# BAB II

# TINJAUAN PUSTAKA DAN LANDASAN TEORI

## A. Tinjauan Pustaka

Penelitian tentang meningkatkan keamanan algoritma *hash* dengan menggunakan metode pengacakan *salt* yang dipublikasikan pada tahun 2018. Penelitian ini membahas bagaimana meningkatkan penggunaan algoritma *hash* dengan melakukan pengacakan posisi antara *password* dan *salt* sebelum melakukan proses *hashing* (Karrar *et al.*, 2018).

Penelitihan tersebut dilakukan dengan menggunakan bahasa pemrograman C# dan menggunakan algoritma SHA384 sebagai objek pengujian. *Future scope* dari penelitian tersebut mengharapkan ada penelitian lebih lanjut dengan menggunakan algoritma yang lain dan kombinasi *password* dan *salt* yang berbeda dengan memanfaatkan *salt*.

Penelitian pada tahun 2018 tentang penggunaan algoritma *hash* sebagai metode autentikasi pada aplikasi web. Penelitian ini bertujuan

untuk meningkatkan keamanan sistem informasi berbasis web dengan menggunakan algoritma *hash* SHA 512. Pada penelitian ini autentikasi pada sistem informasi *low-end* yang masih menggunakan PHP versi < 5.2 diperbaiki menggunakan SHA 512. Penelitian tersebut hanya memeperbarui algoritma yang digunakan dengan algoritma yang lebih baru. Algoritma SHA 512 tentu lebih baik dibanding MD5 karena memiliki jumlah bit yang lebih lebar (Riadi, 2018).

Penelitian pada tahun 2015 yang membahas tentang penggunaan *salt* dan *differential masking* untuk meningkatkan keamanan algoritma *hash*. Penelitian ini memiliki tujuan untuk meningkatkan kemanan proses autentikasi yang menggunakan algoritma *hash*. Penelitian ini mengunakan *salt* sebagai *string* tambahan terhadap *password* dan *differential masking*. *Differential masking* merupakan penggunaan *password* palsu yang di asosiasikan terhadap akun pengguna, sehingga *attacker* akan mendapatkan

beberapa kemungkinan *password* dimana sebagian adalah *password* palsu (Kharod, Sharma and Sharma, 2015).

Penelitian tahun 2014 tentang mengamankan *password* yang disimpan dalam *database*. Penelitian ini menjelaskan bahwa pada saat ini banyak sekali sistem yang berjalan secara *online* dan menyimpan *password* secara *online*. *password* yang disimpan secara *online* membutuhkan keamanan lebih untuk menghindari pencurian *password*. Penelitian ini menambahkan penggunaan *salt* untuk meningkatkan keamanan *password* yang disimpan dalam *diatabase* dengan dipadukan dengan teknik *dual hashing (Ogini and Ogwara, 2014)*.



Rangkuman dari beberapa penelitian terdahulu akan ditampilkan pada tabel 1 di bawah:

Tabel 1: Tabel Penelitian Terdahulu

| No | Jurnal | Judul | Tahun | Penulis | Metode | Algoritma | Keterangan |
|---|---|---|---|---|---|---|---|
| 1 | IJCST Vol . 9 | *Enhancing Salted password Hashing Technique Using Swapping Elements* | 2018 | Dr. Abdelrahman Karrar, Talal Almutiri, Sultan Algrafi, | 1. Menerima *password* dan *salt* dari *user*<br>2. Melakukan *generate salt*<br>3. Melakukan *reversing salt* | Tidak disebutkan. | Dapat diaplikasikan dalam berbagai macam algoritma *hash*. |

| No | Jurnal | Judul | Tahun | Penulis | Metode | Algoritma | Keterangan |
|---|---|---|---|---|---|---|---|
| | | *in an Array Algorithm* | | Naif Alalwi, Ammar Alharbi | 4. Melakukan *rearrangement password* dan *salt*<br>5. Mengirim *hash input* ke fungsi *hashing*<br>6. Melakukan proses *rearrangement salt* | | |

| No | Jurnal | Judul | Tahun | Penulis | Metode | Algoritma | Keterangan |
|---|---|---|---|---|---|---|---|
| | | | | | 7. Menyimpan hasil *hash* dan *salt* ke *database* | | |
| 2 | 4th International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions (ICRITO) | *An Improved Hashing Based password Security Scheme* | 2015 | Seema Kharod, Nidhi Sharma, Alok Sharma | Menggunakan *salt* untuk memperkuat fungsi *hash* dan menambahkan *password* palsu kedalam *database* untuk | Tidak disebutkan | Dapat diaplikasikan dalam berbagai macam algoritma *hash*. |

| No | Jurnal | Judul | Tahun | Penulis | Metode | Algoritma | Keterangan |
|---|---|---|---|---|---|---|---|
| | | *Using Salting and Differential Masking* | | | memperlambat *attacker* mendapatkan *password* asli. | | |
| 3 | International Journal of Cyber-Security and Digital | *Analysis of Secure Hash Algorith* | 2018 | Meiliana Sumagita and Imam Riadi | Menggunakan algoritma yang lebih baru untuk memperkuat *low-* | SHA512 | Lebih menekankan kepada pembaruan algoritma dan membandingkan |

| No | Jurnal | Judul | Tahun | Penulis | Metode | Algoritma | Keterangan |
|---|---|---|---|---|---|---|---|
| | Forensics (IJCSDF) | *m (SHA) 512 for Encryption Process on Web Based Application* | | | *end* sistem. | | kekuatannya. |

| No | Jurnal | Judul | Tahun | Penulis | Metode | Algoritma | Keterangan |
|---|---|---|---|---|---|---|---|
| 4 | IPASJ International Journal of Computer Science (IIJCS) | *Securing database passwords using a combination of hashing and salting techniques* | 2014 | Nicholas Oluwole Ogini, Noah Oghenefego Ogwara | Menggunakan *salt* yang dikombinasikan dengan *password* untuk dilakukan proses *hashing*. Proses *hashing* yang dilakukan dua kali. | MD5 | - |

Pada Tabel 1 telah ditampilkan rangkuman dari beberapa penelitian terdahulu. Penelitian ini berbeda dari segi konsep, namun memiliki tujuan yang sama yaitu untuk meningkatkan kekuatan pengaplikasian algoritma *hash* sehingga dapat meningkatkan keamanan proses autentikasi. Penelitian ini merupakan pengembangan dari penelitian-penelitian yang telah dilakukan oleh beberapa peneliti sebelumnya, sehingga dalam beberapa bagian akan merujuk pada penelitian sebelumnya.

Pada penelitian ini akan lebih fokus kepada bagaimana cara menemukan kombinasi terbaik dari beberapa kombinasi yang telah dirancang sebelumnya. Fokus utama pada penelitian ini ada pada kombinasi *password* dan *salt* yang akan diimplementasikan dengan menggunakan bahasa pemrograman PHP. PHP dipilih sebagai bahasa pendukung dalam penelitian ini dikarenakan sampai saat ini banyak sistem informasi berbasis web yang menggunakan bahasa

pemrograman PHP dan membutuhkan keamanan terhadap autentikasi penggunanya.

## B. Landasan Teori

Adapun landasan teori dalam penelitian ini akan dijelaskan secara garis besar dengan teori-teori yang menjadi dasar atau acuan pada penelitian ini. Teori-teori tersebut diantaranya adalah sebagai berikut:

### 1. Autentikasi dan Privasi

Menurut Stallings dan Brown (Stallings and Brown, 2015), masalah dengan banyak tumpang tindih dengan keamanan komputer adalah masalah privasi. Di satu sisi, skala dan keterkaitan informasi pribadi yang dikumpulkan dan disimpan dalam sistem informasi telah meningkat secara dramatis, dimotivasi oleh penegakan hukum, keamanan nasional, dan insentif ekonomi. Insentif ekonomi mungkin

merupakan kekuatan pendorong utama. Dalam ekonomi informasi global, ada kemungkinan bahwa aset elektronik yang paling bernilai secara ekonomi adalah kumpulan informasi tentang individu. Di sisi lain, individu menjadi semakin sadar akan sejauh mana lembaga pemerintah, bisnis, dan bahkan pengguna internet memiliki akses ke informasi pribadi mereka dan perincian pribadi tentang kehidupan dan kegiatan mereka. Kekhawatiran tentang sejauh mana privasi seseorang telah dijaga menyebabkan lahirnya berbagai pendekatan hukum dan teknis untuk memperkuat hak privasi seseorang.

Menurut Preneel dan Lowry (Preneel and Lowry, 2003), aktivitas untuk memberikan keamanan terhadap informasi atau perlindungan privasi sudah ada sejak dulu, sama tua dengan keberadaan pesan itu sendiri. Kecerdasan manusia menemukan banyak cara untuk menyembunyikan informasi. Salah satu diantaranya adalah menggunakan Steganografi. Steganografi merupakan penyembunyian

keberadaan pesan, kode, di mana kata-kata atau kombinasi kata digantikan oleh simbol tetap, dan kriptologi atau sandi, di mana informasi diubah untuk membuatnya tidak berguna bagi lawan. Perbedaan antara dua yang terakhir agak halus, dan dapat dibuat pada fakta bahwa kode membagi informasi sesuai dengan batas semantik, sementara cipher beroperasi pada bongkahan informasi secara independen dari interpretasi linguistik.

Evolusi teknologi pertukaran informasi dari pengiriman pesan kertas menggunakan POS sampai pada penggunaan email dan layanan lainnya tentunya telah meningkatkan kerentanan informasi untuk dapat dilihat oleh orang-orang yang tidak memiliki hak atas pesan tersebut. Kriptologi adalah satu-satunya solusi yang mampu menangani hal tersebut. Dengan menggunakan kriptologi keamanan pada pesan yang dikirimkan akan meningkat dan dipastikan hanya dapat dibaca oleh orang yang memiliki kunci dari pesan tersebut.

Terlepas dari penyembunyian atau perlindungan privasi, hal lain yang sama pentingnya adalah konten pesan dan keaslian pesan. Pesan yang dikirimkan selain harus tidak dapat dibaca oleh pihak luar namun juga harus dapat dipastikan bahwa pesan tidak mengalami perubahan dalam perjalanan. Upaya-upaya untuk memastikan sebuah pesan adalah asli dan otentik dikenal dengan autentikasi. Seorang *attacker* yang mencoba mengubah konten atau asal informasi disebut penyerang aktif. Fakta bahwa kepentingan relatif dari ancaman ini telah meningkat dapat diilustrasikan dengan munculnya program perangkat lunak berbahaya yang difungsikan untuk keperluan tersebut.



Autentikasi adalah proses memastikan sebuah properti adalah asli, bisa diverifikasi dan dipercaya; keyakinan dalam keabsahan transmisi, pesan, atau pengirim pesan. Autentikasi dapat memverifikasi bahwa pengguna adalah pengguna yang seharusnya dan setiap

masukan yang masuk ke sistem berasal dari sumber tepercaya (Stallings and Brown, 2015).

Autentikasi merupakan proses yang sangat penting karena selain menjaga informasi dari pengguna yang tidak berhak, autentikasi juga menjaga integritas data (Ramos Brandão, 2018). Apabila ada pengguna yang tidak berhak dapat mengakses module-module penting didalam sistem, pengguna tersebut dapat memanipulasi data sehingga berpengaruh terhadap integritas dari data tersebut.

Autentikasi pada dasarnya bertujuan untuk memberikan perlindungan terhadap pihak-pihak yang berkomunikasi dari serangan pihak ketiga. Namun, ancaman yang berbeda muncul ketika kedua pihak yang berkomunikasi saling tidak percaya dan mencoba untuk melakukan penolakan. Dengan kata lain berarti pengirim atau penerima akan mencoba untuk mengubah pesan dan / atau menolak untuk mengirim atau menerima pesan tertentu. Dalam dokumen kertas,

perlindungan terhadap jenis serangan ini ditawarkan oleh tanda tangan tulisan tangan. Jelas bahwa dalam hal pesan elektronik, nama sederhana di akhir pesan tidak menawarkan perlindungan sama sekali. Dari penjelasan diatas dapat disimpulkan fakta bahwa fotokopi dokumen dengan tanda tangan manual tidak memiliki nilai, karena orang lain selalu dapat menghasilkan dokumen palsu dengan melakukan potong dan tempel. Contoh tipuan penipuan ini misalnya adalah komunikasi elektronik pesanan kepada penjual saham. Pelanggan dapat memesan untuk membeli saham perusahaan X. Jika beberapa hari kemudian transaksi ternyata buruk, ia akan mencoba untuk menolak pesanannya. Jika di sisi lain, transaksi berhasil, penjual saham mungkin mengklaim bahwa ia tidak pernah menerima pesanan dengan maksud untuk menjaga keuntungan. Dalam hal apabila terjadi perselisihan, pihak ketiga (hakim), harus mengambil keputusan. Solusi

teknis yang elegan untuk masalah ini ditawarkan oleh konsep skema tanda tangan digital.

**a) Regulasi Hukum Mengenai Privasi**

Sejumlah organisasi dan pemerintah telah memperkenalkan undang-undang dan peraturan yang dimaksudkan untuk melindungi privasi individu. Dengan adanya undang-undang tersebut harapannya keamanan akan penggunaan informasi pribadi seseorang akan lebih terjaga. Setiap pelanggar hak privasi dapat dijerat dengan tuntutan hukum dikarenakan ada undang-undang yang dilanggar. Misalnya di Indonesia terdapat undang-undang Informasi dan Transaksi Elektronik (ITE).



Pada tahun 1998, Uni Eropa mengadopsi arahan tentang Perlindungan Data untuk memastikan bahwa negara-negara anggota melindungi hak-hak privasi dasar ketika memproses informasi pribadi,

dan untuk mencegah negara-negara anggota dari membatasi aliran bebas informasi pribadi di dalam lingkup Uni Eropa. Arahan itu sendiri bukan hukum, tetapi mengharuskan negara-negara anggota untuk memberlakukan hukum yang mencakup ketentuan-ketentuannya.

Mernurut Stallings dan Brown(Stallings and Brown, 2015), arahan yang ditetapkan oleh Uni Eropa terhadap negara-negara anggotanya disusun berdasarkan prinsip-prinsip penggunaan informasi pribadi berikut:

- *Notice*

  Organisasi harus memberi tahu kepada individu mengenai informasi pribadi apa yang mereka kumpulkan, penggunaan akan informasi yang dikumpulkan, dan pilihan apa yang mungkin dimiliki individu tersebut.

- *Concent*

  Individu harus dapat memilih oleh siapa dan bagaimana informasi pribadi mereka digunakan, atau apabila terbuka kepada pihak ketiga maka siapa pihak ketiga yang memiliki akses terhadap informasi mereka. Mereka memiliki hak untuk tidak mengumpulkan informasi sensitif atau digunakan tanpa izin tertulis, termasuk ras, agama, kesehatan, keanggotaan serikat pekerja, kepercayaan, dan kehidupan seks.

- *Consistency*

  Organisasi dapat menggunakan informasi pribadi hanya sesuai dengan ketentuan yang telah diberitahukan sebelumnya kepada subjek dan pilihan apa pun sehubungan dengan penggunaannya dilakukan oleh subjek.

- *Access*

  Individu harus memiliki hak dan kemampuan untuk mengakses informasi mereka dan dapat memperbaiki, memodifikasi, atau menghapus bagian yang tidak sesuai dengan individu tersebut.

- *Security*

  Organisasi harus memberikan keamanan yang memadai, menggunakan cara teknis dan cara lainnya, untuk melindungi integritas dan kerahasiaan informasi pribadi.

- *Onward Transfer*

  Pihak ketiga yang menerima informasi pribadi harus memberikan tingkat perlindungan privasi yang sama dengan organisasi yang darinya informasi tersebut diperoleh.

- *Enforcement*

  Arahan untuk memberikan hak berupa tindakan pribadi untuk data-data pribadi yang bersangkutan ketika organisasi tidak mengikuti hukum. Selain itu, setiap anggota Uni Eropa memiliki lembaga penegak regulasi yang peduli dengan penegakan hak privasi.

Undang-undang privasi pertama yang diadopsi di Amerika Serikat adalah Privacy Act of 1974, yang menangani informasi pribadi yang dikumpulkan dan digunakan oleh agen-agen federal. Menurut Stallings dan Brown (Stallings and Brown, 2015), undang-undang ini dimaksudkan untuk:



- Memberikan ijin kepada individu untuk menentukan catatan apa yang berkaitan dengan mereka yang dikumpulkan, dipelihara, digunakan, atau disebarluaskan.

- Memberikan ijin kepada individu untuk melarang catatan yang diperoleh untuk satu tujuan digunakan untuk tujuan lain tanpa persetujuan.
- Memberikan ijin kepada individu untuk mendapatkan akses ke catatan yang berkaitan dengan mereka dan untuk memperbaiki dan mengubah catatan tersebut sebagaimana mestinya.
- Memastikan bahwa agensi mengumpulkan, memelihara, dan menggunakan informasi pribadi dalam proses untuk memastikan bahwa informasi tersebut terkini, memadai, relevan, dan tidak berlebihan untuk penggunaan yang dimaksudkan.
- Memberikan hak tindakan pribadi kepada individu yang informasi pribadinya tidak digunakan sesuai dengan Undang-Undang.

Di Indonesia undang-undang yang menangani seputaran privasi dan keamanan data dalam proses transaksi elektronik diatur dalam Undang Undang Informasi dan Transaksi Elektronik (UU ITE). UU ITE atau disebut juga undang-undang no. 11 tahun 2008 merupakan undang-undang yang mengatur tentang informasi serta transaksi elektronik, atau teknologi informasi secara umum. UU ini memiliki yurisdiksi yang berlaku untuk setiap orang yang melakukan perbuatan hukum sebagaimana diatur dalam undang-undang ini, baik yang berada di wilayah Indonesia maupun di luar wilayah hukum Indonesia, yang memiliki akibat hukum di wilayah hukum Indonesia dan/atau di luar wilayah hukum Indonesia dan merugikan kepentingan Indonesia. Pada tahun 2019 undang-undang no.11 tahun 2008 direvisi dengan undang-undang yang baru, yaitu undang-undang no. 19 tahun 2019. Undang-undang ini mengatur hal yang sama yaitu mengenai informasi dan

transaksi elektronik dan disebut sebagai UU ITE hanya saja terdapat beberapa perubahan didalamnya.

UU ITE memiliki asas bahwa pemanfaatan Teknologi ITE dilaksanakan berdasarkan asas kepastian hukum, manfaat, kehati-hatian, iktikad baik, dan kebebasan memilih teknologi atau netral teknologi. UU ITE sangat mendukung terhadap keamanan data pribadi seseorang, misalnya pada UU ITE atau undang-undang no 19 tahun 2019 pasal 26 ayat 1 yang menyatakan bahwa penggunaan setiap informasi melalui media elektronik yang menyangkut data pribadi seseorang harus dilakukan atas persetujuan orang yang bersangkutan.



Seperti halnya semua undang-undang dan peraturan privasi, ada pengecualian dan ketentuan yang melekat pada Undang-Undang ini, seperti investigasi kriminal, masalah keamanan nasional, dan konflik antara hak privasi individu yang bersaing.

