0.01084916
34	0.00938231
35	0.00946185
36	0.01067169
37	0.01002807
38	0.00988737
39	0.01138544
40	0.00974453
Rata-rata	0.01027914

Pada Tabel 6.2 merupakan hasil waktu rata-rata yang ditempuh oleh sistem ketika melakukan proses enkripsi keempat *field* yang kosong yaitu 0.01027914 *second* atau 10.279 *milisecond*.



Gambar 6. 4 Grafik waktu pemrosesan enkripsi data kosong

Pada Gambar 6.4 menjelaskan tentang waktu pemrosesan data saat semua *field* kosong pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.1.

6.2.2 Pengujian waktu enkripsi ketika field *database* berisi data 1-char

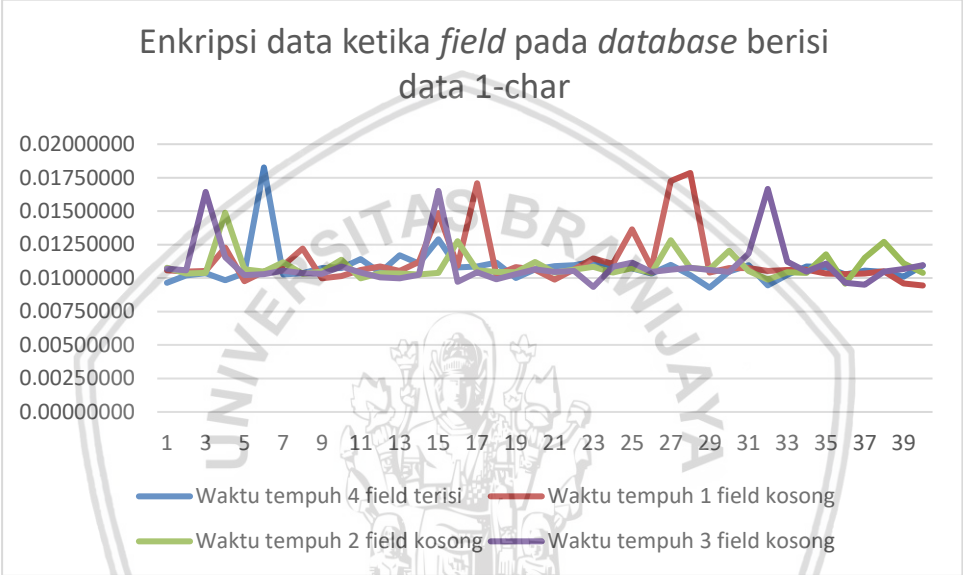
Tabel 6. 3 Pengujian waktu enkripsi ketika *field database* berisi data 1-char

Percobaan	Waktu tempuh 4 *field* terisi (s)	Waktu tempuh 1 *field* kosong (s)	Waktu tempuh 2 *field* kosong (s)	Waktu tempuh 3 *field* kosong (s)
1	0.00964190	0.01057546	0.01076492	0.01069521
2	0.01019314	0.01048993	0.01034752	0.01058017
3	0.01035736	0.01053355	0.01038815	0.01643903

Tabel 6. 3 Pengujian waktu enkripsi ketika *field database* berisi data 1-char (lanjutan)

4	0.00982835	0.01228138	0.01488023	0.01155821
5	0.01036207	0.00975480	0.01065757	0.01021752
6	0.01826426	0.01047326	0.01049507	0.01028979
7	0.01027953	0.01093812	0.01117418	0.01054681
8	0.01036634	0.01220397	0.01031588	0.01035479
9	0.01074268	0.00999001	0.01049122	0.01031673
10	0.01074952	0.01013755	0.01137732	0.01087311
11	0.01142265	0.01059813	0.00996948	0.01041894
12	0.01040740	0.01085643	0.01035950	0.01006185
13	0.01169806	0.01052415	0.01034581	0.00998060
14	0.01107283	0.01121010	0.01025900	0.01024660
15	0.01290960	0.01485500	0.01038259	0.01649762
16	0.01078330	0.01090262	0.01275950	0.00970861
17	0.01087568	0.01708009	0.01059086	0.01043690
18	0.01114211	0.01024403	0.01046043	0.00990619
19	0.01000839	0.01080939	0.01043947	0.01030262
20	0.01066014	0.01063363	0.01120283	0.01064560
21	0.01090091	0.00987796	0.01041424	0.01045144
22	0.01097147	0.01064346	0.01062464	0.01052500
23	0.01121566	0.01145429	0.01083035	0.00933741
24	0.01087867	0.01105230	0.01040868	0.01084275
25	0.01073070	0.01362934	0.01074353	0.01114809
26	0.01033512	0.01076449	0.01039799	0.01045914
27	0.01098730	0.01725628	0.01281595	0.01063363
28	0.01022179	0.01785115	0.01074225	0.01077304
29	0.00928438	0.01041296	0.01064303	0.01059471
30	0.01049207	0.01075679	0.01204018	0.01046941
31	0.01095308	0.01075636	0.01053869	0.01181481
32	0.00944389	0.01052030	0.00987582	0.01666056
33	0.01024147	0.01060797	0.01044546	0.01121566
34	0.01087097	0.01061524	0.01038045	0.01048694
35	0.01081623	0.01033726	0.01178102	0.01109122
36	0.01026755	0.01030390	0.00958202	0.00964489
37	0.01055622	0.01033897	0.01150048	0.00950932
38	0.01045187	0.01050661	0.01269577	0.01048181
39	0.01010676	0.00960212	0.01110191	0.01067725
40	0.01093726	0.00944517	0.01038259	0.01095308
Rata-rata	0.01081072	0.01129561	0.01089016	0.01094618

Pada Tabel 6.3 dilakukan proses enkripsi dengan 4 percobaan, pada percobaan 1 dilakukan dengan mengisi keempat *field* dengan masing masing 1-char hasil waktu yang didapatkan 0.01081072 *second* atau 10.811 *milisecond*, kemudian percobaan 2 pada *field* alamat, keterangan rekam medik, dan gejala diisi masing masing 1-char dan pada *field* nama tidak diisi hasil waktu yang didapatkan 0.01129561 *second* atau 11.296 *milisecond*, lalu percobaan 3 *field* keterangan rekam medik dan gejala diisi masing-masing 1-char sisa *field* tidak diisi hasil waktu yang didapatkan 0.1089016 *second* atau 10.890 *milisecond*, percobaan 4 *field* yang diisi hanya gejala dengan 1-char hasil waktu yang didapatkan 0.01094618 *second* atau 1.946 *milisecond*.



Gambar 6. 5 Grafik perbandingan waktu pemrosesan enkripsi pada data 1-char

Pada Gambar 6.5 menjelaskan tentang perbandingan waktu pemrosesan data 1-char pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.2.