**b) Privasi Penggunaan Komputer**

*Common Criteria Privacy Spesification* (Popp and Poindexter, 2006) telah mencakup definisi dari serangkaian persyaratan fungsional dalam kelas privasi yang harus diimplementasikan dalam sistem yang dapat dikatakan sebagai sistem yang tepercaya. Tujuan dari fungsi privasi adalah untuk memberikan perlindungan pengguna terhadap penemuan dan penyalahgunaan identitas oleh pengguna lain. Spesifikasi ini adalah panduan yang berguna untuk bagaimana merancang fungsi dukungan privasi sebagai bagian dari sistem komputer. Berikut merupakan penguraian privasi menjadi empat area utama, yang masing-masing memiliki satu atau lebih fungsi spesifik:



- ***Anonimity***: Memastikan bahwa pengguna dapat menggunakan sumber daya atau layanan tanpa mengungkapkan identitas pengguna. Secara khusus, ini berarti bahwa pengguna atau subjek lain tidak dapat menentukan identitas pengguna yang

terikat pada suatu subjek (misalnya proses atau grup pengguna) atau operasi. Lebih lanjut berarti bahwa sistem tidak akan meminta nama asli pengguna. Anonimitas tidak perlu bertentangan dengan fungsi otorisasi dan kontrol akses, yang terikat pada ID pengguna berbasis komputer, bukan pada informasi pengguna pribadi.

- ***Pseudonymity***:Memastikan bahwa pengguna dapat menggunakan sumber daya atau layanan tanpa mengungkapkan identitas penggunanya, tetapi masih dapat bertanggung jawab atas penggunaan itu. Sistem akan menyediakan alias untuk mencegah pengguna lain menentukan identitas pengguna, tetapi sistem harus dapat menentukan identitas pengguna dari alias yang ditugaskan.

- ***Unlikability***: Memastikan bahwa pengguna dapat menggunakan beberapa sumber daya atau layanan tanpa orang lain dapat menautkan penggunaan ini bersama-sama.

- ***Unobservability***: Memastikan bahwa pengguna dapat menggunakan sumber daya atau layanan tanpa orang lain, terutama pihak ketiga, dapat mengamati bahwa sumber daya atau layanan sedang digunakan. *Unobservability* mensyaratkan bahwa pengguna dan / atau subjek tidak dapat menentukan apakah suatu operasi sedang dilakukan. Alokasi informasi yang berdampak *unobservability* mensyaratkan bahwa fungsi keamanan menyediakan mekanisme khusus untuk menghindari konsentrasi informasi terkait privasi dalam sistem. *Unobservability* tanpa meminta informasi mensyaratkan bahwa fungsi keamanan tidak mencoba untuk mendapatkan informasi terkait privasi yang dapat digunakan untuk mengkompromikan

*unobservability*. Pengamatan pengguna yang diotorisasi membutuhkan fungsi keamanan untuk memberikan satu atau lebih pengguna yang berwenang kemampuan untuk mengamati penggunaan sumber daya dan atau layanan.

Perlu diperjelas bahwa kriteria spesifikasi diatas lebih berkaitan dengan privasi individu sehubungan dengan penggunaan sumber daya komputer individu tersebut, daripada privasi informasi pribadi mengenai individu tersebut.

**c) Privasi dan Pengawasan Data**



Tuntutan keamanan dan kontra terorisme tanah air telah memberlakukan ancaman baru terhadap privasi pribadi. Lembaga penegak hukum dan intelijen menjadi semakin agresif dalam menggunakan teknik pengawasan data untuk memenuhi tugas mereka. Selain itu, organisasi swasta mengeksploitasi sejumlah tren untuk

meningkatkan kemampuan mereka dalam membangun profil individu secara terperinci, termasuk penyebaran internet, peningkatan metode pembayaran elektronik, penggunaan komunikasi telepon seluler yang hampir universal, perhitungan di mana-mana, dan lain sebagainya.

Menurut Stallings dan Brown (Stallings and Brown, 2015), pendekatan dengan menggunakan kebijakan dan teknis diperlukan untuk melindungi privasi ketika pemerintah dan organisasi non-pemerintah berusaha untuk belajar sebanyak mungkin tentang individu. Dalam hal pendekatan teknis, persyaratan untuk perlindungan privasi untuk sistem informasi dapat diatasi dalam konteks keamanan basis data. Artinya, pendekatan yang sesuai untuk perlindungan privasi melibatkan sarana teknis yang telah dikembangkan untuk keamanan basis data.



Perangkat yang digunakan dalam keperluan keamanan privasi adalah perangkat yang tahan terhadap gangguan, dilindungi kriptografi

yang diselingi antara *database* dan akses antarmuka, dan dipadukan dengan *firewall* atau perangkat pencegahan intrusi. Perangkat harus mengimplementasikan fungsi perlindungan privasi, termasuk memverifikasi izin akses dan informasi-informasi penting pengguna dan membuat *history* sebagai audit. Beberapa fungsi spesifik dari perangkat yang digunakan dalam pengawasan privasi menurut Stalling dan Brown (Stallings and Brown, 2015) adalah sebagai berikut:

- ***Data transformation***: Fungsi ini berfungsi untuk menyandikan atau mengenkripsi bagian dari data sebagai upaya untuk menjaga privasi tetapi tetap menjaga efektifitas analisa data. Contoh fungsi analisis data tersebut adalah deteksi pola aktivitas teroris.



- ***Anonymization***: Fungsi ini bertujuan untuk menghapus informasi-informasi personal yang dihasilkan dari *query database*, seperti nama belakang dan nomor telepon. Meskipun

demikian tetap dibuat semacam proses identifikasi unik yang dianonimkan sehingga analis dapat mendeteksi koneksi setiap *request* dari pengguna.

- ***Selective revelation***: *Selective relation* adalah metode untuk meminimalkan pemaparan informasi individu namun tetap memungkinkan proses analisis terhadap interaksi data tetap dapat berjalan dengan baik. Fungsi ini pada awalnya mengungkapkan informasi kepada analis hanya dalam bentuk yang informasi yang telah disanitasi, yaitu dalam hal statistik dan kategori yang tidak mengungkapkan (baik secara langsung atau tidak langsung) informasi pribadi siapa pun. Jika analis menemukan sesuatu yang mengkhawatirkan, ia dapat menindaklanjuti dengan meminta izin untuk mendapatkan informasi yang lebih detail. Izin ini akan diberikan jika informasi awal memberikan alasan yang cukup untuk

memungkinkan pengungkapan lebih banyak informasi, di bawah pedoman hukum dan kebijakan yang sesuai.

- ***Immutable audit***: Fungsi dari metode ini adalah untuk mengidentifikasi kemana data akan di transmisikan dan siapa yang telah melihat data. Fungsi audit secara otomatis dan permanen mencatat semua akses data, dengan perlindungan kuat terhadap penghapusan, modifikasi, dan penggunaan yang tidak sah.

- ***Associative memory***: Associative memory merupakan modul perangkat lunak yang dapat mengenali pola dan membuat koneksi antara bagian-bagian data yang mungkin terlewatkan oleh pengguna atau tidak diketahui keberadaannya. Dengan metode ini, dapat ditemukan hubungan antara titik data yang ditemukan dalam sejumlah besar data.

**d) Perbedaan Autentikasi dan Privasi**

Sampai saat ini, secara umum diyakini bahwa enkripsi dari sebuah informasi dianggap sudah cukup untuk melindungi keasliannya. Argumen yang mendukung pernyataan tersebut adalah bahwa sebuah *chipertext* dihasilkan setelah dekripsi dalam informasi. Dengan demikian dapat dipastikan pesan berasal dari seseorang yang mengetahui kunci rahasia, yang menjamin keaslian pesan dan pengirim. Akibatnya, jika *attacker* ingin mengubah pesan yang dienkripsi atau ingin mengirim pesan palsu, ia harus mengetahui kunci rahasia yang digunakan untuk memecah algoritma enkripsi yang digunakan.

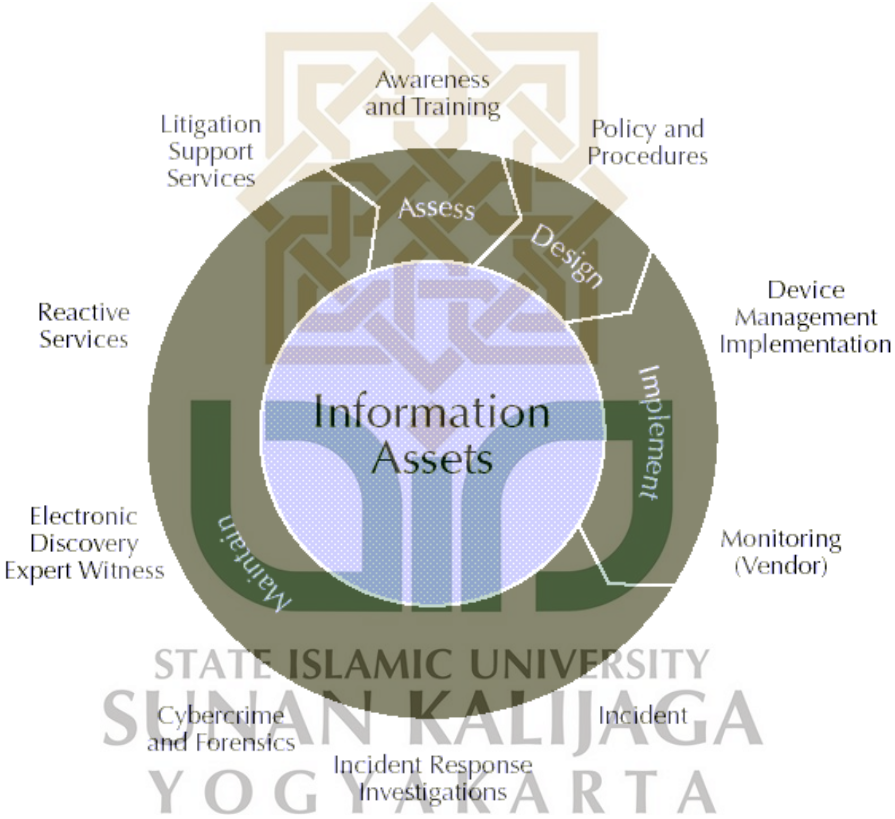
**2. Keamanan Informasi**

Ketika berbicara mengenai *security*, yang muncul dalam pikiran kebanyakan orang adalah *network security* atau keamanan jaringan. Padahal sesungguhnya yang ingin diamankan adalah informasi. Maksudnya keamanan-kemanan jaringan dan lain sebagainya itu ada untuk memastikan bahwa informasi tersebut dikirimkan melalui jaringan adalah benar, tetapi tetap yang ingin diamankan seseungguhnya adalah informasinya.

Keamanan informasi adalah penjagaan informasi dari seluruh ancaman yang mungkin terjadi dalam upaya untuk memastikan atau menjamin kelangsungan bisnis (*business continuity*), meminimalisasi risiko bisnis (*reduce business risk*) dan memaksimalkan atau mempercepat pengembalian investasi dan peluang (Sarno and Iffano, 2009). Menurut Rahardjo (Rahardjo, 2017), rendahnya kesadaran masyarakat atas masalah keamanan informasi merupakan salah

penyebab utama munculnya masalah keamanan. Bahkan para praktisi juga masih sering melakukan kebiasaan buruk, seperti contohnya berbagi *password* admin. Masalah keamanan informasi yang biasanya berupa data teknis harus diterjemahkan ke angka finansial agar dapat dimengerti oleh pihak pimpinan.

**a) *Security Life-Cycle***






Gambar 1: Security Life-Cycle

Menurut Rahardjo (Rahardjo, 2017), banyak orang yang beranggapan bahwa masalah keamanan informasi masih dapat dipecahkan dengan membeli produk keamanan, misalnya *firewall*, anti-virus, dan lainnya. Kemanan informasi sebetulnya berupa sebuah siklus sebagaimana ditampilkan pada gambar diatas. Sesuatu yang akan diamankan merupakan sebuah aset. Untuk itu, hal pertama dalam pengamanan adalah menentukan aset yang ingin dilindungi. Apa saja yang dianggap sebagai aset harus ditentukan bersama dengan pemilik dari sistem. Hal tersebut disebab karena mereka yang lebih mengetahui mana aset dan mana yang bukan aset. Proses ini disebut assesment dan dapat dilakukan dengan melalui *training* atau *awareness* terhadap pihak-pihak terkait. Pada umumnya pemilik aplikasi memahami mana asetnya tetapi pihak operasional (orang-orang IT yang diberi tugas untuk mengamankan sistem) tidak tahu. Setelah mengetahui aset yang ingin diamankan, aset tersebut harus diberi harga (*value*). Pengamanan

nantinya akan disesuai dengan dengan nilai dari aset tersebut. Akan sulit kita melakukan investasi pengamanan yang biasanya lebih mahal dari nilai asetnya. Sebagai contoh, jika terdapat aset berupa sebuah komputer yang harganya Rp. 5.000.000,- maka akan sulit untuk menerima biaya pengamanan yang harganya Rp. 100.000.000,- (lebih mahal). Biaya pengamanan harus lebih murah daripada nilai asetnya, itu rumusnya. Jika biaya pengamanan lebih mahal, mungkin lebih baik membeli barang sejenis saja sebagai duplikat dari aset yang telah hilang atau rusak. Untuk hal-hal yang terkait dengan teknologi informasi, pendaftaran aset-aset ini tidak mudah karena ada hal-hal yang tidak terlihat secara kasat mata. Aset ini dapat kita bagi menjadi tiga jenis item, yaitu *hardware*, *software*, dan data.



Aset apabila berbentuk perangkat keras atau *hardware* akan lebih mudah didata karena dapat dilihat oleh mata, namun ada beberapa hal yang membuatnya menjadi sulit. Salah satu hal yang

menjadi faktor mengapa menjadi sulit adalah mengenai penentuan nilai dari aset tersebut. Harga komputer cenderung jatuh dengan cepat. Berapa depresiasi dari sebuah server? Contoh-contoh aset perangkat keras antara lain komputer, router, perangkat jaringan, *disk*, dan seterusnya. Apakah notebook termasuk aset atau barang habis? Bagaimana dengan USB *flashdisk*? Apakah itu termasuk aset juga? Perntanyaan-pertanyaan seperti diatas sangat mempengaruhi perubahan nilai sebuah aset perangkat keras. Beberapa kejadian terkait dengan kesulitan mendata perangkat keras antara lain tidak diketahuinya pemilik dari perangkat tersebut. *database* perangkat keras sering tidak tersedia. Sebagai contoh, sering kali tidak diketahui *harddisk* dan lokasi *server* yang menggunakan *disk* tersebut.



Apabila dalam mendata perangkat keras sudah tidak mudah, maka pendataan perangkat lunak akan lebih susah lagi. Perangkat lunak tidak terlihat secara kasat mata sehingga pendataannya harus

melalui pemilik layanan atau pemilik aplikasi. Sebagai contoh, sebuah layanan *online* memiliki aplikasi di server web dan juga basis data di server *database*. Aplikasi-aplikasi tersebut berbentuk perangkat lunak yang tentunya juga memiliki harga.

Penentuan nilai dari perangkat lunak atau *software* cukup rumit. Beberapa aplikasi yang didapatkan dari membeli atau dibangun oleh vendor, dapat digunakan harga pembelian sebagai nominal nilai dari aplikasi. Untuk aplikasi yang dikembangkan sendiri ada beberapa orang yang menggunakan jumlah waktu pengembang (dalam man days) yang kemudian dikalikan dengan honor (gaji) orang tersebut. Itulah bagaimana cara menghitung harga dari aplikasi tersebut. Yang masih menjadi pertanyaan adalah bagaimana cara untuk menentukan harga dari *software* yang bersifat *open source* atau gratis, dan menentukan depresiasi nilai perangkat lunak.

Setelah dibahas mengenai perangkat keras dan perangkat lunak, pembahasan selanjutnya dari aset teknologi informasi adalah data. Jika dalam mendata aset *hardware* dan *software* sudah sulit, pendataan data lebih sulit lagi. Penentuan harga atau nilai dari data bisa dikatakan sebagai proses yang sulit. Misalnya jika diberikan sebuah pertanyaan: berapa harga data transkrip mahasiswa? Bagi mahasiswa yang memiliki transkrip tersebut tentunya data mengenai transkrip sangat berarti karena itu adalah perjuangan selama sekian tahun untuk mendapatkan transkrip tersebut. Namun bagi orang lain yang tidak berkaitan data transkrip mahasiswa mungkin tidak akan ada artinya, tidak laku untuk dijual.



Setelah mengetahui aset apa saja yang akan dilindungi hal pertama yang harus dilakukan adalah mendesain keamanan seperti apa yang akan diberikan terhadap aset-aset tersebut. Hal ini dapat dikerjakan dengan mengembangkan kebijakan dan prosedur.

Kebanyakan orang mengabaikan desain keamanan dan langsung melakukan implementasi, padahal tanpa adanya desain keamanan akan menyulitkan proses yang akan dilalui kedepannya. Sebagai contoh, misalnya terdapat aset burupa data transkrip mahasiswa, siapa saja yang mendapatkan akses terhadap data tersebut dan akses seperti apa yang diberikan? Ketentuan-ketentuan tersebut dapat dituangkan dalam kebijakan. Bila tidak ada kebijakan seperti ini, perangkat pengamanan yang ada (*authorization* dan *access control*) akan sulit diterapkan. *Rules* apa yang akan dipakai? Banyak kejadian sistem pengamanan diterapkan dengan salah karena tidak memiliki desain yang benar. Berbekal pola desain keamanan yang berupa kebijakan dan prosedur, ketentuan-ketentuan yang tertera didalamnya dapat diimplementasikan secara teknis melalui perangkat pengamanan. Dalam penerapan pengamanan dapat dilakukan dengan meminta bantuan terhadap vendor-vendor yang berkaitan dengan pengembangan sistem. Hal ini

memungkinkan karena vendor dapat lebih mudah memahami kebutuhan dari suatu organisasi berbekal dari naskah kebijakan dan prosedur. Vendor akan kesulitan mengaplikasikan pola keamanan apabila dari internal organisasi tidak memetakan kebijakan seperti apa yang diperlukan karena requirements terhadap suatu proses bisnis hanya diketahui oleh pihak internal. Meskipun pengamanan sudah diterapkan, insiden keamanan mungkin saja masih dapat terjadi. Ketika insiden ini terjadi, yang harus dilakukan adalah melakukan investigasi. Apakah insiden yang tersebut benar-benar insiden ataukah kejadian biasa saja? Jika memang itu adalah insiden, maka akan diproses lebih lanjut sesuai dengan kebijakan dan prosedur yang sudah dibuat sebelumnya. Hal demikian adalah fungsi lebih lanjut kenapa diperlukan kebijakan dan prosedur yang jelas yang sesuai dengan proses bisnis didalam sebuah organisasi.

Proses-proses terjebut apabila berjalan terus menerus akan menjadi kebiasaan dan menjadi sebuah siklus. Hal yang menjadi pembeda adalah organisasi akan lebih memahami kebutuhan akan keamanan dari organisasinya sehingga akan dapat menekan biaya pengeluaran. Kebanyakan orang masih menganggap bahawa *security* adalah sebuah produk sehingga mereka lebih berfokus kepada pembelian produk dan perangkat-perangkat keamanan tanpa memperhatikan faktor lain, misalnya aset-aset apa saja yang harus dilindungi tidak definisikan.

**b) Keamanan Elemen Sistem**



Orang awam memahami bahwa jika ada sebuah pembicaraan mengenai *security*, yang ada didalam pikiran kebanyakan orang adalah mengenai network *security* (keamanan jaringan). Jika ditelisik lebih dalam keamanan informasi lebih dari itu, karena teknologi sistem informasi tidak sekedar jaringan saja sehingga dibutuhkan sentuhan

keamanan untuk setiap *layer* yang berperan dalam membangun sebuah sistem.