6.2.3 Pengujian waktu enkripsi ketika *field database* berisi data *full*-char

Tabel 6. 4 Pengujian waktu enkripsi ketika *field database* berisi data *full*-char

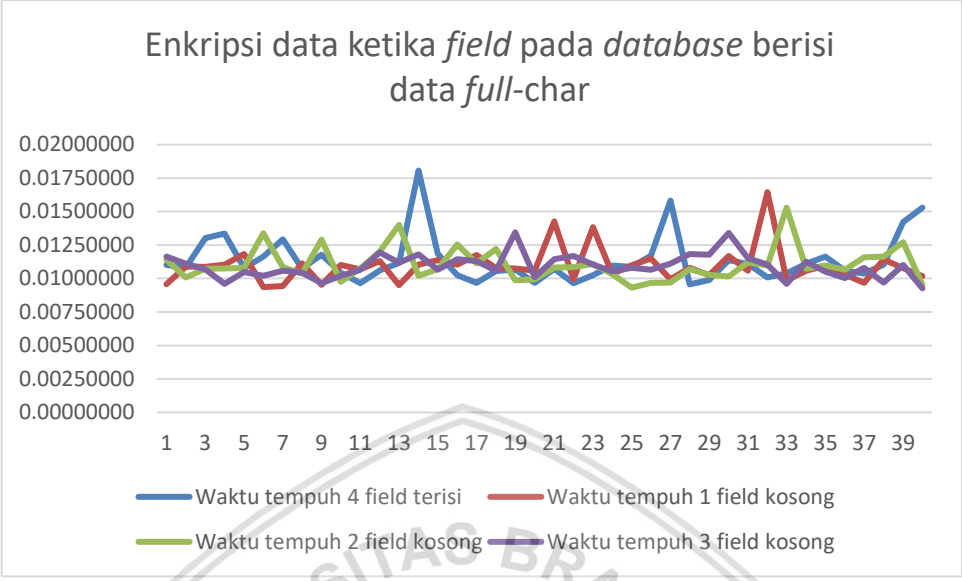
Percobaan	Waktu tempuh 4 *field* terisi (s)	Waktu tempuh 1 *field* kosong (s)	Waktu tempuh 2 *field* kosong (s)	Waktu tempuh 3 *field* kosong (s)
1	0.01102322	0.00959186	0.01139057	0.01164161
2	0.01082308	0.01088637	0.01007896	0.01111816
3	0.01302635	0.01086584	0.01072172	0.01068110
4	0.01335522	0.01104717	0.01076192	0.00959785
5	0.01077176	0.01183320	0.01078929	0.01049635
6	0.01163776	0.00935879	0.01339285	0.01019785
7	0.01292371	0.00943234	0.01085900	0.01059685
8	0.01074268	0.01112970	0.01037960	0.01044674
9	0.01178145	0.00953327	0.01291431	0.00964147

Tabel 6. 4 Pengujian waktu enkripsi ketika *field database* berisi data *full*-char (lanjutan)

10	0.01044845	0.01101851	0.00973940	0.01020811
11	0.00964703	0.01068965	0.01075593	0.01066484
12	0.01060754	0.01130846	0.01203034	0.01194396
13	0.01115408	0.00949991	0.01399456	0.01118530
14	0.01806925	0.01102750	0.01021367	0.01180027
15	0.01184731	0.01137774	0.01069264	0.01066613
16	0.01025857	0.01103648	0.01254481	0.01144360
17	0.00967696	0.01176092	0.01112500	0.01130462
18	0.01057204	0.01076534	0.01220141	0.01057718
19	0.01067254	0.01075978	0.00987882	0.01344802
20	0.00967953	0.01059556	0.00994938	0.01010333
21	0.01073070	0.01426783	0.01083548	0.01145173
22	0.00965644	0.00993313	0.01085430	0.01168566
23	0.01024703	0.01383847	0.01107796	0.01103519
24	0.01096720	0.01035009	0.01031288	0.01053869
25	0.01088124	0.01099542	0.00931902	0.01080169
26	0.01171944	0.01150048	0.00968081	0.01064988
27	0.01581466	0.00994125	0.00970519	0.01112286
28	0.00956406	0.01080340	0.01069392	0.01182849
29	0.00988780	0.01024617	0.01027055	0.01177974
30	0.01136235	0.01169506	0.01015850	0.01340910
31	0.01107582	0.01059385	0.01121566	0.01152913
32	0.01008794	0.01645101	0.01095864	0.01104417
33	0.01039542	0.00986684	0.01529463	0.00960939
34	0.01111003	0.01057333	0.01072557	0.01124474
35	0.01164888	0.01095095	0.01097575	0.01051474
36	0.01057889	0.01030733	0.01065287	0.01002422
37	0.01036463	0.00967996	0.01158516	0.01080811
38	0.01104161	0.01142863	0.01162707	0.00968936
39	0.01422934	0.01077304	0.01271160	0.01101424
40	0.01528950	0.01018459	0.00952343	0.00926556
Rata-rata	0.01138429	0.01094748	0.01106483	0.01092025

Pada Tabel 6.4 dilakukan proses enkripsi dengan 4 percobaan, pada percobaan 1 dilakukan dengan mengisi keempat *field* dengan *full*-char hasil waktu yang didapatkan 0.01138429 *second* atau 11.384 *milisecond*, kemudian percobaan 2 pada *field* alamat, keterangan rekam medik, dan gejala diisi masing-masing *full*-char dan pada *field* nama tidak diisi hasil waktu yang didapatkan 0.01094748 *second* atau 10.947 *milisecond*, lalu percobaan 3 *field* keterangan rekam medik dan gejala diisi masing-masing *full*-char sisanya tidak diisi hasil waktu yang didapatkan 0.01106483 *second* atau 11.065 *milisecond*, percobaan 4 *field* yang

diisi hanya gejala dengan *full*-char hasil waktu yang didapatkan 0.1092025 *second* atau 10.920 *milisecond*.



Gambar 6. 6 Grafik perbandingan waktu pemrosesan enkripsi data *full*-char

Pada Gambar 6.6 menjelaskan tentang perbandingan waktu pemrosesan data *full*-char pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.3.

6.2.4 Pengujian waktu enkripsi ketika *field database* berisi data dari jurnal

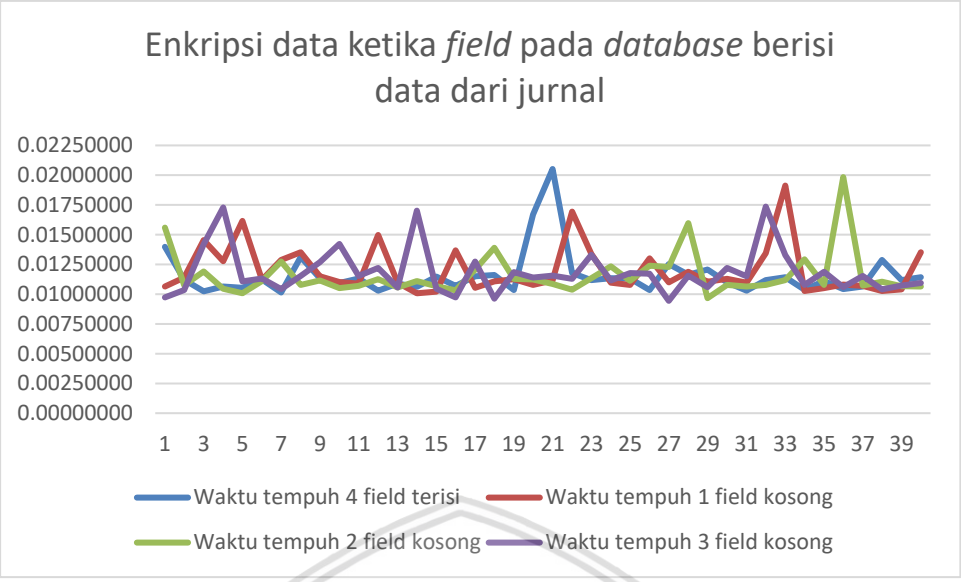
Tabel 6. 5 Pengujian waktu enkripsi ketika *field database* berisi data dari jurnal

Percobaan	Waktu tempuh 4 *field* terisi (s)	Waktu tempuh 1 *field* kosong (s)	Waktu tempuh 2 *field* kosong (s)	Waktu tempuh 3 *field* kosong (s)
1	0.01397275	0.01065586	0.01559099	0.00973470
2	0.01120369	0.01139143	0.01075807	0.01034068
3	0.01024318	0.01454452	0.01190761	0.01418316
4	0.01064090	0.01275693	0.01045658	0.01730033
5	0.01053270	0.01615935	0.01007896	0.01107026
6	0.01124517	0.01120540	0.01109335	0.01130376
7	0.01014225	0.01286983	0.01273897	0.01044161
8	0.01311573	0.01352414	0.01078502	0.01152400
9	0.01133669	0.01151246	0.01116178	0.01267781
10	0.01093213	0.01102621	0.01050533	0.01420882
11	0.01131189	0.01088167	0.01070504	0.01152058
12	0.01030177	0.01497817	0.01125372	0.01219542
13	0.01088680	0.01097276	0.01057162	0.01056862
14	0.01062636	0.01007383	0.01108865	0.01700653
15	0.01147525	0.01021666	0.01067169	0.01044760
16	0.01068537	0.01369563	0.01032999	0.00972229

Tabel 6. 5 Pengujian waktu enkripsi ketika *field database* berisi data dari jurnal (lanjutan)

17	0.01149321	0.01053954	0.01206199	0.01273768
18	0.01160868	0.01106513	0.01389877	0.00961752
19	0.01034795	0.01125501	0.01128280	0.01186142
20	0.01668878	0.01076620	0.01124303	0.01139015
21	0.02051886	0.01114767	0.01085216	0.01155223
22	0.01176948	0.01693982	0.01037661	0.01128922
23	0.01119300	0.01337617	0.01134268	0.01330561
24	0.01134054	0.01097532	0.01233655	0.01114980
25	0.01130376	0.01078929	0.01112286	0.01176520
26	0.01035223	0.01300967	0.01239599	0.01171602
27	0.01252471	0.01098259	0.01227625	0.00944047
28	0.01165016	0.01188409	0.01596519	0.01156933
29	0.01205943	0.01106042	0.00966499	0.01059899
30	0.01098088	0.01129735	0.01078159	0.01221253
31	0.01028081	0.01096377	0.01064175	0.01150433
32	0.01119129	0.01344118	0.01077775	0.01736063
33	0.01143120	0.01912599	0.01119214	0.01328679
34	0.01036634	0.01027269	0.01293098	0.01077432
35	0.01110020	0.01051730	0.01077732	0.01186784
36	0.01042279	0.01079656	0.01983803	0.01058573
37	0.01063363	0.01069906	0.01073241	0.01152229
38	0.01288352	0.01026456	0.01105700	0.01040184
39	0.01116990	0.01040355	0.01066142	0.01072129
40	0.01142564	0.01353098	0.01065330	0.01094025
Rata-rata	0.01158474	0.01203922	0.01161402	0.01183544

Pada Tabel 6.5 dilakukan proses enkripsi dengan 4 percobaan, pada percobaan 1 dilakukan dengan mengisi keempat *field* dengan data asli sesuai pada jurnal acuan hasil waktu yang didapatkan 0.01158474 *second* atau 11.585 *milisecond*, kemudian percobaan 2 pada *field* alamat, keterangan rekam medik, dan gejala diisi dengan data asli dan pada *field* nama tidak diisi hasil waktu yang didapatkan 0.01203922 *second* atau 12.039 *milisecond*, lalu percobaan 3 *field* keterangan rekam medik dan gejala diisi masing-masing data sesuai jurnal sisanya tidak diisi hasil waktu yang didapatkan 0.01161402 *second* atau 11.614 *milisecond*, percobaan 4 *field* yang diisi hanya gejala dengan data sesuai jurnal hasil waktu yang didapatkan 0.01183544 *second* atau 11.835 *milisecond*.



Gambar 6. 7 Grafik perbandingan waktu pemrosesan enkripsi data dari jurnal

Pada Gambar 6.7 menjelaskan tentang perbandingan waktu pemrosesan data dari jurnal pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.4.

6.2.5 Pengujian waktu dekripsi ketika field *database* berisi *ciphertext* data kosong

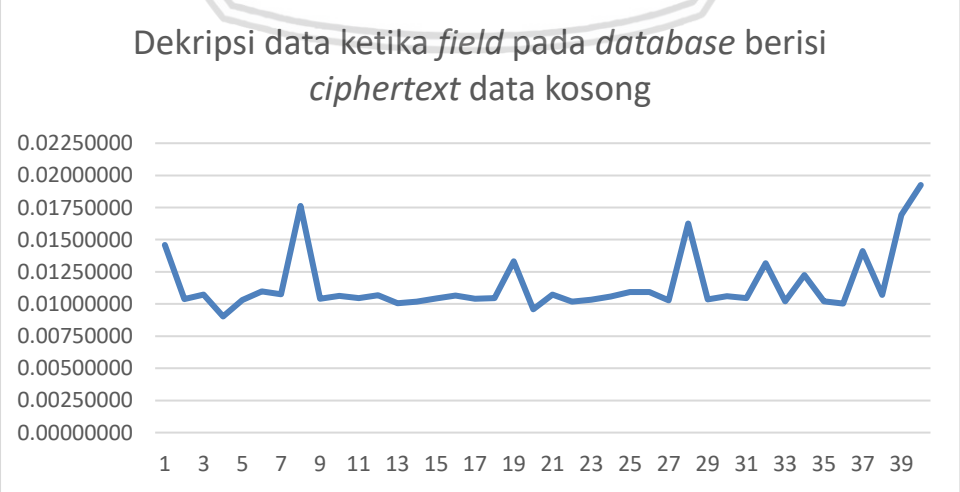
Tabel 6. 6 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data kosong

Percobaan	Kosong Semua Dekrip
1	0.01459199
2	0.01039029
3	0.01073412
4	0.00902821
5	0.01031844
6	0.01097233
7	0.01075422
8	0.01761508
9	0.01041167
10	0.01062464
11	0.01044760
12	0.01068794
13	0.01004603
14	0.01017518
15	0.01042493
16	0.01065544
17	0.01039585

Tabel 6. 6 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data kosong (lanjutan)

18	0.01045871
19	0.01332272
20	0.00958758
21	0.01073412
22	0.01017390
23	0.01032743
24	0.01058273
25	0.01093726
26	0.01091716
27	0.01028295
28	0.01625471
29	0.01036677
30	0.01059642
31	0.01045187
32	0.01316919
33	0.01020255
34	0.01223904
35	0.01020212
36	0.01003277
37	0.01411217
38	0.01069692
39	0.01693084
40	0.01925043
Rata-rata	0.01150261

Pada Tabel 6.6 hasil waktu rata-rata yang ditempuh oleh sistem ketika melakukan proses dekripsi keempat *field* yang kosong yaitu 0.01150261 *second* atau 11.503 *milisecond*.



Gambar 6. 8 Grafik waktu pemrosesan dekripsi data kosong

Pada Gambar 6.8 menjelaskan tentang waktu pemrosesan data saat semua *field* kosong pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.5.