Menurut Rahardjo (Rahardjo, 2017), sebuah sistem yang berbasis teknologi informasi memiliki beberapa elemen (komponen), diantaranya adalah seabgai berikut:

- *host, server, workstation*
- jaringan (beserta perangkatnya), dan
- aplikasi (*software*, *database*).

Setiap elemen diatas memeiliki fungsi yang berbeda-beda dan dikarenakan memiliki perbedaan fungsi dan karakteristik akan membedakan kebutuhan keamanan yang harus diberikan kepada setiap elemen tersebut.

*Computer security* merupakan kemanan yang berkaitan dengan kemanan komputer, baik itu server, workstation, notebook dan sejenisnya. Dalam istilah lain *computer security* juga disebut sebagai *host security*. Permasalahan yang muncul pada keamanan komputer biasanya berhubungan dengan sistem operasi yang digunakan, *patch* yang sudah dipasang, konfigurasi yang digunakan, serta keberadaan aplikasi yang ada. Sistem operasi yang digunakan jika telah *out-of-date* atau kadalwarsa biasanya memiliki beberapa kerentanan yang informasi mengenai kerentanannya telah tersebar sehingga diperlukan pembaruan. Meski demikian pembaruan sistem operasi tidak serta merta dapat dilakukan dikarenakan berkaitan dengan kebutuhan aplikasi yang terinstall didalam perangakt tersebut. Sehingga masih banyak orang yang mempertahankan sistem operasi kadalwarsa demi mendukung sistem yang telah lama dibangun.

Selain pengamanan terkait perangkat komputer yang digunakan, hal selanjutnya yang perlu diperjelas berdasarkan elemen-elemen yang sudah disebutkan sebelumnya adalah mengenai *network-security*. Banyak sekali celah keamanan yang membuat terjadinya kebocoran data timbul karena keamanan pada jaringan yang lemah. Transmisi data dari suatu perangkat ke perangkat lain dicegat dan dapat disadap oleh orang yang tidak berhak. Bila hal demikian terjadi, selain data bocor juga dapat dimungkinkan terjadi modifikasi data, atau hal yang lebih buruk adalah hilangnya paket data yang ditransmisikan. Serangan lain terhadap jaringan yang sangat terkenal adalah DDOS atau *Distributed Denial of Services*. *Denial of Services* bekerja dengan membanjiri jaringan dengan paket-paket yang sangat besar sehingga dapat membuat sistem lumpuh dan tidak dapat menyediakan layanan.

Meskipun perangkat komputer yang digunakan sudah baik dan didukung jaringan yang baik pula, bukan berarti permasalahan selesai disini. Kemananan akan aplikasi yang digunakan atau dibangun merupakan *layer* utama yang harus diberikan sentuhan keamanan. Aplikasi menjadi *front-liner* yang dapat berkomunikasi secara langsung dengan pengguna, sehingga diperlukan upaya agar dalam komunikasi yang terjalin tersebut dapat mengantisipasi adanya pengguna-pengguna yang nakal dan berniat buruk terhadap sistem yang disediakan. Serangan yang mungkin dapat dilakukan oleh pengguna terhadap aplikasi misalnya adalah *SQL Injection*, *File Injections* dan lain sebagainya. *SQL Injection* (SQLi) menurut Stallings dan Brown (Stallings and Brown, 2015), merupakan *network-based* yang berbahaya dan sering terjadi. Stallings menjelaskan bahwa SQLi didesain untuk mengirimkan *query-query* tambahan ke *database* server dengan tujuan untuk menemukan informasi-informasi tertentu.

Terkait dengan *application security* yang berkaitan erat terhadapnya adalah keamanan sistem *database*. Penanganan masalah keamanan *database* mirip dengan penanganan keamanan jaringan (bukan aplikasi).

**c) Prinsip-prinsip Keamanan Informasi**

Ada beberapa prinsip yang harus diperhatikan dalam melakukan keamanan terhadap informasi. Menurut Rahardjo (Rahardjo, 2017), dan Stalling (Stallings and Brown, 2015) prinsip-prinsip keamanan tersebut diantaranya adalah sebagai berikut:

1. *Confidentiality*



*Confidentiality* atau kerahasiaan merupakan aspek yang biasa dipahami tentang keamanan. Pada aspek *confidentiality* mengharuskan data hanya dapat diakses atau dilihat oleh orang yang memiliki hak atas data tersebut. Biasanya aspek ini yang paling mudah dipahami

oleh sebagian besar orang. Apabila terkait dengan data pribadi, aspek *confidentiality* juga dikenal dengan istilah *Privacy*. Serangan-serangan terhadap aspek *confidentiality* biasanya berupa penyadapan terhadap data (melalui jaringan), memasang *keylogger* untuk menyadap sesuatu yang diketikkan di *keyboard*, dan pencurian secara fisik terhadap mesin atau disk yang digunakan untuk menyimpan data. Perlindungan terhadap aspek *confidentiality* dapat dilakukan dengan menggunakan kriptografi, dan membatasi hak akses yang diberikan terhadap setiap pengguna.

2. *Integrity*



Aspek *integrity* berhubungan dengan integritas data. Aspek ini menyatakan bahwa data tidak boleh berubah tanpa ijin dari yang memiliki hak terhadap data tersebut. Sebagai contoh, jika terdapat data transaksi uang dari nomor rekening 1234 ke nomor 5678 dengan nominal uang yang ditransfer sebesar X, maka data tersebut tidak

boleh sembarangan diubah. Apabila terjadi perubahan data transaksi oleh sembarang orang akan menimbulkan kerugian terhadap berbagai macam pihak.

Serangan yang biasanya muncul terhadap aspek integritas data dapat dilakukan dengan teknik MITM atau *Man In The Midle Attack*, yaitu menangkap data di tengah jalan kemudian memodifikasinya dan meneruskannya ke tujuan. Data yang kemudian sampai di tujuan (misal aplikasi di web server) tidak tahu bahwa data sudah dimodifikasi di tengah jalan.

Perlindungan untuk aspek *integrity* dapat dilakukan dengan menggunakan *message authentication code*. Dengan menggunakan *message authentication code* aplikasi tujuan dapat memastikan apakah data sudah dilakukan modifikasi ditengah jalan atua masih orisinil, dan dapat memastikan bahwa informasi yang dikirim telah sesuai isinya dan dikirimkan oleh pengirim yang sesuai pula.

*3. Availability*

Ketergantungan terhadap sistem berbasis teknologi informasi menyebabkan sistem (beserta datanya) harus dapat diakses kapan saja sewaktu diperlukan. Jika sistem tidak tersedia, tidak dapat diakses, *not available*, maka dapat menimbulkan masalah yang menimbulkan kerugian finansial atau bahkan nyawa. Sebagai contoh misalnya sebuah sistem yang dimiliki oleh Bank X mendapatkan serangan yang menyebabkan sistem tersebut lumpuh dan tidak dapat diakses, maka Bank X tidak dapat memberikan layanan untuk nasabah dapat melakukan transaksi. Bank X mengalami kerugian karena selama sistem yang dimilikinnya tersebut lumpuh tidak dapat memberikan layanan dan mendapatkan keuntungan dari layanannya tersebut. Disisi lain para pelaku ekonomi yang merupakan nasabah Bank X juga dirugikan karena proses bisnisnya tidak dapat berjalan karena bergantung terhadap sistem yang dimiliki oleh Bank X. Itulah

sebabnya aspek *availability* menjadi bagian dari aspek keamanan yang sangat penting.

Serangan terhadap aspek *availability* biasanya dilakukan dengan tujuan untuk melumpuhkan layanan atau dapat dikatakan membuat layanan menjadi sangat lambat sehingga sama saja dengan tidak berfungsi sebagaimana mestinya. Salah satu serangannya disebut dengan *Denial of Service* (DOS). Serangan DOS melumpuhkan sistem dengan mengirimkan banyak paket berukuran besar melalui jalur yang digunakan oleh sistem tersebut, sehingga transfer data yang dilakukan oleh sistem menjadi terganggu dan sangat lambat. Sebagai analogi sederhana, serangan DOS dapat digambarkan seperti lalulintas dijalan raya. Semakin padat pengguna jalan, trafik lalulintas akan semakin melambat dan perjalanan akan semakin lama.



Perlindungan yang dapat dilakukan untuk menjaga aspek *availibility* tetap tersedia adalah dengan menyediakan redundansi.

Sebagai contoh, jaringan komputer dapat menggunakan layanan dari dua penyedia jasa yang berbeda. Jika salah satu penyedia jasa terkena serangan atau lumpuh karena ada kendala, jaringan lainnya akan tetap dapat memberikan support agar ketersediaan layanan dapat terjaga.

Aspek-aspek diatas dikenal sebagai CIA yang merupakan singkatan dari *Confidentiality*, *Integrity*, dan *Availability*. Selain ketiga aspek diatas, ada beberapa aspek lain yang mendukung keamanan sebuah sistem, diantaranya adalah sebagai berikut:

1. *Non-repudiation*

Aspek ini disebut juga aspek nir-sangkal dan digunakan untuk membuat para pelaku tidak dapat menyangkal apabila telah melakukan sesuatu. Aspek ini biasanya lekat di dalam sistem yang berhubungan dengan transaksi. Contoh penggunaan aspek *non-repudiation* adalah dalam sistem lelang elektronik. Implementasi dari aspek ini dapat

dilakukan dengan menggunakan *message authentication code* (dengan menggunakan fungsi *hash*) dan pencatatan (*logging*).

2. *Authentication*

Proses *Authentication* menurut Rahardjo (Rahardjo, 2017), digunakan untuk membuktikan klaim bahwa seseorang itu adalah benar-benar yang diklaim. Proses pembuktian seseorang ini lebih mudah dilakukan di dunia nyata namun akan sulit bila dibandingkan dengan di dunia maya. Proses *authentication* ini dapat dilakukan dengan bantuan hal lain, yang sering disebut "faktor". (Sehingga ada istilah *two-factor authentication*.) Faktor-faktur tersebut adalah sebagai berikut:



- Sesuatu yang diketahui. Contoh dari faktor ini adalah nama, userid, *password*, dan PIN.

- Sesuatu yang dimiliki. Contoh dari faktor ini adalah kartu, kunci, dan token.
- Sesuatu yang menjadi bagian dari fisik pengguna. Contoh dari faktor ini adalah sidik jari, retina mata, dan biometric lainnya.

Selain faktor-faktor diatas, beberapa ahli menambahkan faktor-faktor lain sebagai berikut:

- orang tersebut berada di tempat tertentu. (*Proximity*)
- *authentication* dengan menggunakan bantuan pihak lain, pihak ketiga yang terpercaya (*trusted third party*).



3. *Authorization*

Pada aspek *authentication*, dapat diketahui siapa pengguna dan peran dari pengguna tersebut. Selanjutnya hak akses akan diberikan

kepada pengguna sesuai dengan peran yang dimilikinya. Inilah aspek *authorization*.

**3. Algoritma Kriptografi**

Menurut Rahardjo (Rahardjo, 2017), terdapat dua cara yag dapat digunakan untuk mengamankan data, yaitu menyembunyikan data atau menyandikan data. Cara pertama menggunakan steganografi, sementara cara kedua menggunakan kriptografi. Steganografi (*steganography*) merupakan ilmu untuk menyembunyikan pesan sehingga tidak terlihat dengan mudah. Mekanisme penyembunyian yang dilakukan dengan menggunakan media lain, dalam praktiknya bisa dalam sebuah gambar, audio atau meida lainnya. Sejarah mengatakan teknik penyembunyian pesan dapat dilakukan dengan menggunakan meja yang dilapisi lilin, teknik ini dilakukan pada jaman perang antara Yunani dan Persia. Pada masa kini steganografi digunakan sebagai bagian dari *Digital Rights Management* (DRM),

sebagai contoh misalnya dengan menyisipkan informasi mengenai HaKI dari produk digital (musik, *ebook*, foto, dan sejenisnya).

Berbeda dengan steganografi, kriptografi tidak berorientasi terhadap proses menyembunyikan pesan akan tetapi kriptografi mengubah pesan sehingga sulit dibaca isi pesan aslinya. Biasanya pesan diubah dengan cara transposisi (mengubah letak dari huruf) dan substitusi (mengganti huruf/kata dengan huruf/kata lainnya). Pesan yang sudah diubah terlihat seperti sampah, tetapi tetap terlihat oleh penyerang atau orang yang tidak berhak.

Teknik kriptografi sudah digunakan sejak jaman dulu pada masa Romawi kuno. Pada jaman tersebut, Julius Caesar menggunakan teknik kriptografi yang kemudian dikenal dengan *Caesar chiper*. Julius Caesar menggunakan teknik kriptografi tersebut untuk mengirimkan pesan secara rahasia, meskipun dalam ukurannya sudah tidak relevan bila digunakan pada masa kini (Kromodimoeljo, 2009). Selain Julius

Caesar kriptografi juga digunakan pada masa perang dunia dan banyak manuskrip yang menceritakan penggunaan kriptografi pada jaman dulu. Sejarah kriptografi penuh dengan intrik sehingga banyak orang yang menganggap kriptografi penuh dengan misteri.

Kriptografi adalah ilmu mengenai teknik enkripsi dimana data diacak menggunakan suatu kunci enkripsi menjadi sesuatu yang sulit dibaca oleh seseorang yang tidak memiliki kunci dekripsinya. Dekripsi menggunakan kunci yang sama seperti yang digunakan pada proses enkripsi untuk mendapatkan kembali data asli.

Proses enkripsi dilakukan dengan menggunakan suatu algoritma dengan menyertakan parameter-parameter pendukung. Algoritma enkripsi tidak dirahasiakan, bahkan penggunaan kriptografi dengan merahasiakan algoritma yang digunakan dianggap tidak baik. Rahasia terletak pada parameter-parameter yang disertakan dalam

proses enkripsi dengan kata lain kunci dalam proses enkripsi dan dekripsi terletak pada parameter-parameter yang digunakan.

Menurut Kromodimoeljo (Kromodimoeljo, 2009), dalam kriptografi klasik biasanya menggunakan enkripsi simetris dimana kunci enkripsi sama dengan kunci yang digunakan untuk melakukan dekripsi. Sementara jika kunci enkripsi dan dekripsinya berbeda maka dikatakan sebagai kriptografi asimetris. Enkripsi, dekripsi dan proses pembuatan kunci dalam kriptografi asimetris memerlukan komputasi yang lebih intensif dibandingkan pada kriptografi simetris. Hal tersebut dikarenakan pada kriptografi asimetris dibutuhkan bilangan-bilangan yang sangat besar.



Fungsi utama kriptografi menurut Kurniawan (Kurniawan, 2004) adalah untuk menjaga agar baik *plaintext* maupun kunci ataupun keduanya tetap terjaga kerahasiaanya dari penyadap (disebut juga sebagai lawan, penyerang, pencegat, penyelundup pesan, musuh,

*attacker*, dan sebagainya). Pencegat pesan rahasia diasumsikan mempunyai akses yang lengkap kedalam saluran komunikasi antara pengirim dan penerima. Hal semacam ini sangat mudah terjadi pada jalur internet dan saluran telepon. Kriptografi juga digunakan untuk identifikasi pengirim pesan dengan tanda tangan digital dan keaslian pesan dengan sidik jadi digital (*fingerprint*).

**a) Konsep Acak**

Dalam kriptografi yang dimaksud dengan konsep acak atau *randomness* adalah sifat bebas dari kecenderungan sehingga tidak mudah untuk diterka. Dari segi matematika, jika suatu variabel dianggap bersifat acak, maka teori probabilitas dapat digunakan untuk memprediksi kelakuan dari variabel tersebut, antara lain variabel akan memenuhi beberapa kriteria statistik. Metode statistika dapat digunakan, berdasarkan apa yang sudah terjadi, untuk menilai apakah variabel memenuhi kriteria statistik untuk variabel acak. Akan tetapi

jika kriteria statistik terpenuhi, belum tentu variabel benar acak, karena sesuatu yang deterministik seperti *pseudo-random number generator* dapat memenuhi kriteria statistik untuk variabel acak. Jadi kriteria statistik bukan merupakan definisi untuk variabel acak (Kromodimoeljo, 2009).

Sifat acak merupakan sebuah sifat yang tidak dapat didefinisikan secara matematis. Ababila sebuah variabel dapat didefinisikan secara matematis maka dapat dikatakan bahwa variabel tersebut tidak bersifat acak karena sifat acak tidak memiliki ukuran sehingga tidak dapat diukur dan diprediksi.



Dalam penerapan kriptografi, penerapan konsep acak biasanya digunakan pada penentuan variabel pendukung seperti *salt*. *Salt* biasanya di *generate* secara *random* menggunakan algoritma pada bahasa pemrograman tertentu untuk dapat meningkatkan keamanan algoritma kriptografi.

Sifat acak juga dikaitkan dengan tidak adanya korelasi atau dalam beberapa referensi lain dikatakan korelasi mendekati nol. Dalam dunia kriptografi tidak diinginkan adanya hubungan atau korelasi antara naskah asli dengan naskah acak atau kunci dengan naskah acak. Hal ini dimaksudkan untuk mempersulit analisa seperti analisa frekuensi (*frequency analysis*) atau analisa yang lebih canggih seperti *linier cryptanalysis* atau *differential cryptanalysis*. Meskipun tidak acak secara mutlak, sesuatu yang bersifat *pseudo-random* berguna dan digunakan dalam dunia kriptografi namun harus tetap dikombinasikan dengan sesuatu yang benar-benar acak. Seabgai contoh, *pseudo-random number generator* memungkinkan untuk dikombinasikan dengan sumber entropi yang benar-benar acak sebagai *seed* untuk mendapatkan sesuatu yang praktis bersifat *random number generator*.

**b) *Cryptanalysis***

Menurut Kromodimoeljo (Kromodimoeljo, 2009), *cryptanalysis* merupakan teknik yang digunakan untuk memecahkan enkripsi. Ada tiga teknik yang biasanya digunakan untuk kriptografi klasik, yaitu:

1. *known plaintext attacker*
2. analisa statistik, dan
3. *bruteforce search*

Selain digunakan secara terpisah, kombinasi-kombinasi dari teknik-teknik diatas juga sering digunakan oleh para *attacker* untuk memecahkan sebuah enkripsi. Biasanya minimal pemecah memiliki akses ke naskah acak dan kadang memiliki akses langsung ke naskah aslinya.

Algoritma enkripsi klasik yang baik adalah algoritma yang tahan terhadap serangan menggunakan *known plaintext attack* dan analisa frekuensi. Dengan demikian mau tidak mau pencarian kunci harus dilakukan dengan melakukan bruteforce. Untuk mengantisipasi keamanan kriptografi dari serangan bruteforce kunci enkripsi harus cukup besar akan serangan bruteforce menjadi tidak efektif.

Berbeda dengan algoritma enkripsi klasik atau simetris, algoritma asimetris yang menggunakan *public key* mengandalkan keamanannya pada tingkat komputasi yang tinggi untuk mendapatkan kunci rahasia. Hal tersebut dapat dilakukan dengan menerapkan penguraian bilangan bulat yang besar atau komputasi logaritma diskrit untuk *finite field* yang besar. Dengan demikian pemecahan kunci publik difokuskan pada teknik-tenik untuk mempercepat kedua komputasi tersebut.

**c) *Known plaintext attack***

Menurut Kromodimoeljo (Kromodimoeljo, 2009), *known plaintext attack* adalah teknik yang digunakan untuk menemukan kunci enkripsi berdasarkan pengetahuan dari naskah asli dan naskah acak. Salah satu contoh algoritma kriptografi yang rentan terhadap serangan *known plaintext attack* adalah algoritma Caesar.