6.2.6 Pengujian waktu dekripsi ketika field *database* berisi *ciphertext* data 1-char

Tabel 6. 7 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data 1-char

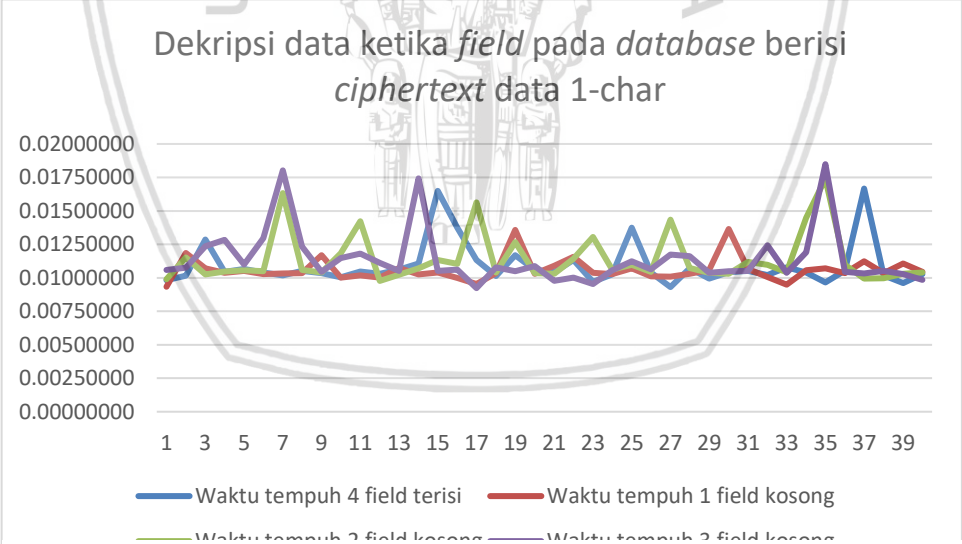
Percobaan	Waktu tempuh 4 *field* terisi (s)	Waktu tempuh 1 *field* kosong (s)	Waktu tempuh 2 *field* kosong (s)	Waktu tempuh 3 *field* kosong (s)
1	0.00982108	0.00933527	0.00989079	0.01059685
2	0.01014824	0.01186399	0.01152443	0.01075593
3	0.01285272	0.01071146	0.01024660	0.01234895
4	0.01044075	0.01038345	0.01047154	0.01283262
5	0.01065159	0.01052415	0.01057546	0.01098687
6	0.01039756	0.01026755	0.01049635	0.01295793
7	0.01018544	0.01031631	0.01633725	0.01802606
8	0.01050704	0.01035052	0.01060155	0.01233697
9	0.01035736	0.01167839	0.01040526	0.01045059
10	0.01005116	0.01002208	0.01184346	0.01148466
11	0.01047283	0.01017903	0.01421951	0.01180967
12	0.01030262	0.01000797	0.00977233	0.01111303
13	0.01062122	0.01073669	0.01022393	0.01050704
14	0.01106898	0.01024574	0.01070804	0.01742136
15	0.01650019	0.01040269	0.01134353	0.01051517
16	0.01374609	0.00998915	0.01103348	0.01062464
17	0.01133027	0.00954909	0.01563376	0.00922964
18	0.01012686	0.01034581	0.01040312	0.01079357
19	0.01167924	0.01358658	0.01267910	0.01048608
20	0.01057889	0.01030091	0.01036207	0.01087311
21	0.01043477	0.01090775	0.01029834	0.00978901
22	0.01134268	0.01156249	0.01143419	0.01002294
23	0.00969663	0.01037832	0.01304431	0.00953199
24	0.01024574	0.01028894	0.01061866	0.01057589
25	0.01373711	0.01070847	0.01097062	0.01124432
26	0.01039842	0.01012386	0.01043690	0.01064261
27	0.00931046	0.01007767	0.01433369	0.01172885
28	0.01070761	0.01030733	0.01073412	0.01160825
29	0.00994895	0.01059172	0.01038687	0.01036591
30	0.01041338	0.01364602	0.01030305	0.01050490
31	0.01052971	0.01066741	0.01119086	0.01052500
32	0.01021880	0.01006656	0.01097489	0.01243020
33	0.01079100	0.00947981	0.01043263	0.01038302
34	0.01040312	0.01056648	0.01449791	0.01191188

Tabel 6. 7 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data 1-char (lanjutan)

35	0.00964917	0.01071702	0.01740682	0.01848964
36	0.01054254	0.01034496	0.01096976	0.01043519
37	0.01666740	0.01123619	0.00994339	0.01033726
38	0.01018544	0.01034154	0.00997504	0.01052842
39	0.00961324	0.01107967	0.01030262	0.01029065
40	0.01026926	0.01045829	0.01040055	0.00984631
Rata-rata	0.01092364	0.01060868	0.01143567	0.01143357

Pada Tabel 6.7 dilakukan proses dekripsi dengan 4 percobaan, pada percobaan 1 dekripsi keempat *field* yang sudah terenkripsi dengan data 1-char hasil waktu yang didapatkan 0.01092364 *second* atau 10.924 *milisecond*, kemudian percobaan 2 pada *field* alamat, keterangan rekam medik, dan gejala telah terenkripsi dengan data 1-char pada *field* nama terenkripsi dengan data kosong hasil waktu yang didapatkan 0.01060868 *second* atau 10.609 *milisecond*, lalu percobaan 3 *field* keterangan rekam medik dan gejala yang terenkripsi dengan data 1 char sisanya terenkripsi dengan data kosong waktu yang didapatkan 0.01143567 *second* atau 11.436 *milisecond*, percobaan 4 hanya *field* gejala yang terenkripsi dengan data 1-char sisa *field* yang lain terenkripsi dengan data kosong hasil waktu yang didapatkan 0.01143357 *second* atau 11.434 *milisecond*.

Pada Gambar 6.9 menjelaskan tentang perbandingan waktu pemrosesan



Gambar 6. 9 Grafik perbandingan pemrosesan waktu dekripsi 1-char

data dari jurnal pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.6.

6.2.7 Pengujian waktu dekripsi ketika field *database* berisi *ciphertext* data *full*-char

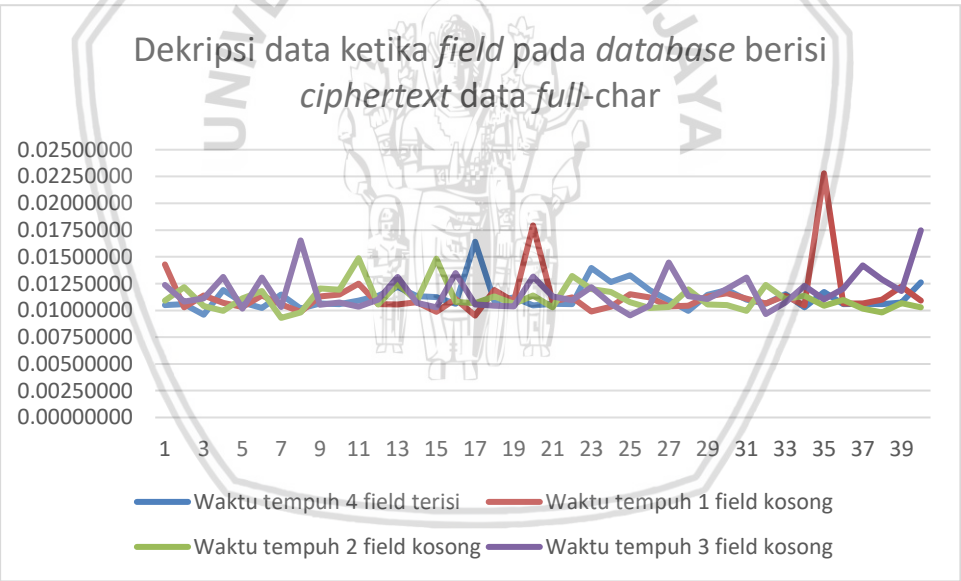
Tabel 6. 8 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data *full*-char