Julius Caesar menukarkan setiap huruf pada naskah asli dengan huruf lain dalam naskah acak. Besar atau kecilnya huruf dipertahankan dalam naskah acak (huruf besar ditukar dengan huruf besar dan huruf kecil ditukar dengan huruf kecil). Sementara spasi, titik, koma dan tanda baca lainnya tidak ditukar oleh Julius Caesar. Algoritma Caesar Chiper adalah jenis enkripsi yang termasuk dalam *simple subtitution chiper* dimana setiap huruf dalam naskah asli ditukar dengan huruf lain dalam naskah acak.

*Known plaintext attack* terhadap algoritma Caesar Chiper merupakan contoh *attack* yang bersifat deterministik dimana jika pasangan (atau beberapa pasangan) naskah asli dan naskah acak diketahui maka kunci dapat ditemukan dengan mudah. Meskipun demikian tidak semua *known plaintext attack* bersifat deterministik, *linier cryptanalysis* meskipun menggunakan *known plaintext attack* namun bersifat probabilistik.

Tingkat kesukaran dalam mengaplikasikan *known plaintext attack* sangat bergantung terhadap rumus yang digunakan dalam melakukan enkripsi. Semakin rumit rumus yang digunakan dalam melakukan enkripsi, maka semakin sukar untuk melakukan *known plaintext attack*. Rumus yang bersifat linier masih tergolong sederhana dan dianggap rentan terhadap *known plaintext attack*. Dalam arti yang lain dapat dikatakan bahwa semakin semakin non-linier dan semakin banyak parameter yang digunakan dalam menyusun rumus ketika

melakukan enkripsi, maka akan semakin sulit untuk melakukan kalkulasi kunci berdasarkan rumus.

**d) Analisa Statistik**

Analisa statistik merupakan teknik yang digunakan untuk menemukan *plaintext* dengan mengamati pola-pola yang telah terjadi sebelumnya berdasarkan tingkatan frekuensi dari pola yang ada. Menurut Kromodimoeldjo (Kromodimoeljo, 2009), semua algoritma enkripsi (kecuali *one-time pad*) sebelum adanya *Data Encryption Standard* (DES) merupakan algoritma yang rentan terhadap analisa statistik.



Sebagai contoh dapat dilihat dari bagaimana enkripsi dengan cara *shift transformation* seperti *Caesar cipher* rentan terhadap analisa statistik yang sederhana yaitu analisa frekuensi. Misalnya terdapat rumus enkripsi sebagai berikut:

$$C \equiv P + B \pmod{n}$$

Jika $n$ diketahui dan sepasang $C$ dan $P$ dapat diterka dengan akurat, maka parameter $B$ (kunci) dapat dicari. Setiap proses pencarian nilai $B$ cukup dengan sepasang nilai $C$ dan $P$. Dari sekian banyak kandidat pasangan yang dapat digunakan adalah pasangan yang sesuai dengan statistik frekuensi penggunaannya.

Untuk mencapai keberhasilan dalam melakukan serangan dengan teknik analisa frekuensi, dibutuhkan pengetahuan yang mendalam mengenai statistik penggunaan huruf, naskah acak yang dapat dianalisa harus cukup besar, rumus atau paling tidak jenis enkripsinya telah diketahui. Dengan mengetahui jenis enkripsi, meskipun tidak diketetahui rumus yang digunakan namun dapat ditebak pola yang digunakan pada proses enkripsi tersebut. Penggunaan komputer sangat membantu dalam menggunakan teknik analisa frekuensi, terutama pada analisa frekuensi yang rumit. Semakin

tinggi spesifikasi komputer yang digunakan akan semakin memudahkan proses analisa frekuensi.

Pada umumnya, semakin besar data yang digunakan sebagai dasar, baik untuk probabilitas *a priori* seperti frekuensi penggunaan huruf, maupun yang bersifat *a posteriori* yaitu naskah acak, semakin pasti (tidak tentatif) hasil percobaan kalkulasi. Sebagai contoh, untuk *shift transformation* dengan perbendaharaan 26 huruf dan naskah dalam bahasa Inggris, analisa frekuensi dengan naskah acak sebesar 50 karakter atau lebih akan mendapatkan hasil dengan kepastian mendekati 100 persen, berdasarkan pengetahuan *a priori* mengenai penggunaan huruf dalam bahasa Inggris.



Pada umumnya untuk algoritma kriptografi yang rentan terhadap *known plaintext attack* yang deterministik juga merupakan algoritma kriptografi yang rentan terhadap analisa frekuensi. Hal tersebut sangat mungkin jika data empiris mengenai naskah asli sudah

diketahui. Keterkaitan analisa statistik dan *known plaintext attack* membuat analisa statistik dapat dipandang sebagia *known plaintext attack* yang probabilistik, dimana pasangan naskah asli dan naskah acak diperkirakan berdasarkan data-data empiris.

Analisa frekuensi lebih mudah dilakukan pada *subtitution chiper* dibandingkan diaplikasikan pada *block chiper*. Enigma berhasil dipecahkan oleh pihak sekutu menggunakan analisa frekuensi karena enkripsi yang digunakan Enigma adalah *substitution cipher*, meskipun jenisnya adalah *polyalphabetic* (pertukaran huruf berubah terus). Analisa frekuensi terhadap *polyalphabetic substitution cipher* dapat dikatakan lebih sukar dibandingkan dengan analisa frekuensi terhadap *simple substitution cipher*, meskipun demikian tetap jauh lebih mudah dibandingkan analisa frekuensi pada *block chiper*. Analisa frekuensi merupakan salah satu contoh dari analisa statistik yang sederhana.

**e) *Brute-Force Search***

Teknik *Brute-Force* merupakan teknik yang mengharuskan mencoba setiap kemungkinan kunci untuk mendapatkan kunci yang sebenarnya. Salah satu dari kriteria kunci enkripsi yang baik adalah bahwa dekripsi tanpa kunci hanya dapat dipecahkan dengan *bruteforce attack*. Dalam melakukan *brute-force attack* diperlukan *list* kemungkinan kunci-kunci yang dapat digunakan untuk melakukan serangan. Banyaknya jumlah dugaan kunci tentu sangat berpengaruh terhadap lama waktu serangan dan keberhasilan serangan. Apabila sebuah enkripsi dapat dipecahkan dengan *brute-force attack* hanya dengan jumlah dugaan yang sedikit maka dapat dikatakan enkripsi tersebut sangat rentan terhadap serangan.



Menurut Kromodimoeldjo (Kromodimoeljo, 2009), besarnya jumlah bit dalam kunci enkripsi berpengaruh terhadap jumlah kemungkinan kunci yang harus dicoba dalam *brute force search*.

Untuk kunci sebesar *n* bits, jumlah kemungkinan kunci adalah *2n* dan rerata, kunci akan ditemukan setelah mencoba *2n−1* kemungkinan (setengah dari semua kemungkinan). Sehingga enkripsi rentan terhadap *brute force search* jika *2n* kemungkinan kunci dapat dicoba dalam waktu yang tidak terlalu lama. Selain tergantung pada jumlah kemungkinan kunci yang harus dicoba, waktu yang diperlukan juga tergantung pada kemampuan *hardware* yang digunakan. Semakin tinggi spesifikasi *hardware* yang digunakan untuk melakukan proses *brute-force* maka semakin cepat juga waktu yang diiperlukan untuk mencoba setipa dugaan kunci yang ada. Juga batas waktu yang tidak terlalu lama tergantung pada aplikasi, apakah kurang dari 1 bulan, kurang dari 5 tahun, kurang dari 1000 tahun, atau ada batas waktu lain.



Algoritma kriptografi Caesar cipher, selain dapat dipecahkan dengan analisa frekuensi atau *known plaintext attack*, dapat juga dipecahkan dengan *brute force search*. Hal ini dikarenakan semua

kemungkinan kunci (*b = 1* sampai dengan *b = 25*) dapat dicoba dalam waktu yang tidak terlalu lama.

Besarnya kunci enkripsi sangat menentukan tingkat keberhasilan dalam melakukan *brute-force search*. Meskipun telah tersedia jumlah kemungkinan kunci yang sangat besar, namun jika ternyata kunci enkripsi yang dicari juga berukuran besar akan membutuhkan waktu yang sangat lama untuk melakukan kombinasi-kombinasi dari dugaan kata kunci untuk dapat menemukan kunci aslinya.

**f) Macam-macam Algoritma Kriptografi**



Kriptografi dibedakan menjadi 3 bagian, yaitu kriptografi simetris, kriptografi asimetris, dan fungsi *hash* satu arah. Perbedaan dari ketiga fungsi kriptografi tersebut dapat dilihat pada Tabel 2:

Tabel 2: Perbandingan Fungsi Hash, Kriptografi Simetris dan Kriptografi Asimetris Secara Umum (sumber: https://www.cryptomathic.com/)

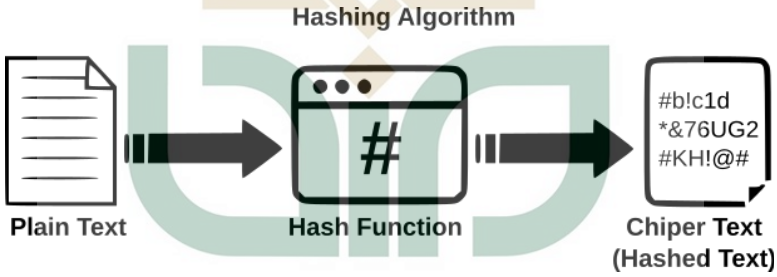
| Feature/ Algorithm | Hash | Symmetric | Assymmetric |
|---|---|---|---|
| Number of Key | 0 | 1 | 2 |
| NIST Recomennded Key Lenght | 256 bits | 128 bits | 2048 bits |
| Commonly Used | SHA | AES | RSA |
| Key Management/ Sharing | N/A | Big Issue | Easy & Secure |
| Effect of Key Compromise | N/A | Lost of Both sender & receiver | Only loss for owner of Assymmetric Key |
| Speed | Fast | Fast | Relative Slow |
| Complexity | Medium | Medium | HIgh |
| Examples | SHA 224,SHA 256,SHA 384, SHA 512 | AES, Blowfish, Serpent,Twofish, 3DES, and RC4 | RSA, DSA, ECC, Diffle Hellman |

Pada tabel diatas telah ditampilkan perbedaan-perbedaan mendasar ketiga macam jenis dari kriptografi. Penjelasan lebih detail mengenai *hash*, *symmetric* dan *assymmetric* kriptografi adalah sebagai berikut:

*1) Hash Function*

Fungsi *hash* satu arah, juga dikenal sebagai rangkuman pesan atau fungsi kompresi yaitu fungsi matematis yang mengambil masukkan panjang variabel dan mengubahnya ke dalam urutan biner dengan panjang yang tetap. Fungsi *hash* satu arah dirancang dengan cara yang sulit untuk membalik proses, yaitu untuk menemukan rangkaian pada nilai tertentu (karena itu dinamakan satu arah). Fungsi *hash* dinilai baik apabila sulit untuk menemukan 2 *string* yang akan menghasilkan nilai *hash* yang sama.

Semua algoritma *hash* modern menghasilkan nilai 128 bit atau lebih. Bahkan, perubahan dalam input *string* juga bisa menyebabkan nilai *hash* berubah secara drastis. Meskipun fungsi *hash* satu arah digunakan paling banyak untuk menghasilkan tanda tangan digital, tetapi masih banyak penerapan lain yaitu seperti penyimpanan *password* dalam *database*.



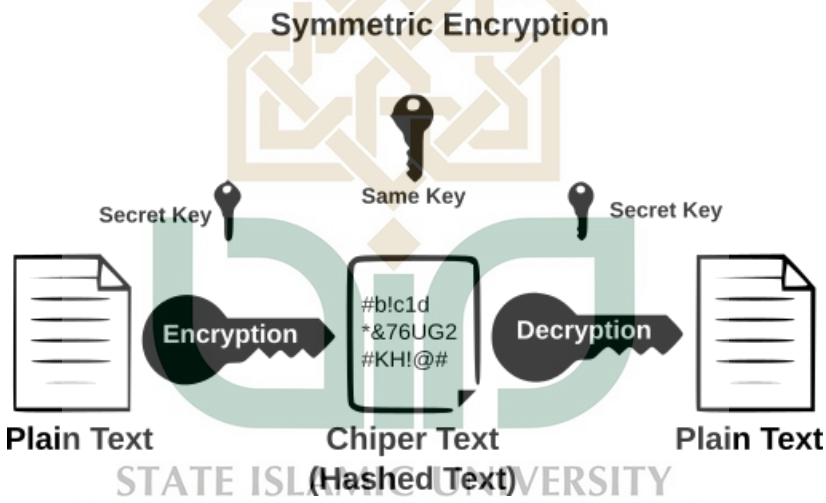

Gambar 2: Cara kerja fungsi Hash (sumber: https://www.ssl2buy.com)



*2) Symmetric Function*

Kriptografi Simetris disebut kriptografi kunci rahasia merupakan jenis kriptografi yang paling intuitif. Ini termasuk penggunaan

kunci rahasia yang dikenal hanya pada pengguna komunikasi yang aman.

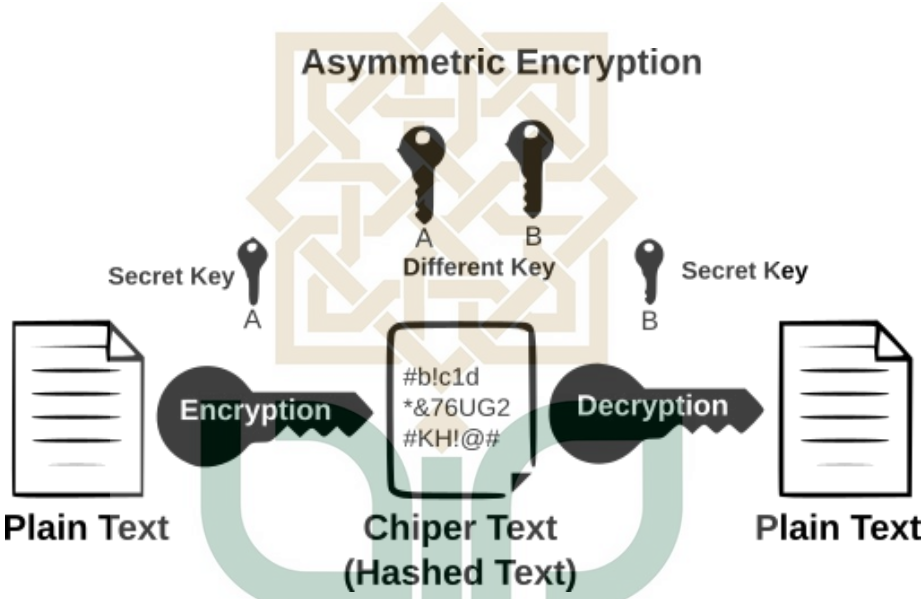





Gambar 3: Cara kerja fungsi Kriptografi Simetris (sumber: https://www.ssl2buy.com)

*3) Assymmetric Function*

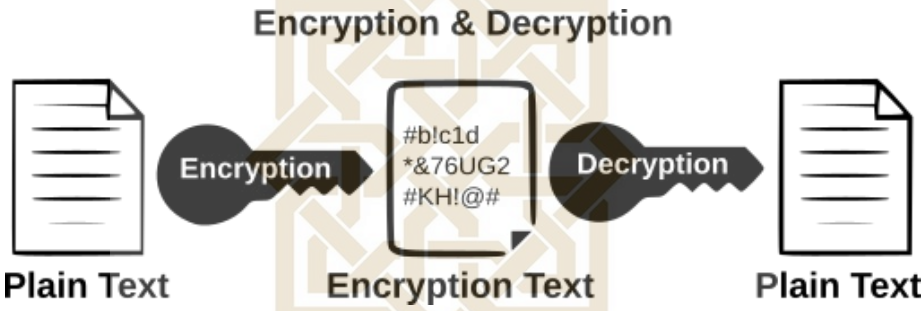
Berbeda dengan kriptografi simetris, kriptografi asimetris menggunakan dua kunci yang berbeda, yaitu kunci publik dan kunci rahasia atau kunci pribadi. Kunci-kunci itu berhubungan secara matematis, tetapi tidak mungkin secara perhitungan untuk menarik kesimpulan satu dengan yang lain. Dengan kunci publik orang dapat mengenkrip pesan tetapi tidak mendekripnya.




Gambar 4: Cara kerja fungsi Kriptografi Asimetris (sumber: https://www.ssl2buy.com)

**4. Dekripsi**

Pada dekripsi, data dilewatkan secara terbalik dari proses enkripsi. Pemetaan kotak-s untuk dekripsi juga kebalikan dari pemetaan untuk proses enkripsi. Jadi, kotak-s haruslah bijektiv, yaitu merupakan pemetaan satu-ke-satu. Urutan *subkey* pada dekripsi juga merupakan kebalikan dari *subkey* pada proses enkripsi (Kurniawan, 2004). Berikut ilustrasi mekanisme enkripsi dan dekripsi:




Gambar 5: Mekanisme Enkripsi dan Dekripsi (sumber: https://www.ssl2buy.com)

**5. *Secure Hashing***



Menurut Kromodimoeldjo (Kromodimoeljo, 2009), *secure hashing* adalah proses pembuatan sidik jari atau digest untuk suatu naskah. Naskah yang dimaksud adalah *plaintext* atau dokumen yang penting sehingga diperlukan proses *hashing* untuk memberikan

keamanan. Sementara digest yang dimaksud adalah *chipertext* yang dihasilkan dari proses *hashing*. *Chipertext* adalah *string* acak yang yang tidak dapat dibaca dan *chipertext* dari proses *hashing* bersifat searah atau dengan kata lain tidak memungkinkan untuk dilakukan proses dekripsi untuk menemukan naskah asli atau *plaintext*-nya.

Meskipun banyak beredar algoritma *secure hashing*, tidak semua dari algoritma tersebut dapat digunakan begitu saja dikarenakan alasan keamanan setiap algoritma berbeda. Menurut Kromodimoeldjo, kriteria yang sudah menjadi standard untuk keamanan algoritma *secure hashing* adalah sebagai berikut:



- *Preimage resistance*

  Untuk suatu nilai *hash* yang sembarang (tidak diketahui asal-usulnya), sangat sukar untuk mencari nilai asli yang mempunyai nilai *hash* tersebut.

- *Second preimage resistance*

  Untuk suatu naskah *m1*, sangat sukar untuk mencari naskah lain *m2* yang mempunyai nilai *hash* yang sama (*hash(m1) = hash(m2)*). Persyaratan seperti ini disebut sebagai *weak collision resistance*.

- *Collision resistance*

  Tingkat kesulitan yang tinggi untuk dapat menemukan dua naskah *m1* dan (*hash(m1) = hash(m2)*). Persyaratan ini kerap disebut juga *strong col-m2*.



Setiap algoritma *secure hashing* biasanya akan membagi naskah asli atau *plaintext* kedalam beberapa blok, dimana setiap blok biasanya berukuran antara 512 sampai dengan 1024 bit. Kemudian untuk menyesuaikan besaran dengan setiap blok, naskah akan diberi

*padding* meskipun besarnya merupakan kelipatan dari besarnya blok dan *padding* diakhiri dengan nilai dari ukuran naskah.