Percobaan	Waktu tempuh 4 *field* terisi (s)	Waktu tempuh 1 *field* kosong (s)	Waktu tempuh 2 *field* kosong (s)	Waktu tempuh 3 *field* kosong (s)
1	0.01049207	0.01428836	0.01092101	0.01238059
2	0.01059599	0.01027739	0.01217831	0.01082350
3	0.00959357	0.01137561	0.01039457	0.01112842
4	0.01190205	0.01069307	0.00994510	0.01311830
5	0.01074695	0.01030390	0.01113441	0.01017775
6	0.01023890	0.01140511	0.01181310	0.01305329
7	0.01148722	0.01055066	0.00929763	0.01031759
8	0.01016150	0.00986214	0.00981039	0.01652414
9	0.01064303	0.01128623	0.01205429	0.01053355
10	0.01058316	0.01147867	0.01193626	0.01073370
11	0.01090690	0.01250162	0.01487082	0.01035351
12	0.01136064	0.01059172	0.01056050	0.01100141
13	0.01220354	0.01055708	0.01245757	0.01312343
14	0.01133840	0.01077005	0.01089022	0.01071189
15	0.01123234	0.00985145	0.01481651	0.01028338
16	0.01061224	0.01105786	0.01078074	0.01347496
17	0.01640953	0.00950889	0.01067938	0.01055323
18	0.01089407	0.01190590	0.01128665	0.01043776
19	0.01121267	0.01082436	0.01060412	0.01036463
20	0.01046342	0.01792299	0.01137903	0.01314609
21	0.01060968	0.01081025	0.01027012	0.01131317
22	0.01059514	0.01121139	0.01321067	0.01094710
23	0.01395137	0.00988352	0.01198501	0.01215394
24	0.01263548	0.01033256	0.01171645	0.01061010
25	0.01326755	0.01151288	0.01077603	0.00953498
26	0.01189649	0.01122422	0.01021196	0.01043947
27	0.01093256	0.01038772	0.01032315	0.01445728
28	0.00996135	0.01044717	0.01196620	0.01133241
29	0.01144489	0.01130761	0.01055365	0.01106128
30	0.01191103	0.01160526	0.01048566	0.01200853
31	0.01101509	0.01108052	0.00996477	0.01306099
32	0.01060583	0.01064303	0.01236862	0.00966884
33	0.01150476	0.01136962	0.01105059	0.01072942
34	0.01028979	0.01047839	0.01136278	0.01229079
35	0.01172928	0.02279612	0.01044589	0.01103263
36	0.01058530	0.01064004	0.01097575	0.01202222

Tabel 6. 8 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data *full*-char (lanjutan)

37	0.01061951	0.01064047	0.01016492	0.01419428
38	0.01058316	0.01099200	0.00979842	0.01287582
39	0.01073370	0.01221766	0.01068794	0.01180839
40	0.01260768	0.01090219	0.01028936	0.01747438
Rata-rata	0.01126395	0.01143739	0.01116046	0.01178143

Pada Tabel 6.8 dilakukan proses dekripsi dengan 4 percobaan, pada percobaan 1 dekripsi keempat *field* yang sudah terenkripsi dengan data *full*-char hasil waktu yang didapatkan 0.01126395 *second* atau 11.264 *milisecond*, kemudian percobaan 2 pada *field* alamat, keterangan rekam medik, dan gejala telah terenkripsi dengan data *full*-char pada *field* nama terenkripsi dengan data kosong hasil waktu yang didapatkan 0.01143739 *second* atau 11.437 *milisecond*, lalu percobaan 3 *field* keterangan rekam medik dan gejala yang terenkripsi dengan data *full*-char sisanya terenkripsi dengan data kosong waktu yang didapatkan 0.01116046 *second* atau 11.160 *milisecond*, percobaan 4 hanya *field* gejala yang terenkripsi dengan data *full*-char sisa *field* yang lain terenkripsi dengan data kosong hasil waktu yang didapatkan 0.01178143 *second* atau 11.781 *milisecond*.



Gambar 6. 10 Grafik perbandingan waktu pemrosesan dekripsi data *full*-char

Pada Gambar 6.10 menjelaskan tentang perbandingan waktu pemrosesan data dari jurnal pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.7.

6.2.8 Pengujian waktu dekripsi ketika field *database* berisi *ciphertext* data dari jurnal

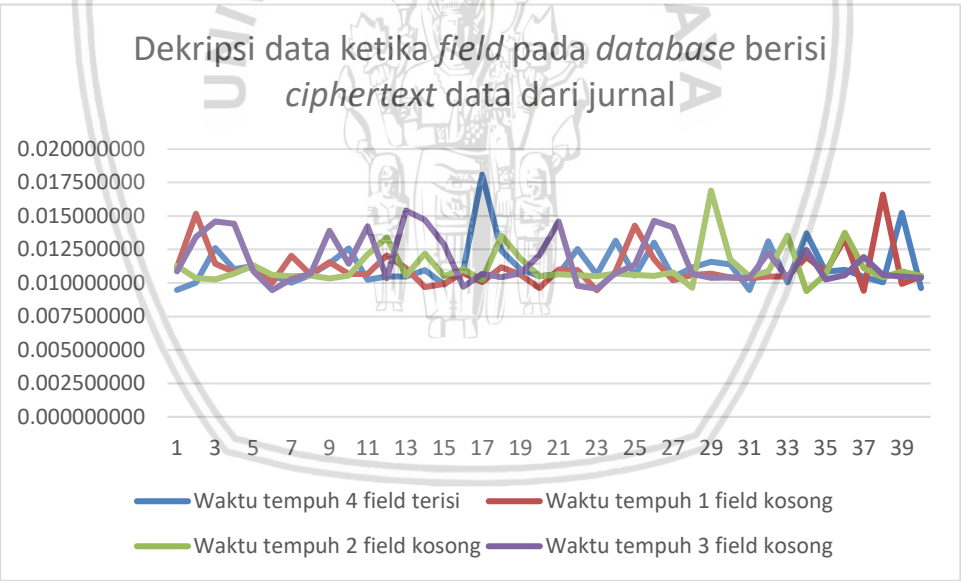
Tabel 6. 9 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data dari jurnal