Algoritma *secure hashing* memiliki dua tahapan untuk dapat melakukan proses *hashing* naskah asli ke dalam bentuk *chipertext*. Tahapan-tahapan tersebut adalah tahap *preprocessing* dan *hashing*. Tahap *preprocessing* merupakan tahap yang dilakukan untuk menata naskah sebelum nantinya dilakukan proses *hashing*. Tahap *preprocessing* pada dasarnya terdiri dari proses pemberian *padding* dan parameter *setup*. Setelah tahap pra pemrosesan selesai dilakukan tahap *hashing* baru dapat dilakukan.



Keamanan algoritma *secure hashing* sangat bergantung terhadap proses *hashing* atau kompresi yang dilakukan pada setiap algoritma. Selain 3 kriteria keamanan yang sudah disebutkan sebelumnya, baru-baru ini algoritma *hashing* juga diharapkan

meningkatkan standard kemanan mereka dengan memiliki resistensi terhadap *length extension attack*.

*Length Extension Attack* merupakan sebuah serangan terhadap algoritma *hash* yang dapat mengetahui jenis algoritma *hash* yang digunakan dan panjangnya naskah asli tanpa mengetahui jenis algoritma yang digunakan.

Menurut Sobti (Sobti and Geetha, 2012), *secure hash function* memiliki beberapa kriteria atau ketentuan sebagai berikut:

- *H* dapat diaplikasikan kepada blok data dengan panjang berapa pun. (Dalam praktiknya, "panjang berapapun" mugkin ada batasan-batasan tertentu yang ternyata masih lebih besar dari pesan yang pernah di lakukan proses *hash*)

- *H* selalu memproduksi *fixed output* atau nilai yang dihasilkan dari proses *hash* bersifat tetap dan panjangnya sama meskipun

pesan yang diinputkan berbeda. Hal ini tentu bergantung kepada algoritma *hash* yang digunakan, apabila algoritma *hash* yang digunakan sama seharusnya panjang *output* yang dihasilkan juga sama.

- Bila diberikan algoritma *hash H* dan *x* sebagai inputan, komputer akan mudah melakukan kalkulasi dengan *H(x)*.
- Bila diberikan *H* dan *H(x)*, maka tidak mungkin dapat diartikan sebagai *H(x) = H(`x)*.

Tiga ketentuan pertama dapat diaplikasikan kedalam autentikasi pesan dan tanda tangan digital. Sedangkan untuk ketentuan keempat dikenal juga dengan *pre-image resistance* atau *one way property*. *Pre-image resistance* atau *one-way property* adalah berarti mudah untuk dilakukan proses *hashing* namun tidak memungkinkan untuk dikembalikan lagi kedalam naskah asli. Dan untuk ketentuan ke

lima juga dikenal sebagai second *pre-image resistance* sebagai jaminan keamanan sebuah properti.

**a) *Secure Hashing* dan Autentikasi Pengguna**

Ketika seseorang membuat akun pada sebuah aplikasi, dia akan diminta untuk memasukan informasi-informasi untuk melakukan proses registrasi. Diantara informasi-informasi yang disimpan satu diantaranya adalah *password* yang akan dia gunakan untuk login ke dalam sistem tersebut diwaktu yang lain. Informasi-informasi tersebut kemudian akan disimpan oleh sistem kedalam *database* dengan apa adanya, namun tidak dengan *password*. Sebelum disimpan ke dalam *database*, *password* akan dilakukan proses *hashing* sehingga yang tersimpan didalam *database* bukan merupakan *plaintext* dari *password* yang diinputkan melainkan nilai yang dihasilkan dari proses *hashing*.

Hal tersebut bertujuan untuk memberikan proteksi lebih terhadap *password* sehingga tidak sembarang orang dapat menggunakan akun yang telah dibuat melainkan hanya orang yang berhak yang dapat menggunakan akun tersebut. Yang digunakan dalam proses tersebut adalah *hashing* yang merupakan *one-way encryption* bukan enkripsi dua arah yang memungkinkan untuk dilakukan dekripsi terhadap *chipertext* yang dihasilkan. Hal tersebut dikarenakan jika menggunakan enkripsi dua arah akan menimbulkan kerentanan terhadap *password* yang disimpan karena memiliki kunci yang dapat digunakan untuk menemukan nilai asli dari *password* yang disimpan.

**b) Penggunaan *Secure Hashing* pada *Checksum***



Menurut Stallings dan Brown (Stallings and Brown, 2015), *hash* merupakan algoritma yang dapat mengubah *string* menjadi sederetan karakter acak. *Hash* juga disebut sebagai *one-way function* atau *one-way encryption* dikarenakan dalam proses *hashing* hanya

memungkinkan proses enkripsi dan tidak ada kunci yang dapat digunakan untuk melakukan dekripsi. *Hash* seringkali digunakan untuk proses autentikasi dimana autentikasi dapat memastikan sebuah properti tetap bersifat asli atau tidak ada perubahan sehingga dapat digunakan untuk memastikan sebuah properti berasal dari sumber yang dipercaya.

*Hash* dapat melakukan enkripsi terhadap variabel yang besar. Besar atau kecil variabel yang diproses nilai akhir dari proses *hashing* bersifat tetap dan seukuran tergantung pada algoritma yang digunakan. Misalnya hasil dari proses *hashing* menggunakan algoritma MD5 akan selalu berukuran 32 bit, hasil dari proses *hashing* dengan algoritma SHA256 akan selalu berukuran 256 bit, hasil dari proses *hashing* dengan menggunakan algoritma SHA512 akan selalu berukuran 512 bit dan semacamnya. Panjang dari nilai *hash* tersebut berpengaruh terhadap keamanan yang dihasilkan dari setiap algoritma. Semakin

tinggi nilai *hash* yang dihasilkan akan semakin menyulitkan *attacker* untuk mencari naskah asli atau *plaintext* dari *chipertext* yang dihasilkan.

*Hash* biasanya digunakan untuk mengamankan *password* sebelum disimpan ke dalam *database* dan pada proses *login* sebuah sistem untuk mencocokan dengan *password* yang telah disimpan. Selain digunakan dalam proses tersebut, menurut Stallings dan Brown (Stallings and Brown, 2015), *hash* juga kerap kali digunakan untuk memastikan sebuah pesan, dokumen, file atau data-data yang lain benar-benar otentik, terjaga keasliannya dan berasal dari sumber yang terpercaya. Dalam istilah lain penggunaan *hash* tersebut disebut dengan *checksum*.



*Checksum* dapat dikatakan sebagai serangkaian karakter acak yang dapat digunakan untuk memastikan sebuah file tersebut otentik. Dengan menggunakan *checksum* dapat dideteksi bila terjadi kesalahan

terhadap file yang akan digunakan. Sebagai contoh, misalnya dari situs resmi Ubuntu menyediakan ISO yang dapat diunduh dan di-*install* ke perangkat komputer. Setelah melakukan penguduhan file ISO, file yang berhasil diunduh belum pasti sesuai dengan file yang disediakan oleh situs resmi Ubuntu. Terdapat kemungkinan-kemungkinan kesalahan ketika proses transfer file dari server ke perangkat komputer lokal. Untuk memastikan keaslian file dan memastikan tidak terjadi kerusakan selama proses pengunduhan, Ubuntu menyediakan MD5 *Checksum* (pengecekan *checksum* menggunakan algoritma MD5) yang dapat digunakan untuk memastikan apakah file yang berhasil diunduh sama atau tidak dengan file yang berada diserver. Hal serupa juga dapat digunakan apabila kita mendapatkan file atau dokumen dari penyedia layanan pihak ketiga, bukan dari situs resmi, *checksum* dapat digunakan untuk memastikan bahwa file atau dokumen yang kita unduh bersifat otentik dan tidak ada kerusakan.

**c) *Hash Message Authentication Code***

*Hash message authentication code* atau HMAC adalah metode *authentication* untuk pesan atau naskah menggunakan *secure hashing* dan kunci rahasia (Kromodimoeljo, 2009). Menurut Stallings (Stallings and Brown, 2015), *Message authentication code* merupakan salah satu teknik yang melibatkan penggunaan kunci rahasia yang menghasilkan blok kecil dari data. Misalnya ada dua pihak yang saling berkomunikasi (A dan B), berbagi kunci rahasia yang didefinisikan sebagai KAB. Ketika A melakukan pengiriman pesan untuk B, *secure hashing* mengirimkan pesan berupa fungsi yang kompleks dengan disertai dengan kunci rahasianya. Pesan dan kunci rahasisa kemudian dikirimkan kepada penerima atau B. B kemudian melakukan dekripsi pesan dengan menggunakan kunci rahasia yang yang sama dengan yang dikirimkan oleh A.

Menurut Stallings (Stallings and Brown, 2015), jika kunci rahasia yang dikirimkan dengan yang diperhitungkan oleh B sama, maka:

- Penerima dapat memastikan bahwa pesannya belum diubah. Jika *attacker* mengubah pesan tetapi tidak mengubah kode, maka perhitungan kode penerima akan berbeda dari kode yang diterima. Karena *attacker* diasumsikan tidak mengetahui kunci rahasia, *attacker* tidak dapat mengubah kode agar sesuai dengan perubahan dalam pesan.
- Penerima bisa memastikan bahwa pesan tersebut berasal dari pengirim yang sesungguhnya. Karena tidak ada orang lain yang mengetahui kunci rahasia, tidak ada orang lain yang bisa memberikan pesan dengan kode yang sama.

- Jika pesan menyertakan nomor urut (seperti yang digunakan dengan X.25, HDLC, dan TCP), maka penerima dapat memastikan bahwa urutan tersebut merupakan urutan yang tepat, karena *attacker* tidak akan mungkin berhasil mengubah urutan nomor tersebut.

Menurut Kromodimoeldjo (Kromodimoeljo, 2009) , jika *k* merupakan representasi dari kunci rahasia dan *m* adalah representasi dari pesan atau naskah asli, maka rumus yang digunakan dalam *HMAC* adalah sebgai berikut:

$$HMAC(k,m) = H((k \oplus po) \circ H((k \oplus pi) \circ m)).$$



dimana *H* merupakan algoritma *secure hashing* (MD5, SHA1 atau lainnya), ◦ adalah operasi penyambungan (*concatenation*), *po* dan *pi* masing-masing merupakan *padding* sebesar blok yang digunakan *H*. Padding *po* berisi byte *0x5c (hexadecimal)* yang diulang untuk

memenuhi blok, dan *padding* pi berisi byte *0x36 (hexadecimal)* yang diulang untuk memenuhi blok.

Apabila kunci *k* berukuran lebih kecil dari blok, maka *k* di*padding* dengan 0 sampai memenuhi blok. Jika *k* berukuran lebih besar dari blok, maka *k* akan dipotong pada bagian belakangnya sehingga besarnya sama dengan blok. Metode HMAC digunakan karena metode yang menggunakan rumus berikut:

$$MAC(k,m) = H(k \circ m)$$

memiliki kelemahan yaitu kelemahan terhadap serangan *length extension attack*. Tergantung dari fungsi *H*, seseorang dapat menambahkan sesuatu (misalnya ma) ke *m*:

$$m' = m \circ m_a$$

dan tanpa mengetahui nilai *k* dapat membuat simulasi

$$MAC(k, m')$$

Biasanya HMAC digunakan oleh SSL *(Secure Socket Layer)* / TLS dan IPSec. Jika yang digunakan pada HMAC adalah fungsi algoritma MD5 maka dikatakan sebagai HMAC-MD5 dan jika fungsi yang digunakan adalah SHA1 maka dikatakan sebagai HMAC-SHA-1.

**d) Macam-macam Algoritma *Hash***

Berbagai macam algoritma *hash* dapat digunakan untuk mendukung proses autentikasi. Algoritma-algoritma tersebut memiliki kelebihan dan kekurangannya masing-masing. Seiring berkembangnya dunia kemanan informasi, algoritma-algoritma baru pun muncul dan menawarkan keamanan yang lebih terhadap nilai dari proses *hashing*. Beberapa diantara sekian banyak algoritma *hash* yang banyak digunakan adalah sebagai berikut:

1. MD5 (Message Digest 5)

Menurut Kromodimoeldjo (Kromodimoeljo, 2009), algoritma *secure hashing* MD5 merupakan algoritma yang dirancang oleh Ron Rivest dan penggunaannya sangat populer dikalangan komunitas penggiat *open source* untuk digunakan sebagai *checksum* file yang dapat diunduh. MD5 juga digunakan sebagai algoritma untuk memberikan keamanan pada *password* dalam suatu aplikasi sebelum disimpan kedalam *database*. Penggunaan MD5 yang lain adalah sebagai algoritma dalam pengaplikasian *digital signature* dan *certificate*.



Spesifikasi lengkap untuk algoritma MD5 ada pada suatu RFC (*request for comment*). Besarnya blok yang digunakan dalam algoritma MD5 adalah sebesar 512 bit sedangkan *digest size* adalah 128 bit. Karena *word size* ditentukan sebesar 32 bit, satu blok terdiri dari 16 *word* sedangkan *digest* terdiri dari 4 *word*. Indeks untuk bit dimulai

dari 0. *Preprocessing* pada MD5 dimulai dengan *padding* sebagai berikut:

- Bit dengan nilai 1 ditambahkan setelah akhir naskah
- Deretan bit dengan nilai 0 ditambahkan setelah itu sehingga besar dari naskah mencapai nilai 448 (mod 512) (sedikitnya 0 dan sebanyaknya 511 bit dengan nilai 0 ditambahkan sehingga tersisa 64 bit untuk diisi agar besar naskah menjadi kelipatan 512).
- 64 bit yang tersisa diisi dengan besar naskah asli dalam bit. Jika besar naskah asli lebih dari 264 bit maka hanya 64 *lower order* bit yang dimasukkan. *Lower order word* untuk besar naskah asli dimasukkan sebelum *high-order word*.

Setelah *padding*, naskah terdiri dari *n word M[0 . . . n − 1]* dimana *n* adalah kelipatan 16. Langkah berikutnya dalam *preprocessing* adalah menyiapkan *MD buffer* sebesar 4 *word*:

*(A, B, C, D)*

dimana *A* merupakan *lower order word*. Selanjutnya *Buffer* diberi nilai awal (nilai dalam *hexadecimal* dimulai dengan *lower order byte*) sebagai berikut:

A: 01 23 45 67

B: 89 *ab cd ef*

C: *fe dc ba* 98

D: 76 54 32 10.



Proses *hashing* pada algoritma MD5 dilakukan per blok, dengan setiap blok melalui 4 putaran. Proses *hashing* yang dilakukan dengan menggunakan algoritma MD5 menggunakan 4 fungsi *F, G, H* dan *I*

dimana masing-masing diantaranya memiliki input 3 *word* dan output 1 *word*:

$$F(X,Y,Z)=(X\wedge Y)\vee(\neg X\wedge Z)$$

$$G(X,Y,Z)=(X\wedge Z)\vee(Y\wedge\neg Z)$$

$$H(X,Y,Z)=X\oplus Y\oplus Z$$

$$I(X,Y,Z)=Y\oplus(X\vee\neg Z)$$

Selain keempat fungsi diatas, proses *hashing* pada algoritma MD5 juga menggunakan tabel dengan total 64 *word*, *T[1]*. Nilai yang tersimpan dalam tabel yang digunakan oleh algoritma MD5 berbentuk *hexadecimal*. Berikut adalah tabel yang digunakan oleh algoritma MD5:

Tabel 3: Tabel yang digunakan MD5 dalam melakukan proses *hashing*

| ***T*** | ***Word*** | ***T*** | ***Word*** |
|---|---|---|---|
| 1 | d76aa478 | 33 | fffa3942 |
| 2 | e8c7b756 | 34 | 8771f681 |
| 3 | 242070db | 35 | 6d9d6122 |
| 4 | c1bdceee | 36 | fde5380c |
| 5 | f57c0faf | 37 | a4beea44 |
| 6 | 4787c62a | 38 | 4bdecfa9 |
| 7 | a8304613 | 39 | f6bb4b60 |
| 8 | fd469501 | 40 | bebfbc70 |
| 9 | 698098d8 | 41 | 289b7ec6 |
| 10 | 8b44f7af | 42 | eaa127fa |
| 11 | ffff5bb1 | 43 | d4ef3085 |
| 12 | 895cd7be | 44 | 4881d05 |
| 13 | 6b901122 | 45 | d9d4d039 |
| 14 | fd987193 | 46 | e6db99e5 |
| 15 | a679438e | 47 | 1fa27cf8 |

| *T* | *Word* | *T* | *Word* |
|---|---|---|---|
| 16 | 49b40821 | 48 | c4ac5665 |
| 17 | f61e2562 | 49 | f4292244 |
| 18 | c040b340 | 50 | 432aff97 |
| 19 | 265e5a51 | 51 | ab9423a7 |
| 20 | e9b6c7aa | 52 | fc93a039 |
| 21 | d62f105d | 53 | 655b59c3 |
| 22 | 2441453 | 54 | 8f0ccc92 |
| 23 | d8a1e681 | 55 | ffeff47d |
| 24 | e7d3fbc8 | 56 | 85845dd1 |
| 25 | 21e1cde6 | 57 | 6fa87e4f |
| 26 | c33707d6 | 58 | fe2ce6e0 |
| 27 | f4d50d87 | 59 | a3014314 |
| 28 | 455a14ed | 60 | 4e0811a1 |
| 29 | a9e3e905 | 61 | f7537e82 |
| 30 | fcefa3f8 | 62 | bd3af235 |
| 31 | 676f02d9 | 63 | 2ad7d2bb |

| *T* | *Word* | *T* | *Word* |
|---|---|---|---|
| 32 | 8d2a4c8a | 64 | eb86d391 |

Secara garis besar proses *hashing* pada algoritma MD5 untuk satu blok adalah sebagai berikut (menggunakan *array* 16 *word X[0]* sampai dengan *X[15]* yang dapat menyimpan satu blok):

- Copy satu blok (*wordM[16i]* sampai dengan *M[16i+15]*) ke *X[0]* sampai dengan *X[15]*.
- Simpan A,B, C,D dalam A?, B?, C?,D?.
- Lakukan putaran 1 pada A,B, C,D.
- Lakukan putaran 2 pada A,B, C,D.
- Lakukan putaran 3 pada A,B, C,D.
- Lakukan putaran 4 pada A,B, C,D.

- Tambahkan nilai simpanan pada A,B, C,D:

$$A \leftarrow (A + A')\ mod\ 2^{32},$$

$$B \leftarrow (B + B')\ mod\ 2^{32},$$

$$C \leftarrow (C + C')\ mod\ 2^{32},$$

$$D \leftarrow (D + D')\ mod\ 2^{32}.$$

Algoritma diatas diulang terus-menerus samapai semua blok dalam *M* terproses (dimulai dengan $i = 0$ sampai dengan $i = n/16 - 1$). Putaran dari 1 sampai dengan 4 yang dilakukan pada setiap proses *A,B,C,D* masing-masing berbeda. Untuk putaran 1, operasi *P1(A,B, C,D, k, s, i)* didefinisikan sebagai berikut:

$$A \leftarrow B + ((A+F(B, C,D) + X[k] + T[i]) <<< s)$$

dimana X <<< s adalah rotasi *X* kekiri *s* bit. Putaran 1 terdiri dari:

*P1(A,B, C,D, 0, 7, 1),* *P1(D, A,B, C, 1, 12, 2),*

*P1(C,D, A,B, 2, 17, 3),* *P1(B, C,D, A, 3, 22, 4),*

*P1(A,B, C,D, 4, 7, 5),* *P1(D, A,B, C, 5, 12, 6),*

*P1(C,D, A,B, 6, 17, 7),* *P1(B, C,D, A, 7, 22, 8),*

*P1(A,B, C,D, 8, 7, 9),* *P1(D, A,B, C, 9, 12, 10),*

*P1(C,D, A,B, 10, 17, 11),* *P1(B, C,D, A, 11, 22, 12),*

*P1(A,B, C,D, 12, 7, 13),* *P1(D, A,B, C, 13, 12, 14),*

*P1(C,D, A,B, 14, 17, 15),* *P1(B, C,D, A, 15, 22, 16).*



Sedangkan untuk putaran ke-2, operasi *P2(A,B, C,D, k, s, i)* didefinisikan sebagai berikut:

$$A \leftarrow B + ((A+G(B, C,D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi $X$ kekiri $s$ bit. Putaran 2 terdiri dari:

*P2(A,B, C,D, 1, 5, 17),* *P2(D, A,B, C, 6, 9, 18),*

*P2(C,D, A,B, 11, 14, 19),* *P2(B, C,D, A, 0, 20, 20),*

*P2(A,B, C,D, 5, 5, 21),* *P2(D, A,B, C, 10, 9, 22),*

*P2(C,D, A,B, 15, 14, 23),* *P2(B, C,D, A, 4, 20, 24),*

*P2(A,B, C,D, 9, 5, 25),* *P2(D, A,B, C, 14, 9, 26),*

*P2(C,D, A,B, 3, 14, 27),* *P2(B, C,D, A, 8, 20, 28),*

*P2(A,B, C,D, 13, 5, 29),* *P2(D, A,B, C, 2, 9, 30),*

*P2(C,D, A,B, 7, 14, 31),* *P2(B, C,D, A, 12, 20, 32).*

Untuk putaran selanjutnya atau putaran ke-3, operasi *P3(A,B, C,D, k, s, i)* didefinisikan sebagai berikut:

$$A \leftarrow B + ((A+H(B, C,D) +X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi $X$ kekiri $s$ bit. Putaran 3 terdiri dari:

*P3(A,B, C,D, 5, 4, 33),* *P3(D, A,B, C, 8, 11, 34),*

*P3(C,D, A,B, 11, 16, 35),* *P3(B, C,D, A, 14, 23, 36),*

*P3(A,B, C,D, 1, 4, 37),* *P3(D, A,B, C, 4, 11, 38),*

*P3(C,D, A,B, 7, 16, 39),* *P3(B, C,D, A, 10, 23, 40),*

*P3(A,B, C,D, 13, 4, 41),* *P3(D, A,B, C, 0, 11, 42),*

*P3(C,D, A,B, 3, 16, 43),* *P3(B, C,D, A, 6, 23, 44),*

*P3(A,B, C,D, 9, 4, 45),* *P3(D, A,B, C, 12, 11, 46),*

*P3(C,D, A,B, 15, 16, 47),* *P3(B, C,D, A, 2, 23, 48).*

Untuk putaran 4, operasi *P4(A,B, C,D, k, s, i)* didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + I(B, C,D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi $X$ kekiri $s$ bit. Putaran 4 terdiri dari:

*P4(A,B, C,D, 0, 6, 49),* *P4(D, A,B, C, 7, 10, 50),*

*P4(C,D, A,B, 14, 15, 51),* *P4(B, C,D, A, 5, 21, 52),*

*P4(A,B, C,D, 12, 6, 53),* *P4(D, A,B, C, 3, 10, 54),*

*P4(C,D, A,B, 10, 15, 55),* *P4(B, C,D, A, 1, 21, 56),*

*P4(A,B, C,D, 8, 6, 57),* *P4(D, A,B, C, 15, 10, 58),*

*P4(C,D, A,B, 6, 15, 59),* *P4(B, C,D, A, 13, 21, 60),*

*P4(A,B, C,D, 4, 6, 61),* *P4(D, A,B, C, 11, 10, 62),*

*P4(C,D, A,B, 2, 15, 63),* *P4(B, C,D, A, 9, 21, 64).*

Setelah semua blok selesai terproses, hasil dari *A, B, C, D* menjadi MD5 *digest* dari naskah asli. Urutan *byte* untuk *digest* dimulai dengan *lower order byte* dari *A* dan diahiri oleh *higher order byte* yaitu *D*.

Beberapa peneliti sebelumnya telah berhasil membuat *collision* untuk algoritma MD5. Salah satu diantaranya adalah Xiaoyun Wang dan rekannya (Wang and Yu, 2005) yang berhasil membuat *collision* untuk naskah yang masing-masing terdiri dari 2 blok. Wang menggunakan teknik *differential cryptanalysis* dengan menerapkan 2 perbedaan yaitu perbedaan bit *(exclusive or)* dan selisih (menggunakan pengurangan).

Penerapan *differential cryptanalysis* biasanya digunakan dalam algoritma DES. Pada algoritma DES penggunaan *cryptanalysis* berfokus kepada perbedaan bit. Sedangkan Wang menggunakan *differential cryptanalysis* dengan dikombinasikan dengan selisih. Kombinasi ini menghasilkan perbedaan yang lebih rinci. Dalam upaya meningkatkan probabilitas karakteristik *differential* dalam penggunaannya, naskah yang dibuat secara acak dapat dilakukan proses modifikasi. Modifikasi naskah dilakukan agar karakteristik putaran pertama dijamin mempunyai probabilitas 1, baik untuk tahap 1 maupun tahap 2. Modifikasi naskah juga dilakukan agar sebagian dari persyaratan untuk putaran kedua terpenuhi. Dengan melakukan berbagai modifikasi tersebut, probabilitas karakteristik tahap pertama ditingkatkan menjadi $2^{-30}$ sedangkan probabilitas karakteristik tahap kedua ditingkatkan menjadi $2^{-30}$ . Dengan menggunakan

metode ini, Wang berhasil menemukan *collision* untuk MD5 dalam waktu kurang dari 1 jam dengan komputer.

2. SHA

Algoritma *secure hashing* SHA merupakan algoritma yang dirancang dan dikembangkan oleh National Security Agency (NSA). Terdapat 4 macam SHA dengan parameter yang berbeda-beda. Keempat macam algoritma SHA dan perbandingannya dapat dilihat pada tabel di bawah ini:

Tabel 4: Macam-macam algoritma SHA dan perbedaanya

| Algoritma | Naskah (bit) | Blok (bit) | Word (bit) | Digest (bit) | Kemanan (bit) |
|---|---|---|---|---|---|
| SHA-1 | $< 2^{64}$ | 512 | 32 | 160 | 80 |
| SHA-256 | $< 2^{64}$ | 512 | 32 | 256 | 128 |
| SHA-384 | $< 2^{128}$ | 1024 | 64 | 384 | 192 |

| Algoritma | Naskah (bit) | Blok (bit) | Word (bit) | Digest (bit) | Kemanan (bit) |
|---|---|---|---|---|---|
| SHA-512 | $< 2^{128}$ | 1024 | 64 | 512 | 256 |

SHA-1 memiliki kapasitas input sebesar $2^{64}$ -1 dimana 160 bit merupakan hasil output dari proses *hashing* dan $2^{80}$ evaluasi kemanan algoritma *hash*-nya. SHA 256 dan 384 tidak banyak digunakan meskipun memiliki keamanan yang lebih baik dibandingkan SHA-1 namun proses *hashing* yang memakan waktu lama membuat algoritma SHA-256 dan SHA-384 tidak banyak diminati.

Menurut Mulya (Mulya, 2009), SHA-512 merupakan pengembangan dari SHA-1 dan dalam meningkatkan keamanannya algoritma ini berbasis kepada MD4. Panjang dari nilai *hash* yang dihasilkan dari algoritma SHA-512 adalah senilai 512 bit. SHA-512 sebagai salah satu dari fungsi algoritma *hash*, memiliki sifat-sifat berikut:

- *h* mudah untuk dihitung apabila diberikan nilai *M*

  Sifat ini merupakan sebuah keharusan, karena apabila *h* sukar untuk dihitung maka fungsi *hash* tersebut tidak dapat digunakan dengan maksimal.

- *M* tidak dapat dihitung jika hanya diketahui nilai *h*

  Sifat ini disebut juga dengan *one-way function* seperti sifat yang dimiliki oleh algoritma *hash* pada umumnya.

- Tidak mungkin dicari nilai *M* dan *M`* sedemikian sehingga *H(M) = H(`M)*



  Sifat ini disebut juga *collision free* dimana dapat digunakan sebagai cara untuk mencegah pemalsuan.

Keamanan algoritma SHA didasarkan pada fakta bahwa *birthday attack* pada *digest* sebesar *n* bit menghasilkan *collision* dengan faktor kerja sekitar 2n/2 (Kromodimoeljo, 2009).

**6. *Security Test***

*Security test* adalah uji coba yang digunakan untuk mengetahui seberapa aman kah sebuah aplikasi web dalam menerapkan pengamanan sistem. Namun, *security testing* tidak menjamin bahwa aplikasi yang diuji coba aman tetapi hal ini penting dilakukan untuk melihat kemampuan sistem seberapa baik dalam hal perlindungan. Secara umum, *security test* terbagi dalam beberapa jenis sesuai dengan metodologi yang digunakan (Sofyan, 2014). Gambar 5 berikut menunjukan macam-macam dari *security testing*:


Gambar 6: Security Testing



Menurut OWASP Testing Guide (The OWASP Foundation, 2014), testing adalah proses membandingkan keadaan suatu sistem atau aplikasi terhadap serangkaian kriteria. Dalam industri keamanan orang sering menguji terhadap serangkaian kriteria mental yang tidak

didefinisikan dengan baik atau lengkap. Sebagai akibatnya, banyak orang luar menganggap pengujian keamanan sebagai hal yang buruk. Presepsi itu sebenarnya tidaklah benar. Pengujian akan membantu organisasi menemukan versi terbaik dari sebuah aplikasi yang berjalan di organisasinya sehingga akan memastikan data yang tersebar pada aplikasi tidak mengalami kerusakan dan perubahan kecuali sesuai dengan kriteria yang ditentukan.

Organisasi harus dapat memahami apa saja yang termasuk dalam program pengujian. Mereka harus mengidentifikasi langkah-langkah yang perlu dilakukan untuk membangun dan mengoperasikan program pengujian pada sbeuah aplikasi. Organisasi dapat menmeukan banyak panduan yang tersebar di internet mengenai skema pengujian, atau dapat mendatangkan ahli keamanan untuk dapat memberikan panduan praktis mengenai proses menjalankan pengamanan pada sebuah aplikasi sesuai standar yang diperlukan. Panduan ini memberikan

pandangan secara luas tentang elemen-elemen yang diperlukan untuk membuat program keamanan aplikasi yang komprehensif. Panduan ini dapat digunakan sebagai panduan referensi dan sebagai metodologi untuk membantu menentukan kesenjangan antara praktik yang ada dan praktik terbaik dalam dunia industri. Panduan ini memungkinkan organisasi untuk membandingkan diri mereka dengan rekan-rekan yang lain dalam dunia industri. Untuk memahami besarnya sumber daya yang diperlukan untuk menguji dan memelihara perangkat lunak, atau untuk mempersiapkan audit.

**a) Prinsip-prinsip *Security Testing***



Menurut Felderer (Felderer *et al.*, 2016),*Security Testing* adalah pengujian persyaratan keamanan yang terkait dengan properti keamanan seperti kerahasiaan, integritas, ketersediaan, otentikasi, otorisasi, dan penolakan. *Security Testing* atau pengujian keamanan seharusnya dapat mengidentifikasi apakah sebuah properti keamanan

yang ditentukan atau yang dimaksudkan, untuk serangkaian aset tertentu, diimplementasikan dengan benar. Hal ini dapat dilakukan dengan mencoba menunjukkan kesesuaian dengan properti keamanan (mirip dengan pengujian berbasis persyaratan) atau dengan mencoba mengatasi kerentanan yang telah diketahui, yang mirip dengan pengujian berbasis kesalahan tradisional, atau destruktif. Secara intuitif, pengujian kesesuaian mempertimbangkan input yang diharapkan dengan baik. Hal ini bermaksud untuk menguji apakah sistem memenuhi sifat keamanan sehubungan dengan input yang diharapkan ini didefinisikan dengan baik atau tidak. Sebaliknya, menangani kerentanan yang diketahui berarti menggunakan data input berbahaya yang tidak diharapkan yang kemungkinan akan mengeksploitasi kerentanan yang dianggap memiliki celah.

Berdasarkan Open Web Application Project (OWASP) Testing Guide (The OWASP Foundation, 2014), prinsip-prinsip testing diantaranya adalah sebagai berikut:

- *There is No Silver Bullet*

Meskipun kebanyakan orang berpikir bahwa *firewall* aplikasi akan memberikan banyak pertahanan terhadap serangan atau mengidentifikasi sejumlah masalah, pada kenyataannya tidak ada sebuah kepastian yang dapat menjamin bahwa sebuah perangkat lunak benar-benar aman. Beberapa *software* yang digunakan untuk pengujian aplikasi, meskipun berguna sebagai jalan pembuak pertama yang dapat digunakan sebagai acuan namun tidak selamanya *software* tersebut dapat menjamin bahwa hanya ada masalah-masalah yang berhasil ditemukan oleh *software*. Dalam artian lain masih banyak celah yang mungkin saja dimiliki aplikasi yang tidak dapat diterka oleh *software* pengujian.

- *Think Strategically, Not Tactically*

Selama beberapa tahun terakhir para profesional dalam bidang keamanan telah menyadari kekeliruan dari model *patch-and-penetrate* yang meresap dalam keamanan informasi selama tahun 1990-an. Model *patch-and-penetrate* melibatkan perbaikan bug yang dilaporkan, tetapi tanpa investigasi yang tepat dari akar penyebabnya. Evolusi kerentanan dalam perangkat lunak umum yang digunakan di seluruh dunia telah menunjukkan ketidakefektifan model ini.

Beberapa penelitian tentang kerentanan telah menunjukkan bahwa dengan waktu reaksi penyerang di seluruh dunia, waktu yang dibutuhkan utntuk pemasangan *patch* tidaklah cukup. Hal tersebut dikarenakan waktu antara kerentanan yang terbuka dan serangan otomatis terhadap itu sedang dikembangkan dan dirilis semakin berkurang setiap tahun.

Terdapat beberapa asumsi yang salah dalam terhadap model *patch-and-penetrate*. Banyak pengguna percaya bahwa *patch* mengganggu operasi normal dan mungkin merusak aplikasi yang ada. Selain itu asumsi bahwa semua pengguna mengetahui *patch* yang baru dirilis juga tidaklah benar. Akibatnya tidak semua pengguna produk akan menerapkan *patch*, baik karena mereka berpikir *patch* dapat mengganggu cara kerja perangkat lunak atau karena mereka tidak memiliki pengetahuan tentang keberadaan *patch*. Sangat penting untuk membangun keamanan ke dalam *Software Development Life Cycle* (SDLC) untuk mencegah berulangnya masalah keamanan dalam aplikasi. Pengembang dapat membangun keamanan ke dalam SDLC dengan mengembangkan standar, kebijakan, dan pedoman yang sesuai dan bekerja dalam metodologi pengembangan. Pemodelan ancaman dan teknik lainnya harus digunakan untuk membantu menetapkan

sumber daya yang sesuai untuk bagian-bagian sistem yang paling berisiko.

- *The SDLC is King*

SDLC merupakan proses yang sangat dikenal bagi para pengembang aplikasi. Dengan mengintegrasikan keamanan ke dalam setiap fase SDLC, memungkinkan untuk pendekatan holistik untuk keamanan aplikasi yang memanfaatkan prosedur yang sudah ada dalam organisasi. Nama-nama dari berbagai fase waktu pengembangan dapat berubah tergantung pada model SDLC yang digunakan oleh suatu organisasi. Setiap fase-fase konseptual dari SDLC tersebut kemudian akan digunakan sebagai acuan dalam mengembangkan aplikasi, misalnya mendefinisikan, merancang, mengembangkan, menyebarkan dan memelihara. Setiap fase memiliki pertimbangan keamanan yang harus menjadi bagian dari proses yang ada, untuk memastikan program keamanan yang efektif dan komprehensif.

Terdapat beberapa kerangka kerja *Secure* SDLC yang dapat memberikan saran deskriptif dan preskriptif bagi para pengembang dan organisasi. Saran-saran deskriptif atau preskriptif tersebut dapat diambil atau tidak tergantung pada kematangan proses SDLC. Pada dasarnya, saran preskriptif menunjukkan bagaimana *Secure* SDLC harus bekerja, dan saran deskriptif menunjukkan bagaimana penerapan SDLC digunakan di dunia nyata. Keduanya memiliki tempat masing-masing. Sebagai contoh, jika organisasi tidak mengetahui dalam melakukan pengembangan harus mulai dari mana, kerangka kerja preskriptif dapat menyediakan menu kontrol keamanan potensial yang dapat diterapkan dalam SDLC. Penjelasan-penjelasan deskriptif kemudian dapat membantu mengarahkan proses pengambilan keputusan dengan menghadirkan riwayat tentang proses apa saja yang telah bekerja dengan baik untuk organisasi lain.

- *Test Early and Test Often*

Ketika bug terdeteksi lebih awal dalam proses pengembangan sistem, bug tersebut dapat diatasi lebih cepat dan dengan biaya yang lebih rendah. Keamanan terhadap bug tidak berbeda dengan bug fungsional atau berbasis kinerja. Langkah praktis yang dapat dilakukan dalam penganangan bug pada proses pengembangan adalah dengan mendidik tim pengembangan dan QA *(Quality Assurance)* tentang masalah keamanan umum dan bagaimana cara untuk mendeteksi dan mencegahnya. Meskipun ada banyak pustaka yang telah mengidentifikasi bug-bug yang pernah muncul, ancaman baru akan muncul terus-menerus dan pengembang harus menyadari ancaman yang mempengaruhi perangkat lunak yang mereka kembangkan. Pendidikan dalam pengujian keamanan juga membantu pengembang memperoleh pola pikir yang sesuai untuk menguji aplikasi dari perspektif penyerang. Dengan demikian setiap organisasi seharusnya

mempertimbangkan masalah keamanan sebagai bagian dari tanggung jawab mereka.

- *Understand the Scope of Security*

Hal penting yang harus dipahami oleh pengembang adalah mengetahui berapa banyak keamanan yang diperlukan oleh suatu proyek. Informasi dan aset yang harus dilindungi harus diberi klasifikasi yang menyatakan bagaimana penanganannya. Diskusi-diskusi mengenai hal tersebut harus dilakukan dengan organisasi yang bersangkutan dan pihak-pihak yang memahami hukum yang berlaku di negara tersebut. Hal tersebut dilakukan karena terkait klasifikasi properti organisasi yang memiliki properti yang lebih tau bagaimana klasifikasi yang sesuai dan terkait dengan memahami hukum hal tersebut perlu dilakukan agar aplikasi yang dibangun tidak melanggar ketentuan-ketentuan hukum yang ada dimana aplikasi tersebut dijalankan.