Percobaan	Waktu tempuh 4 *field* terisi (s)	Waktu tempuh 1 *field* kosong (s)	Waktu tempuh 2 *field* kosong (s)	Waktu tempuh 3 *field* kosong (s)
1	0.00947639	0.01114253	0.01127938	0.01085601
2	0.01003063	0.01517745	0.01035052	0.01343433
3	0.01261024	0.01144360	0.01025301	0.01460055
4	0.01103562	0.01082436	0.01068452	0.01441965
5	0.01127511	0.01123320	0.01130804	0.01088722
6	0.01055921	0.01000027	0.01060968	0.00946485
7	0.01001866	0.01204446	0.01051431	0.01027867
8	0.01059599	0.01061823	0.01056178	0.01067468
9	0.01145515	0.01154624	0.01034197	0.01389620
10	0.01258031	0.01066057	0.01054425	0.01143291
11	0.01024403	0.01066271	0.01213298	0.01422678
12	0.01047154	0.01204959	0.01339328	0.01037019
13	0.01048694	0.01110533	0.01054082	0.01540283
14	0.01095266	0.00967996	0.01219799	0.01471815
15	0.00988694	0.00992286	0.01050875	0.01286128
16	0.01096762	0.01077304	0.01103733	0.00971631
17	0.01808679	0.01006698	0.01022094	0.01068024
18	0.01242464	0.01116434	0.01353526	0.01043519
19	0.01104032	0.01062336	0.01183918	0.01071360
20	0.01050704	0.00961666	0.01051474	0.01207439
21	0.01067383	0.01099841	0.01065715	0.01458430
22	0.01252899	0.01095095	0.01058102	0.00979115
23	0.01062636	0.00944688	0.01050661	0.00958202
24	0.01315465	0.01086499	0.01068751	0.01074011
25	0.01056135	0.01428451	0.01060754	0.01133840
26	0.01301694	0.01177290	0.01053612	0.01464802
27	0.01042322	0.01018502	0.01078886	0.01416733
28	0.01115365	0.01058915	0.00963591	0.01066100
29	0.01158259	0.01068281	0.01689833	0.01037404
30	0.01139057	0.01039799	0.01173526	0.01040911
31	0.00947597	0.01035394	0.01051046	0.01037618
32	0.01310119	0.01047497	0.01085301	0.01224075
33	0.01004175	0.01047283	0.01354595	0.01036078
34	0.01370589	0.01193369	0.00938958	0.01245800
35	0.01084360	0.01082308	0.01064090	0.01025558

Tabel 6. 9 Pengujian waktu dekripsi ketika *field database* berisi *ciphertext* data dari jurnal (lanjutan)

36	0.01098387	0.01333469	0.01375465	0.01056563
37	0.01046085	0.00941567	0.01108908	0.01192642
38	0.01005373	0.01660240	0.01040226	0.01060668
39	0.01524545	0.00992971	0.01087311	0.01048951
40	0.00961666	0.01050447	0.01051773	0.01038430
Rata-rata	0.01133367	0.01110937	0.01116449	0.01167758

Pada Tabel 6.9 dilakukan proses dekripsi dengan 4 percobaan, pada percobaan 1 dekripsi keempat *field* yang sudah terenkripsi dengan data asli sesuai pada jurnal hasil waktu yang didapatkan 0.01133367 *second* atau 11.334 *milisecond*, kemudian percobaan 2 pada *field* alamat, keterangan rekam medik, dan gejala telah terenkripsi dengan data asli pada *field* nama terenkripsi dengan data kosong hasil waktu yang didapatkan 0.01110937 *second* atau 11.109 *milisecond*, lalu percobaan 3 *field* keterangan rekam medik dan gejala yang terenkripsi dengan data asli sisanya terenkripsi dengan data kosong waktu yang didapatkan 0.01116449 *second* atau 11.165 *milisecond*, percobaan 4 hanya *field* gejala yang terenkripsi dengan data asli sisa *field* yang lain terenkripsi dengan data kosong hasil waktu yang didapatkan 0.01167758 *second* atau 11.678 *milisecond*.



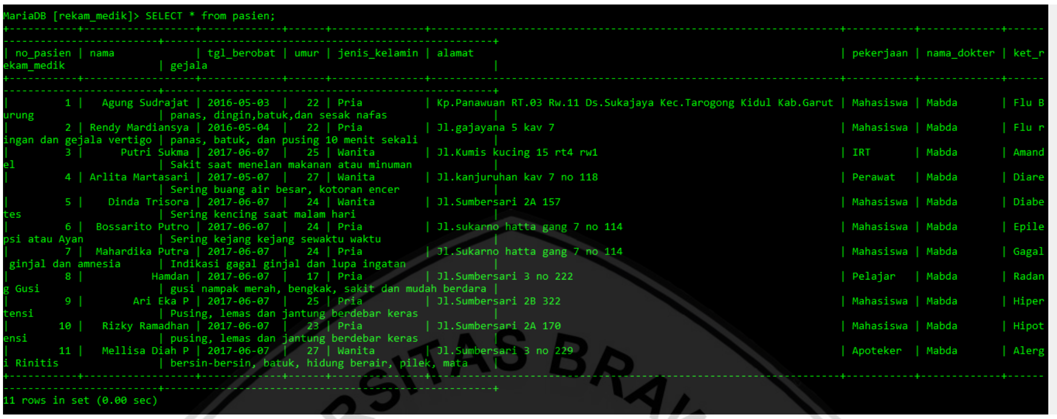
Gambar 6. 11 Grafik perbandingan waktu pemrosesan dekripsi data dari jurnal

Pada Gambar 6.11 menjelaskan tentang perbandingan waktu pemrosesan data dari jurnal pada proses enkripsi, data didapatkan dan dijelaskan pada Tabel 6.8.

6.3 Pengujian Keamanan

Pengujian keamanan dilakukan dengan melakukan akses ke dalam *database*, kemudian menjalankan *query* untuk menampilkan data pasien pada *database* rekam medik.

6.3.1 Data sebelum enkripsi



Gambar 6. 12 Data sebelum enkripsi

Pada Gambar 6.12 dapat dilihat data pada *database* rekam medik sebelum di enkripsi, pada *field* nama, alamat, keterangan rekam medik, dan gejala semua data dapat terbaca dengan jelas.

6.3.2 Data sesudah enkripsi

Gambar 6. 13 Data sesudah enkripsi

Pada Gambar 6.13 dapat dilihat ketika *database* yang sudah di enkripsi, data yang sebelumnya dapat dilihat seperti pada gambar 6.5 tidak dapat terlihat

atau tidak dapat dibaca pada *field* nama, alamat, keterangan rekam medik, dan gejala.

6.4 Pengujian Validasi Enkripsi dan Dekripsi

Pengujian validasi enkripsi dan dekripsi digunakan untuk memastikan data dienkripsi sesuai dengan algoritme yang digunakan. Untuk mendapatkan hasil validasi tersebut dapat dibuktikan dengan cara mencocokkan hasil dekripsi dengan *plaintext* sebelum dilakukan proses enkripsi. Dalam melakukan proses enkripsi dibutuhkan sebuah *key*, *key* yang digunakan pada pengujian ini adalah "coba123ah", dan dibutuhkan *plaintext* dengan menggunakan 5 *plaintext* yang masing-masing berbeda.

Plaintext masukan = "Jl.Sumbersari 3 no 229"

Tabel 6. 10 Pengujian validasi enkripsi dan dekripsi 1

Key	*Plaintext* sebelum di enkripsi	*Plaintext* sesudah di dekripsi
coba123ah	Jl.Sumbersari 3 no 229	Jl.Sumbersari 3 no 229

Plaintext masukan = "a"

Tabel 6. 11 Pengujian validasi enkripsi dan dekripsi 2

Key	*Plaintext* sebelum di enkripsi	*Plaintext* sesudah di dekripsi
coba123ah	a	a

Plaintext masukan = "!@#$%^&*()[];',./`"

Tabel 6. 12 Pengujian validasi enkripsi dekripsi 3

Key	*Plaintext* sebelum di enkripsi	*Plaintext* sesudah di dekripsi
coba123ah	!@#$%^&*()[];',./`	!@#$%^&*()[];',./`

Plaintext masukan = "Kp.Panawuan RT.03 Rw.11 Ds.Sukajaya Kec.Tarogong Kidul Kab.Garut"

Tabel 6. 13 Pengujian validasi enkripsi dan dekripsi 4

Key	*Plaintext* sebelum di enkripsi	*Plaintext* sesudah di dekripsi
coba123ah	Kp.Panawuan RT.03 Rw.11 Ds.Sukajaya Kec.Tarogong Kidul Kab.Garut	Kp.Panawuan RT.03 Rw.11 Ds.Sukajaya Kec.Tarogong Kidul Kab.Garut

Plaintext masukan = "123456789qwertyuiopasdfghjklzxcvbnm!@#$%^&*()[];',./`12345678900"

Tabel 6. 14 Pengujian validasi enkripsi dan dekripsi 5

Key	*Plaintext* sebelum di enkripsi	*Plaintext* sesudah di dekripsi

Tabel 6. 14 Pengujian validasi enkripsi dan dekripsi 5 (lanjutan)

coba123ah	123456789qwertyuiopasdfghjklzxcvbnm!@#$%^&*()[];',./`12345678900	123456789qwertyuiopasdfghjklzxcvbnm!@#$%^&*()[];',./`12345678900

6.5 Pengujian Fungsional

6.5.1 Pengujian fungsional fungsi *login*

Tabel 6. 15 Pengujian fungsional fungsi *login*

Nama Kasus Uji	*Login*
Prosedur	- Membuka halaman *login* - Mengisi *username* dan *password* yang sesuai pada *database* - Menekan tombol "Masuk"
Hasil yang diharapkan	Sistem akan mengarahkan pengguna ke halaman utama sesuai dengan *role*
Hasil	Sistem mengarahkan pengguna ke halaman utama sesuai dengan *role*
Status	Valid

Tabel 6. 16 Pengujian fungsional fungsi *login* alternatif 1

Nama Kasus Uji	*Login* alternatif 1
Prosedur	- Membuka halaman *login* - Mengisi *username* dan *password* yang tidak sesuai pada *database* - Menekan tombol "Masuk"
Hasil yang diharapkan	Sistem akan mengembalikan pengguna ke halaman *login*
Hasil	Sistem mengembalikan pengguna ke halaman *login*
Status	Valid

6.5.2 Pengujian fungsional fungsi *insert* data

Tabel 6. 17 Pengujian fungsional fungsi *insert* data

Nama Kasus Uji	Insert data

Tabel 6. 17 Pengujian fungsional fungsi *insert* data (lanjutan)

Prosedur	- Membuka halaman *insert* data dengan menekan menu *insert* data - Memasukkan data rekam medik pasien - *Submit* data dengan menekan tombol "*submit*"
Hasil yang diharapkan	Sistem akan menampilkan pesan "*success*" dan mengarahkan pengguna ke halaman pilih
Hasil	Sistem menampilkan pesan "*success*" dan mengarahkan pengguna ke halaman pilih
Status	Valid

Tabel 6. 18 Pengujian fungsional fungsi *insert* data alternatif 1

Nama Kasus Uji	Insert data alternatif 1
Prosedur	- Membuka halaman *insert* data dengan menekan menu *insert* data - Memasukkan data rekam medik pasien tidak lengkap - *Submit* data dengan menekan tombol "*submit*"
Hasil yang diharapkan	Sistem akan menampilkan pesan "*please fill out the field*" agar pengguna mengisi semua form yang tersedia
Hasil	Sistem menampilkan pesan "*please fill out the field*" agar pengguna mengisi semua form yang tersedia
Status	Valid

6.5.3 Pengujian fungsional fungsi *show* data

Tabel 6. 19 Pengujian fungsional *show* data

Nama Kasus Uji	Menampilkan semua data
Prosedur	Menekan menu "*Show* Data" pada halaman pilih dengan kondisi *database* sudah terenkripsi dengan hak akses admin
Hasil yang diharapkan	Sistem akan menampilkan semua data yang sudah terenkripsi pada *database* rekam medik
Hasil	Sistem menampilkan semua data yang sudah terenkripsi pada *database* rekam medik

Tabel 6. 19 Pengujian fungsional *show* data (lanjutan)

Status	Valid

Tabel 6. 20 Pengujian fungsional *show* data alternatif 1

Nama Kasus Uji	Menampilkan semua data alternatif 1
Prosedur	Menekan menu "Show Data" pada halaman pilih dengan kondisi *database* belum terenkripsi dengan hak akses admin
Hasil yang diharapkan	Sistem akan menampilkan semua data yang belum terenkripsi pada *database* rekammedik
Hasil	Sistem menampilkan semua data yang belum terenkripsi pada *database* rekammedik
Status	Valid

Tabel 6. 21 Pengujian fungsional show data alternatif 2

Nama Kasus Uji	menampilkan semua data alternatif 2
Prosedur	Menekan menu "Show Data" pada halaman pilih dengan kondisi *database* yang sudah terenkripsi dengan hak akses pengguna
Hasil yang diharapkan	Sistem akan menampilkan semua data yang sudah terenkripsi pada *database* rekammedik
Hasil	Sistem menampilkan semua data yang sudah terenkripsi pada *database* rekammedik
Status	Valid

6.5.4 Pengujian fungsional fungsi enkripsi

Tabel 6. 22 Pengujian fungsional enkripsi

Nama Kasus Uji	Melakukan operasi enkripsi
Prosedur	Menekan menu "Enkripsi" pada halaman pilih dengan hak akses admin
Hasil yang diharapkan	Sistem akan melakukan proses enkripsi pada semua data di *database* rekam medik
Hasil	Sistem melakukan proses enkripsi pada semua data di *database* rekam medik
Status	Valid

6.5.5 Pengujian fungsional fungsi dekripsi

Tabel 6. 23 Pengujian fungsional dekripsi

Nama Kasus Uji	Melakukan operasi dekripsi
Prosedur	Menekan menu "Dekripsi" pada halaman pilih dengan hak akses admin
Hasil yang diharapkan	Sistem akan melakukan proses dekripsi pada semua data di *database* rekam medik
Hasil	Sistem melakukan proses dekripsi pada semua data di *database* rekam medik
Status	Valid

6.5.6 Pengujian fungsional fungsi *logout*

Tabel 6. 24 Pengujian fungsional *logout*

Nama Kasus Uji	*Logout*
Prosedur	Pengguna menekan menu *logout* pada halaman pilih
Hasil yang diharapkan	Sistem akan mengarahkan ke halaman *logout*
Hasil	Sistem mengarahkan ke halaman *logout*
Status	Valid

6.6 Pengujian Non-Fungsional

6.6.1 Pengujian non-fungsional *portability*

Tabel 6. 25 Pengujian non-fungsional *portability*

Nama Kasus Uji	*Portability*
Prosedur	Sistem dijalankan pada berbagai web browser seperti Mozilla Firefox, Google Chrome, Opera, dan Microsoft Edge pada sistem operasi windows
Hasil yang diharapkan	Sistem akan berjalan lancar dan tampilan dapat menyesuaikan
Hasil	Sistem berjalan lancar dan tampilan dapat menyesuaikan
Status	Valid