- *Develop the Right Mindset*

Keberhasilan dalam menguji suatu aplikasi terhadap kerentanan keamanan membutuhkan pemikiran *out of the box*. Kasus penggunaan aplikasi secara normal dapat disikapi dengan pola yang normal, karena pengguna menyesuaikan langkah-langkah yang mereka lakukan terhadap cara yang kita sediakan. Namun berbeda dengan kasus keamana karena para penyerang dalam menyerang sebuah aplikasi selalu menggunakan cara yang tidak dapat disangka-sangka. Dengan demikian dalam menyikapi serangan keamanan harus menggunakan pola pikir yang tidak biasa pula. Pengujian keamanan yang baik harus dapat melampaui pemikiran-pemikiran nalar kebanyakan orang dan berpikir seperti seorang penyerang yang mencoba merusak aplikasi. Berpikir kreatif dapat membantu menentukan data tak terduga apa yang dapat menyebabkan aplikasi memiliki kerentanan. Hal ini juga dapat membantu menemukan asumsi apa yang dibuat oleh

pengembang web yang tidak selalu benar dan bagaimana mereka dapat melakukan kesalahan dalam pengembangan. Salah satu alasan mengapa *software* pengujian kemaanan otomatis benar-benar buruk dalam menguji kerentanan adalah bahwa pemikiran kreatif ini harus dilakukan berdasarkan kasus per kasus.

- *Understand the Subject*

Salah satu langkah utama yang harus dilakukan dalam program keamanan yang baik adalah dengan meminta dokumentasi aplikasi yang akurat. Arsitektur, diagram aliran data, kasus penggunaan, dll, harus ditulis dalam dokumen formal dan tersedia untuk ditinjau. Spesifikasi teknis dan dokumen aplikasi harus mencakup informasi yang mencantumkan tidak hanya kasus penggunaan yang diinginkan, tetapi juga kasus penggunaan yang tidak diizinkan secara khusus. Dengan adanya dokumentasi dari aplikasi yang dijadikan subjek, pengetahuan akan aplikasi akan sangat membantu mengenal aplikasi

lebih dalam sebagaimana dengan apa yang dikembangkan oleh para pengembang aplikasi.

- *Use the Right Tools*

Meskipun tidak ada tools yang benar-benar dapat digunakan sebagai senjata pamungkas, namun penggunaan tools yang tepat dapat membantu proses dalam melakukan pengujian keamanan. Terdapat banyak toosls pengujian kemanan baik itu *open source* atau pun enterprise yang dapat mengotomatisasi banyak tugas keamanan standar. Alat-alat tersebut dapat menyederhanakan dan mempercepat proses keamanan dengan membantu petugas keamanan dalam tugas mereka. Namun, penting untuk dapat dipahami bahwa tidak semua proses dapat dilakukan oleh tools sehingga harus bisa dibedakan dengan tepat apa yang bisa dan tidak bisa dilakukan alat-alat yang digunakan.

- *The Devil is in the Details*

Sangat penting untuk tidak melakukan tinjauan keamanan yang dangkal dari suatu aplikasi dan menganggapnya telah lengkap dan mendalam. Hal yang demikian akan menanamkan rasa percaya diri yang salah yang bisa sama berbahayanya dengan tidak melakukan tinjauan keamanan sejak awal. Sangat penting meninjau kembali temuan-temuan tersebut dan menyingkirkan segala kesalahan positif yang mungkin tetap ada dalam laporan. Melaporkan temuan keamanan yang salah sering dapat merusak pesan valid dari sisa laporan keamanan. Perawatan harus diambil untuk memverifikasi bahwa setiap bagian yang mungkin dari logika aplikasi telah diuji, dan bahwa setiap skenario kasus penggunaan dieksplorasi untuk kemungkinan kerentanan.

- *Use Source Code When Available*

Sementara hasil tes *blackbox* bisa mengesankan dan berguna untuk menunjukkan bagaimana kerentanan diekspos dalam lingkungan produksi, pengujian *blackbox* bukan cara yang paling efektif atau efisien untuk mengamankan aplikasi. Pengujian dinamis sangat sulit untuk dapat menguji seluruh basis kode, terutama jika ada banyak pernyataan bersyarat yang ada. Jika *source code* untuk aplikasi tersedia, harus diberikan kepada staf keamanan untuk membantu mereka saat melakukan peninjauan. Mungkin untuk menemukan kerentanan dalam sumber aplikasi yang akan terjawab selama keterlibatan *blackbox*.



- *Develop Metrics*

Bagian penting dari program keamanan yang baik adalah kemampuan untuk menentukan apakah semuanya menjadi lebih baik

atau tidak. Sangat penting untuk melacak hasil keterlibatan pengujian dalam progres organisasi, dan mengembangkan matrik yang akan mengungkapkan tren keamanan aplikasi dalam organisasi. Matrik yang bagus akan dapat menampilkan beberapa informasi sebagai berikut:

- ➔ Diperlukan pendidikan/pelatihan terhadap staf yang berkaitan
- ➔ Keberadaan mekanisme yang tidak sepenuhnya dipahami oleh pengembang aplikasi
- ➔ Jumlah insiden tentang keamanan meningkat/ menurun.



Metrik yang konsisten yang dapat dihasilkan secara otomatis dari *source code* yang tersedia juga akan membantu organisasi dalam menilai efektivitas mekanisme yang diperkenalkan untuk mengurangi bug keamanan dalam pengembangan perangkat lunak. Metrik tidak mudah dikembangkan, jadi menggunakan standar metrik seperti yang

disediakan oleh proyek organisasi-organisasi penyedia layanan keamanan adalah hal yang baik untuk diadaptasi.

- *Document the Test Results*

Untuk menyimpulkan proses pengujian, dapat digunakan catatan resmi tentang tindakan pengujian apa yang diambil, oleh siapa, kapan tindakan itu dilakukan, dan rincian temuan dari pengujian yang telah dilakukan. Format yang digunakan harusnya dapat disepakati dan seharusnya dapat diterima oleh semua pihak yang memiliki kepentingan dan keterlibatan didalamnya. Pihak-pihak tersebut meliputi pengembang, project manager, *bussines owner*, departemen TI, auditor, staff dan lain sebagainya.



Laporan yang diberikan kepada *bussines owner* harus jelas dalam mengidentifikasi dimana risiko materil ada dan cukup untuk mendapatkan dukungan mereka untuk tindakan mitigasi selanjutnya.

Laporan tersebut juga harus jelas kepada pengembang dalam menunjukkan dengan tepat fungsi persis yang dipengaruhi oleh kerentanan dan rekomendasi terkait untuk menyelesaikan masalah dalam bahasa yang akan dipahami pengembang. Laporan tersebut juga harus memungkinkan penguji keamanan lain mereproduksi hasilnya. Menulis laporan itu seharusnya tidak terlalu membebani penguji keamanan itu sendiri. Penguji keamanan pada umumnya tidak terkenal karena keterampilan menulis yang baik dan menyetujui laporan yang kompleks dapat menyebabkan contoh dimana hasil tes tidak didokumentasikan dengan baik. Menggunakan template laporan pengujian keamanan dapat menghemat waktu dan memastikan bahwa hasilnya didokumentasikan secara akurat dan konsisten, dan berada dalam format yang sesuai untuk audiens.

***b) Manajemen Resiko dan Security Testing***

Praktisi perangkat lunak memiliki banyak tugas yang harus dilakukan dalam mengelola resiko yang mungkin akan ditimbulkan dari aplikasi yang dikembangkan. Tugas-tugas tersebut menurut Potter dan McGraw (Potter and McGraw, 2004) adalah sebagai berikut:

- menciptakan kasus penyalahgunaan atau pelanggaran keamanan
- membuat daftar persyaratan keamanan normatif
- membuat arsitektur analisis risiko
- membangun rencana uji keamanan berbasis risiko
- menggunakan alat analisa statis
- melakukan tes keamanan

- melakukan pengujian penetrasi di lingkungan final, dan
- membersihkan setelah pelanggaran keamanan,

***c) Deriving Security Testing Requirements***

Berdasarkan OWASP Testing Guide (The OWASP Foundation, 2014), untuk mendapatkan program pengujian yang sukses, tujuan dari pengujian harus diketahui sebelumnya. Tujuan-tujuan ini ditentukan oleh persyaratan keamanan. Bagian ini membahas secara rinci bagaimana mendokumentasikan persyaratan untuk pengujian keamanan dengan menurunkannya dari standar dan peraturan yang berlaku, dan dari persyaratan aplikasi positif dan negatif. Pada bagian ini juga membahas bagaimana persyaratan keamanan secara efektif mendorong pengujian keamanan selama SDLC dan bagaimana data uji keamanan dapat digunakan untuk secara efektif mengelola risiko keamanan perangkat lunak.

- *Testing Objectives*

Salah satu tujuan pengujian keamanan berdasarkan OWASP Testing Guide (The OWASP Foundation, 2014) adalah untuk memvalidasi bahwa kontrol keamanan beroperasi seperti yang diharapkan. Tujuan pengujian ini didokumentasikan melalui persyaratan keamanan yang menggambarkan fungsionalitas dari kontrol keamanan. Pada tingkatan yang lebih tinggi, proses pengujian berarti membuktikan kerahasiaan, integritas, dan ketersediaan data serta layanan. Tujuan lainnya adalah untuk memvalidasi bahwa kontrol keamanan dilaksanakan dengan sedikit atau tanpa kerentanan.



- *Security Requirements Documentation*

Langkah pertama yang dapat dilakukan dalam mendokumentasi kan persyaratan keamanan adalah dengan memahami persyaratan dari proses bisnis yang dimiliki. Dokumen persyaratan bisnis dapat

memberikan informasi yang sangat baik sebagai permulaan tentang fungsionalitas yang diharapkan dari aplikasi. Misalnya, tujuan utama aplikasi mungkin untuk menyediakan layanan keuangan kepada pelanggan atau untuk memungkinkan barang dibeli dari katalog online. Bagian keamanan dari persyaratan bisnis harus menyoroti kebutuhan untuk melindungi data pelanggan serta untuk mematuhi dokumentasi keamanan yang berlaku seperti peraturan, standar, dan kebijakan.

Membuat ceklis mengenai auturan, standar, dan kebijakan yang berlaku adalah analisis kepatuhan keamanan awal yang baik untuk aplikasi web. Misalnya, peraturan kepatuhan dapat diidentifikasi dengan memeriksa informasi tentang sektor bisnis dan negara atau negara tempat aplikasi akan beroperasi. Beberapa pedoman dan peraturan kepatuhan ini dapat diterjemahkan ke dalam persyaratan teknis khusus untuk kontrol keamanan.

Standar industri yang berlaku untuk keamanan juga harus tertulis dalam ceklis persyaratan keamanan. Misalnya, pada kasus yang menangani data kartu kredit pelanggan, bukti kepatuhan terhadap standar keamanan yang ditetapkan diantaranya adalah dengan tidak boleh menyimpan PIN, melindungi strip magnetik dan melakukan enkripsi terhadap transaksi data.

Bagian lain dari ceklis perlu mencantumkan persyaratan umum mengenai regulasi keamanan dengan standar dan kebijakan keamanan informasi organisasi. Dari perspektif persyaratan fungsional, persyaratan untuk kontrol keamanan perlu dipetakan ke bagian spesifik dari standar keamanan informasi. Contoh persyaratan tersebut dapat berupa kompleksitas kata sandi dari enam karakter alfanumerik harus dikontrol otentikasi yang digunakan oleh aplikasi. Ketika persyaratan keamanan dipetakan ke aturan kepatuhan, tes keamanan dapat memvalidasi paparan risiko yang ditimbulkan. Jika pelanggaran

dengan standar dan kebijakan keamanan informasi ditemukan, hal ini akan dapat menghasilkan risiko yang dapat didokumentasikan dan bisnis yang harus dikelola. Karena persyaratan kepatuhan keamanan ini dapat difungsikan dengan semestinya, persyaratan tersebut harus didokumentasikan dengan baik dan divalidasi dengan tes keamanan.

- *Security Requirements Validation*

Dari perspektif fungsionalitas, validasi persyaratan keamanan adalah tujuan utama pengujian keamanan. Dari perspektif manajemen risiko, validasi persyaratan keamanan adalah tujuan dari penilaian keamanan informasi. Pada tingkatan yang lebih tinggi, tujuan utama penilaian keamanan informasi adalah mengidentifikasi kesenjangan dalam kontrol keamanan, seperti kurangnya otentikasi dasar, automatisasi, atau kontrol enkripsi. Lebih mendalam, tujuan penilaian keamanan adalah analisis risiko, seperti identifikasi kelemahan potensial dalam kontrol keamanan yang memastikan kerahasiaan,

integritas, dan ketersediaan data. Misalnya, ketika aplikasi berurusan dengan informasi pengidentifikasi data pribadi dan data sensitif, persyaratan keamanan yang akan divalidasi adalah kepatuhan terhadap kebijakan keamanan informasi perusahaan yang memerlukan enkripsi data tersebut dalam perjalanan dan penyimpanan. Dengan asumsi enkripsi digunakan untuk melindungi data, algoritma enkripsi dan panjang kunci harus mematuhi standar enkripsi organisasi. Ini mungkin mengharuskan hanya algoritma dan panjang kunci tertentu yang dapat digunakan. Sebagai contoh, persyaratan keamanan yang dapat diuji misalnya menerapkan algoritma tertentu sebagai upaya dalam meningkatkan keamanan pada *password* (misalnya, SHA-256, RSA, AES) dengan panjang kunci minimum yang diizinkan (misalnya, lebih dari 128 bit untuk simetris dan lebih banyak lagi) dari 1024 untuk enkripsi asimetris).

Dari perspektif penilaian keamanan, persyaratan keamanan dapat divalidasi pada fase yang berbeda dari SDLC dengan menggunakan berbagai fakta dan metodologi pengujian. Sebagai contoh, pemodelan ancaman fokus pada mengidentifikasi kelemahan keamanan selama desain, analisis kode aman dan ulasan fokus pada mengidentifikasi masalah keamanan dalam *source code* selama pengembangan, dan pengujian penetrasi berfokus pada pengidentifikasian kerentanan dalam aplikasi selama pengujian atau validasi.

Masalah keamanan yang diidentifikasi sejak awal pada SDLC dapat didokumentasikan dalam rencana pengujian sehingga kemudian dapat divalidasi dengan tes keamanan. Dengan menggabungkan hasil dari teknik pengujian yang berbeda, dimungkinkan untuk mendapatkan kasus uji keamanan yang lebih baik dan meningkatkan tingkat kepastian persyaratan keamanan. Misalnya, membedakan kerentanan

sebenarnya dari yang tidak dapat dieksploitasi adalah mungkin ketika hasil uji penetrasi dan analisis *source code* digabungkan.

- *Threats and Countermeasures Taxonomies*

Berdasarkan OWASP Testing Guide (The OWASP Foundation, 2014), ancaman dan cara untuk mengatasinya perlu diklasifikasikan. Selanjutnya perlu ditelisik apakah akar penyebab kerentanannya. Tahap ini adalah faktor penting dalam memverifikasi bagaimana kontrol keamanan dirancang, dikodekan, dan dibangun untuk mengurangi dampak pemaparan kerentanan tersebut. Dalam hal aplikasi web, pemaparan kontrol keamanan terhadap kerentanan umum dapat menjadi titik awal yang baik untuk mendapatkan persyaratan keamanan umum. Lebih khusus lagi, kerangka keamanan aplikasi web dapat memberikan klasifikasi terhadap kerentanan yang dapat didokumentasikan dalam pedoman dan standar yang berbeda dan divalidasi dengan tes keamanan.

Fokus dari mengklasifikasikan ancaman dan penanggulangannya adalah untuk menentukan persyaratan keamanan dalam hal ancaman dan akar penyebab kerentanan. Akar penyebab permasalahan keamanan dapat dikategorikan sebagai cacat keamanan dalam desain, bug keamanan dalam pengkodean, atau masalah karena konfigurasi yang tidak aman. Sebagai contoh, akar penyebab kelemahan otentikasi yang lemah mungkin adalah kurangnya autentikasi timbal balik ketika data melintasi batas kepercayaan antara klien dan server aplikasi. Persyaratan keamanan yang menangkap ancaman non-penolakan selama tinjauan desain arsitektur memungkinkan dokumentasi persyaratan untuk tindakan pencegahan yang dapat divalidasi nanti dengan uji keamanan.



Klasifikasi ancaman dan penanggulangannya terhadap kerentanan sebuah aplikasi juga dapat digunakan untuk mendokumentasikan persyaratan keamanan sebagai landasan dalam produksi yang aman,

misalnya standar pengodean yang aman bagi organisasi tersebut. Contoh kesalahan pengkodean umum dalam kontrol otentikasi terdiri dari penerapan fungsi *hash* untuk mengenkripsi kata sandi, tanpa menerapkan seed terhadap nilai. Dari perspektif secure coding, hal ini adalah sebuah celah yang memengaruhi enkripsi yang digunakan untuk otentikasi yang bermula pada kerentanan dalam kesalahan pengkodean. Karena penyebabnya adalah pengkodean yang tidak aman, persyaratan keamanan dapat didokumentasikan dalam standar pengkodean yang aman dan divalidasi melalui tinjauan secure coding selama fase pengembangan SDLC.

- *Security Testing and Risk Analysis*



Persyaratan keamanan perlu mempertimbangkan tingkat kerusakan yang mungkin ditimbulkan oleh suatu celah keamanan. Hal ini untuk mendukung strategi mitigasi risiko. Dengan asumsi bahwa organisasi yang bersangkutan menyimpan riwayat kerentanan yang

ditemukan dalam aplikasi, masalah keamanan dapat dilaporkan berdasarkan jenis, masalah, mitigasi, penyebab utama, dan dipetakan ke aplikasi dimana mereka ditemukan. Pengetahuan dasar terhadap celah keamanan seperti itu juga dapat digunakan untuk menetapkan metrik untuk menganalisis efektivitas tes keamanan di seluruh SDLC.

Sebagai contoh, pertimbangkan masalah validasi input, seperti *SQL Injection*, yang diidentifikasi melalui analisis *source code* dan dilaporkan dengan penyebab utama kesalahan pengkodean dan tipe kerentanan validasi input. Penjelasan mengenai celah keamanan seperti itu dapat dinilai melalui penetration testing, dengan memeriksa *form input* dengan beberapa vektor serangan injeksi SQL. Tes ini mungkin memvalidasi bahwa karakter khusus difilter sebelum sampai pada basis data dan mengurangi kerentanan. Hasil analisis *source code* dan pengujian penetrasi dapat digunakan untuk menentukan kemungkinan dan paparan kerentanan sekaligus menghitung peringkat

risiko kerentanan. Dengan melaporkan peringkat risiko kerentanan yang ditemukan, dimungkinkan untuk memutuskan strategi mitigasi yang paling sesuai untuk sebuah kasus tertentu. Sebagai contoh, kerentanan risiko tinggi dan menengah dapat diprioritaskan untuk remediasi, sementara risiko rendah dapat diperbaiki dalam rilis lebih lanjut.