6.6.2 Pengujian non-fungsional *performance*

Tabel 6. 26 Pengujian non-fungsional *performance*

Nama Kasus Uji	*Performance*

Tabel 6. 26 Pengujian non-fungsional *performance* (lanjutan)

Prosedur	Menjalankan semua fungsi yang terdapat pada *website* rekam medik
Hasil yang diharapkan	Waktu respon sistem kurang dari 10 *second* atau 10000 *milisecond*
Hasil	Semua fungsi yang terdapat pada sistem dapat berjalan lancar dengan waktu respon kurang dari 10 *second* atau 10000 *milisecond*
Status	Valid

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil penelitian implementasi algoritme SPECK dan SHA-3 pada *database* rekam medik, dapat disimpulkan beberapa hal yaitu:

1. Algoritme SPECK 128/128 dapat diimplementasikan pada *field* nama, alamat, keterangan rekam medik, dan gejala pada tabel pasien *database* rekam medik, dan algoritme SHA-3 dapat diimplementasikan pada *field password* pada tabel *user database* rekam medik.
2. Algoritme SPECK 128/128 yang dibuat oleh peneliti memiliki hasil *output ciphertext* yang sama dengan *test vector* pada jurnal, dan hasil *output digest library* SHA-3 yang digunakan memiliki hasil yang sama dengan *test vector* pada jurnal, maka kedua algoritme yang digunakan pada penelitian ini bersifat *valid*.
3. Algoritme SPECK dapat berjalan dengan baik dan cepat dibuktikan dengan pengujian *performance*. Pada pengujian *performance* dilakukan pada *field* nama, alamat, keterangan rekam medik, dan gejala. Setiap *field* diuji dengan menggunakan data 1-char, *full*-char, keempat *field* kosong, dan data dari jurnal. Percobaan setiap data dilakukan sebanyak 40 kali. Hasil yang didapatkan pada proses enkripsi menghasilkan waktu berkisar antara 10-12 *milisecond*, dan pada proses dekripsi menghasilkan waktu berkisar antara 10-12 *milisecond*.
4. Pada pengujian keamanan, data pada *field* nama, alamat, keterangan rekam medik, dan gejala pada tabel pasien sebelum di enkripsi dengan menggunakan algoritme SPECK dapat terlihat dengan jelas, dan ketika data sesudah di enkripsi tidak dapat dibaca atau berupa data *hexadecimal*.
5. Pada pengujian fungsional dan non-fungsional sistem, fungsi-fungsi pada *website* rekam medik dapat berjalan sesuai dengan fungsinya, sistem dapat berjalan pada berbagai web browser, dan waktu untuk melakukan akses setiap halaman *website* tidak lebih dari 10 *second* atau 10000 *milisecond*.

7.2 Saran

Setelah diketahui hasil penelitian, peneliti memberikan beberapa saran untuk penelitian selanjutnya, antara lain:

1. Aplikasi *website* dapat dikembangkan seperti melengkapi fitur-fitur pada *website* rekam medik, dan memperbaiki tampilan *website*.
2. Aplikasi *website* dan *database* pada penelitian ini diimplementasikan pada *localhost*, untuk penelitian selanjutnya algoritme dapat diimplementasikan pada *website* dan *database* asli.

3. Pada penelitian ini dapat membandingkan algoritme SPECK dengan algoritme *block cipher* yang lain yang bertujuan untuk membandingkan kinerja algoritme mana yang lebih baik.

DAFTAR PUSTAKA

Ariyus, D., 2008. *Pengantar Ilmu Kriptografi: Teori Analisis & Implementasi*. 1st ed. [online] Yogyakarta: PENERBIT ANDI. Available at: <https://books.google.co.id/books?hl=en&lr=&id=3SSTJONEmX0C&oi=fnd&pg=PR3&dq=info:hsyDDvhY3hMJ:scholar.google.com&ots=uiEArbqIjz&sig=iewXuPgk6rHozUomMCmzXRPFmRQ&redir_esc=y#v=onepage&q&f=false>.

Beaulieu, R., Shors, D., Smith, J. dan Treatman-clark, S., 2013. *The simon and speck families of lightweight block ciphers*. *Cryptology ePrint Archive*.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B. dan Wingers, L., 2015a. *Simon and Speck: Block Ciphers for the Internet of Things*. *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, [online] (July), pp.1–6. Available at: <http://dl.acm.org/citation.cfm?doid=2744769.2747946>.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B. dan Wingers, L., 2015b. *The simon and speck block ciphers on avr 8-bit microcontrollers*. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Beaulieu, R. dan Treatman-clark, S., 2013. *Implementation and Performance of the Simon and Speck Lightweight Block Ciphers on ASICs*. *Cryptology ePrint Archive*.

Biryukov, A., Roy, A. dan Velichkov, V., 2015. Differential analysis of block ciphers Simon and Speck. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Dworkin, M.J., 2015. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. *Draft FIPS PUB 202*, [online] (Agustus). Available at: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.

Handiwidjojo, W., 2009. *Rekam medis elektronik*. *Eksis*.

Jamaludin, F., 2016. *Data rekam medis mahal, malware ini justru serang sektor kesehatan*. [online] www.merdeka.com. Available at: <https://www.merdeka.com/teknologi/data-rekam-medis-mahal-malware-ini-justru-serang-sektor-kesehatan.html>.

Kurniawan, F., Kusyanti, A. dan Nurwarsito, H., 2017. *Analisis dan Implementasi Algoritma SHA-1 dan SHA-3 pada Sistem Autentikasi Garuda Training Cost.* 1(9), pp.803–812.

Kusuma, H., 2011. *Database Oracle Untuk Pemula*. *Eastern Light Publication*.

Menteri Kesehatan, R., 2008. *Peraturan Menteri Kesehatan Republik Indonesia No 269/Menkes/PER/2008 tentang Rekam Medis*. *Menteri Kesehatan*, .

Pertiwi, W.K., 2017. *Hacker Curi Data Pasien Klinik Bedah Plastik, Ancam Sebar Foto*. [online] www.kompas.com. Available at: <https://tekno.kompas.com/read/2017/10/25/13050037/hacker-curi-data-pasien-klinik-bedah-plastik-ancam-sebar-foto>.

Raharjo, B., 2015. *Mudah Belajar PYTHON untuk Aplikasi Desktop dan Web*. 1st ed. Bandung: INFORMATIKA Bandung.

Santoso, K.I. dan Priyoatmoko, W., 2016. *Pengamanan Data Mysql pada E-commers dengan Algoritma Aes 256*. *Seminar Nasional Sistem Informasi Indonesia*, 1(1), pp.1–8.

Setyaningsih, E., 2015. *Kriptografi & Implementasinya menggunakan MATLAB*. 1st ed. Yogyakarta: PENERBIT ANDI.

Sudrajat, A. dan Gunadhi, E., 2016. *Pengamanan Data Rekam Medis Pasien Menggunakan Kriptografi Vigènere Cipher*. 13, pp.295–301.

Wibisono, S. dan Munawaroh, S., 2012. *Sistem Informasi Manajemen Puskesmas (Simpuskesmas) berbasis Cloud Computing*. *Jurnal Teknologi Informasi Dinamik*, 17(2), pp.141–146.

Yusuf, O., 2016. *'Hacker' Jual Rekam Medis 655.000 Pasien*. [online] www.kompas.com. Available at: <https://tekno.kompas.com/read/2016/06/29/12010097/.hacker.jual.rekam.medis.655.000.pasien>.