***d) Security Testing Workflows***

Alur pengujian keamanan dapat diterapkan dalam proses pengembangan aplikasi dan proses pengujian tersendiri secara terpisah. Penjelasan mengenai keduanya menurut OWASP Testing Guide (The OWASP Foundation, 2014), adalah sebagai berikut:



- *Security Testing in the Development Workflow*

Pengujian keamanan selama fase pengembangan merupakan kesempatan pertama bagi pengembang untuk memastikan bahwa

komponen perangkat lunak yang mereka kembangkan diuji keamanannya sebelum diintegrasikan dengan komponen lain dan dimasukkan ke dalam aplikasi. Komponen perangkat lunak dapat terdiri dari artefak perangkat lunak seperti fungsi, metode, dan kelas, serta antarmuka pemrograman aplikasi, library, dan file yang dapat dieksekusi. Untuk pengujian keamanan, pengembang dapat mengandalkan hasil analisis *source code* untuk memverifikasi secara statis bahwa *source code* yang dikembangkan tidak mencakup kerentanan potensial dan sesuai dengan standar pengkodean yang aman. Tes unit keamanan selanjutnya dapat memverifikasi secara dinamis bahwa komponen berfungsi seperti yang diharapkan. Pengujian keamanan selama fase pengembangan SDLC merupakan kesempatan pertama bagi pengembang untuk memastikan bahwa komponen perangkat lunak individu yang mereka kembangkan diuji keamanannya sebelum diintegrasikan dengan komponen lain dan

dimasukkan ke dalam aplikasi. Komponen perangkat lunak dapat terdiri dari artefak perangkat lunak seperti fungsi, metode, dan kelas, serta antarmuka pemrograman aplikasi, perpustakaan, dan file yang dapat dieksekusi. Untuk pengujian keamanan, pengembang dapat mengandalkan hasil analisis *source code* untuk memverifikasi secara statis bahwa *source code* yang dikembangkan tidak mencakup kerentanan potensial dan sesuai dengan standar pengkodean yang aman. Tes unit keamanan selanjutnya dapat memverifikasi secara dinamis (mis., Pada saat run time) bahwa komponen berfungsi seperti yang diharapkan. Sebelum mengintegrasikan perubahan kode baru dan yang sudah ada dalam membangun aplikasi, hasil analisis statis dan dinamis harus ditinjau dan divalidasi.



Validasi *source code* sebelum integrasi dalam pembuatan aplikasi biasanya menjadi tanggung jawab pengembang senior. Pengembang senior lebih ahli mengenai masalah keamanan perangkat lunak.

Mereka harus membuat keputusan apakah akan menerima kode yang akan dirilis dalam pembuatan aplikasi atau untuk meminta perubahan dan pengujian lebih lanjut. Alur kerja peninjauan kode aman ini dapat ditegakkan melalui penerimaan formal serta pemeriksaan di alat manajemen alur kerja.

- *Security Testing in the Test Workflow*

Setelah perubahan komponen dan kode diuji oleh pengembang dan diperiksa untuk di-*deploy*, langkah berikutnya yang paling mungkin dalam alur kerja proses pengembangan perangkat lunak adalah untuk melakukan tes pada aplikasi secara keseluruhan. Level pengujian ini biasanya disebut sebagai tes terintegrasi dan uji level sistem. Ketika tes keamanan adalah bagian dari aktivitas pengujian ini, tes tersebut dapat digunakan untuk memvalidasi fungsionalitas keamanan aplikasi secara keseluruhan, serta paparan kerentanan tingkat aplikasi. Tes keamanan ini pada aplikasi mencakup pengujian *whitebox*, seperti analisis *source*

*code*, dan pengujian *blackbox*, seperti pengujian penetrasi. Pengujian *greybox* mirip dengan pengujian *blackbox*. Dalam pengujian *greybox*, diasumsikan bahwa *tester* memiliki pengetahuan parsial tentang manajemen *session* aplikasi, dan hal tersebut akan membantu dalam memahami apakah fungsi *logout* dan batas waktu diamankan dengan benar.

Target untuk tes keamanan adalah keseluruhan sistem yang akan berpotensi diserang, mencakup seluruh *source code* yang dapat dieksekusi. Salah satu ciri khas fase ini adalah bahwa mungkin bagi penguji keamanan untuk menentukan apakah kerentanan dapat dieksploitasi dan menguji aplikasi dengan memberikan resiko yang nyata. Hal ini termasuk juga kerentanan pada aplikasi berbasis web, serta masalah keamanan yang telah diidentifikasi sebelumnya di SDLC dengan kegiatan lain seperti pemodelan ancaman, analisis *source code*, dan tinjauan *secure code*.

Biasanya *pentester* melakukan tes keamanan ketika aplikasi berada dalam ruang lingkup untuk pengujian sistem integrasi. *Pentester* memiliki pengetahuan keamanan tentang kerentanan aplikasi web, teknik pengujian keamanan *blackbox* dan *whitebox*, dan memiliki validasi persyaratan keamanan. Untuk melakukan tes keamanan syarat yang harus dipenuhi adalah kasus uji keamanan didokumentasikan dalam pedoman dan prosedur pengujian keamanan.

Seorang *pentester* yang melakukan validasi keamanan aplikasi dalam lingkungan sistem terintegrasi dapat melepaskan aplikasi untuk pengujian di lingkungan operasional. Pada tahap validasi, pengujian fungsional aplikasi biasanya menjadi tanggung jawab penguji QA, sementara konsultan keamanan biasanya bertanggung jawab untuk pengujian keamanan. Beberapa organisasi mempercayakan proses pengujian kepada departemen IT yang menguasi penetration testing ketika pengujian oleh pihak ketiga dirasa tidak diperlukan.

Karena tes ini adalah upaya terakhir untuk memperbaiki kerentanan sebelum aplikasi dirilis ke produksi, setiap masalah yang ditemukan dalam proses pengujian harus segera dibenahi dan diselesaikan berdasarkan rekomendasi dari tim penguji. Rekomendasi dapat mencakup perubahan kode, desain, atau konfigurasi. Pada tingkat ini, auditor keamanan dan petugas keamanan informasi membahas masalah keamanan yang dilaporkan dan menganalisis risiko potensial sesuai dengan prosedur manajemen risiko informasi. Prosedur tersebut mungkin mengharuskan tim pengembangan untuk memperbaiki semua kerentanan risiko tinggi sebelum aplikasi dapat digunakan, kecuali risiko tersebut diakui dan diterima.



***e) Analysis and Reporting***

Menentukan tujuan dan pengukuran pengujian keamanan adalah prasyarat untuk menggunakan data pengujian keamanan sebagai bahan analisis risiko dan proses manajemen. Misalnya, pengukuran seperti

jumlah total kerentanan yang ditemukan dengan tes keamanan mungkin mengukur postur keamanan aplikasi. Pengukuran ini juga membantu mengidentifikasi tujuan keamanan untuk pengujian keamanan perangkat lunak. Misalnya, mengurangi jumlah kerentanan menjadi angka yang dapat diterima (minimum) sebelum aplikasi digunakan untuk produksi.

Berdasarkan OWASP Testing Guide (The OWASP Foundation, 2014), tujuan lain yang dapat dikelola adalah membandingkan postur keamanan aplikasi terhadap baseline untuk menilai peningkatan dalam proses keamanan aplikasi. Misalnya, garis dasar metrik keamanan mungkin terdiri dari aplikasi yang diuji hanya dengan penetration testing. Data keamanan yang diperoleh dari aplikasi yang juga diuji keamanan selama pengkodean harus menunjukkan perbaikan bila dibandingkan dengan baseline yang didapatkan sebelumnya.

Dalam pengujian perangkat lunak tradisional, jumlah cacat perangkat lunak, seperti bug yang ditemukan dalam suatu aplikasi, dapat memberikan ukuran kualitas perangkat lunak. Demikian pula, pengujian keamanan dapat memberikan ukuran keamanan perangkat lunak. Dari perspektif manajemen, kualitas perangkat lunak dan pengujian keamanan dapat menggunakan kategorisasi yang sama untuk menentukan penyebab utama dan upaya untuk memperbaiki kerentanan yang ditemukan. Dari perspektif akibat yang ditimbulkan, cacat keamanan dapat disebabkan oleh kesalahan dalam desain atau karena kesalahan dalam pengkodean. Dari perspektif upaya yang diperlukan untuk memperbaiki cacat, baik cacat keamanan dan kualitas dapat diukur dalam hal jam pengembang untuk menerapkan perbaikan, alat dan sumber daya yang diperlukan untuk memperbaiki, dan biaya untuk menerapkan perbaikan.

Perbandingan karakteristik data hasil pengujian dengan kualitas data merupakan kategorisasi dalam hal ancaman, paparan kerentanan, dan dampak potensial yang ditimbulkan oleh kerentanan untuk menentukan risiko. Aplikasi pengujian untuk keamanan terdiri dari mengelola risiko teknis untuk memastikan bahwa tindakan pencegahan aplikasi memenuhi tingkat yang dapat diterima. Untuk alasan ini, data pengujian keamanan perlu mendukung strategi risiko keamanan di bagian-bagian pemeriksaan yang kritis selama SDLC. Sebagai contoh, kerentanan yang ditemukan dalam *source code* dan analisa *source code* merepresentasikan ukuran resiko awal. Ukuran resiko kerentanan dapat diperhitungkan dengan melakukan beberapa faktor validasi dengan kerentanan yang ditemukan selama proses pengujian.



Ketika mengevaluasi keamanan suatu aplikasi, sangat penting untuk mempertimbangkan faktor-faktor tertentu. Faktor-faktor tersebut misalnya ukuran aplikasi yang sedang dikembangkan. Ukuran aplikasi

telah terbukti secara statistik terkait dengan jumlah masalah yang ditemukan dalam aplikasi selama pengujian. Salah satu parameter dari ukuran aplikasi adalah jumlah baris kode *(Line of Code)* aplikasi. Rata-rata terdapat kesalahan dalam kode perangkat lunak berkisar dari sekitar 7 hingga 10 kerusakan per seribu baris kode baru dan yang diubah. Untuk aplikasi yang memiliki jumlah baris lebih besar seharusnya intensitas proses pengujian lebih sering dan lebih lama. Hal ini karena kode yang di uji lebih banyak, memungkinkan adanya celah-celah pada baris kode yang lebih banyak pula.

Ketika pengujian keamanan dilakukan dalam beberapa fase SDLC, data uji dapat membuktikan kemampuan tes keamanan dalam mendeteksi kerentanan segera setelah diperkenalkan. Data uji juga dapat membuktikan keefektifan menghilangkan kerentanan dengan menerapkan langkah-langkah penanggulangan di berbagai titik pemeriksaan SDLC. Pengukuran tipe ini menyediakan ukuran

kemampuan penilaian keamanan yang dilakukan pada setiap fase proses pengembangan untuk menjaga keamanan dalam setiap fase dan ditulis dalam sebuah metrik. Metrik ini juga merupakan faktor penting dalam menurunkan biaya dalam memperbaiki kerentanan. Karena biaya yang dikeluarkan ketika celah ditemukan dalam proses pengembangan SDLC akan lebih murah dibandingkan ketika sudah dalam fase produksi.

Metrik uji keamanan dapat mendukung upaya menanggulangi risiko keamanan, meringankan biaya, dan analisis manajemen kerentanan ketika dikaitkan dengan tujuan dan berjangka waktu seperti mengurangi jumlah keseluruhan kerentanan hingga 30% dan memperbaiki masalah keamanan dengan tenggat waktu tertentu.



Data uji keamanan dapat bersifat absolut, seperti jumlah kerentanan yang terdeteksi selama proses *review* kode secara manual, serta secara komparatif, seperti jumlah kerentanan yang terdeteksi

dalam *review* kode yang dibandingkan dengan tes penetrasi. Untuk menjawab pertanyaan tentang kualitas proses keamanan, hal penting yang harus dilakukan adalah menentukan garis dasar untuk apa yang dapat dianggap dapat diterima dan baik. Data uji keamanan juga dapat mendukung tujuan spesifik dari analisis keamanan. Objek-objek ini dapat nilai konsistensi terhadap peraturan keamanan dan standar keamanan informasi, manajemen proses keamanan, identifikasi akar penyebab keamanan dan peningkatan proses pengembangan, dan analisis manfaat biaya dalam proses keamanan.

Ketika data uji keamanan dilaporkan, data tersebut harus menyertakan metrik untuk mendukung proses analisis. Ruang lingkup analisis adalah interpretasi data uji untuk menemukan petunjuk tentang keamanan perangkat lunak yang diproduksi serta efektivitas proses. Untuk membuat penilaian yang baik menggunakan data pengujian, hal penting yang harus dimiliki adalah pemahaman yang baik tentang

proses pengujian serta alat pengujian. Taksonomi alat harus diadopsi untuk memutuskan alat keamanan mana yang digunakan. Alat keamanan dapat dikualifikasikan sebagai ahli dalam menemukan kerentanan umum yang diketahui yang menargetkan artefak yang berbeda.

Hal lain yang menjadi permasalahan adalah *issue* atau *problem* keamanan tidak diketahui jika tidak dilakukan pengujian. Meskipun tidak ditemukan insiden keamanan terhadap suatu aplikasi, tidak ada jaminan bahwa aplikasi tersebut benar-benar bagus dapat dapat diandalkan. Diperlukan pengujian secara mendalam terhadap suatu aplikasi dengan berbagai macam metode untuk dapat menemukan kerentanan-kerentanan yang mungkin dimilik oleh sebuah aplikasi.



Meskipun telah menggunakan aplikasi pengujian keamanan yang dapat berjalan otomatis, namun alat-alat tersebut tidak akan mampu mengupas secara tuntas setiap kerentanan yang kemungkinan dimiliki.

Kebanyakan *security tester* yang sudah berpengalaman tidak cocok dengan hasil test yang dihasilkan oleh *software* pengujian otomatis. Hanya mengandalkan hasil dari *software* pengujian otomatis akan membuat prasangka yang tidak sesuai mengenai aplikasi dan kerentanan yang dimilikinya. Biasanya, semakin berpengalaman seorang *tester* dengan metodologi pengujian keamanan dan alat pengujian, hasil pengujian akan semakin baik pula. Dengan demikian setiap organisasi yang peduli terhadap keaamanan data terhadap aplikasi yang dimilikinya sebaiknya tidak sekedar berinvestasi dengan membeli *software* keamanan namun juga memperkerjakan orang yang secara fokus mempelajari keamanan dan dapat melakukan testing terhadap kemanan informasi. Selain hal tersebut, organisasi dapat mengirimkan staf keamanan pada pelatihan-pelatihan keamanan yang dapat meningkatkan skill dan pemahaman staff yang bersangkutan

dalam mengimplementasikan pengujian terhadap sistem yang dimiliki organisasi.

Bentuk keamanan suatu aplikasi dapat dipandang dari segi akibat yang mungkin akan ditimbulkan, seperti jumlah kerentanan dan tingkatan risiko kerentanan, serta dari perspektif penyebab, seperti kesalahan pengkodean, kelemahan arsitektur, dan masalah konfigurasi. Kerentanan dapat diklasifikasikan berdasarkan kriteria yang berbeda.

Beberpaa infiormasi yang sebaiknya disertakan dalam laporan uji kemanan menurut OWASP Testing Guide (The OWASP Foundation, 2014), diantaranya adalah sebai berikut:



- Kategorisasi dari kerentanan yang ditemukan berdasarkan jenisnya
- Ancaman keamanan yang ditimbulkan dari kerentanan yang ada

- Penyebab utama dari masalah keamanan yang ditemukan
- Teknik pengujian yang digunakan untuk menenemukan masalah-masalah keamanan yang dilaporkan
- Langkah-langkah penanggulangan yang sudah disiapkan atau yang mampu dilakukan untuk menanggulangi masalah keamanan yang ditemukan
- Tingkat keparahan dari kerentanan yang ditemukan

Dengan memberikan gambaran mengenai kerentanan-kerentanan yang ditemukan akan dapat ditemukan solusi terbaik untuk menanggulangi permasalahan yang ditemukan. Dan jika permalsahan tersebut belum dapat diatasi dengan sempurna, analisis mendalam mengenai kerentanan-kerentanan yang ditemukan dapat membantu untuk mencari solusi terbaik dan jawaban kenapa langkah yang dilakukan belum mampu untuk menanggulangi masalah tersebut.

Melaporkan akar penyebab masalah dapat membantu menentukan apa saja yang perlu diperbaiki. Dalam kasus pengujian *whitebox*, misalnya, akar masalah keamanan perangkat lunak dari kerentanan akan menjadi bagian dari source yang tidak diperkenankan. Setelah masalah dilaporkan, harus diberikan panduan kepada pengembang perangkat lunak tentang cara menguji ulang dan menemukan kerentanannya. Hal ini mungkin melibatkan penggunaan teknik pengujian *whitebox* (misalnya tinjauan kode keamanan dengan penganalisa kode statis) untuk menemukan apakah kode tersebut rentan atau tidak. Jika kerentanan dapat ditemukan melalui teknik *blackbox* (uji penetrasi), laporan pengujian juga perlu memberikan informasi tentang cara bagaimana kerentanan tersebut dapat ditemukan.



Informasi tentang bagaimana cara memperbaiki kerentanan harus dirasa cukup bagi pengembang untuk dapat menerapkan perbaikan.

Informasi ini harus memberikan contoh bagiamana pengkodean yang aman, perubahan konfigurasi, dan memberikan referensi yang memadai. Tingkat keparahan dari kerentanan yang ditemukan dapat berkontribusi terhadap perhitungan risiko dan membantu memprioritaskan upaya perbaikan. Biasanya, menetapkan peringkat risiko untuk kerentanan melibatkan analisis risiko eksternal berdasarkan faktor-faktor seperti dampak dan jangkauan paparan.

***f) Authentication Testing***

Menurut OWASP Testing Guide (The OWASP Foundation, 2014), otentikasi adalah tindakan menetapkan atau mengkonfirmasi sesuatu (atau seseorang) sebagai sesuatu yang otentik, yaitu, klaim yang dibuat oleh atau tentang hal tersebut adalah benar. Melakukan autentikasi terhadap suatu objek dapat berarti mengkonfirmasikan asalnya, sedangkan melakukan autentikasi terhadap seseorang sering kali terdiri

dari memverifikasi identitasnya. Otentikasi tergantung pada satu atau beberapa faktor otentikasi.

Dalam keamanan komputer, otentikasi adalah proses mencoba memverifikasi identitas digital pengirim dalam jalinan komunikasi. Contoh umum dari proses semacam itu adalah proses login. Menguji skema otentikasi berarti memahami bagaimana proses otentikasi bekerja dan menggunakan informasi itu untuk menghindari mekanisme otentikasi.

***g) Authorization Testing***

Menurut OWASP Testing Guide (The OWASP Foundation, 2014), *authorization* atau otorisasi adalah konsep yang memungkinkan akses ke sumber daya hanya kepada mereka yang diizinkan menggunakannya. Pengujian untuk Otorisasi berarti memahami

bagaimana proses otorisasi bekerja, dan menggunakan informasi itu untuk menghindari mekanisme otorisasi.

Otorisasi adalah proses yang muncul setelah otentikasi berhasil, sehingga *tester* akan memverifikasi titik ini setelah ia memegang kredensial yang valid, yang terkait dengan serangkaian peran dan hak istimewa yang ditetapkan dengan baik. Selama penilaian semacam ini, harus diverifikasi jika dimungkinkan untuk memotong skema otorisasi, menemukan kerentanan jalur traversal, atau menemukan cara untuk meningkatkan hak istimewa yang ditugaskan untuk *tester*.

***h) Session Manajemen testing***



Salah satu komponen inti dari aplikasi berbasis web apa pun adalah mekanisme yang digunakannya untuk mengontrol dan mempertahankan status bagi pengguna yang berinteraksi dengannya. Hal ini disebut sebagai *Session Management* dan didefinisikan sebagai

himpunan semua kontrol yang mengatur interaksi penuh-negara antara pengguna dan aplikasi berbasis web. Ini secara luas mencakup segala hal mulai dari bagaimana otentikasi pengguna dilakukan, hingga apa yang terjadi setelah mereka keluar.

HTTP adalah protokol *stateless*, artinya server web merespon permintaan klien tanpa menautkannya satu sama lain. Bahkan logika aplikasi sederhana membutuhkan beberapa permintaan pengguna untuk dikaitkan satu sama lain dalam *session*. Hal ini mengharuskan solusi pihak ketiga melalui solusi *middleware* dan server web *Off-The-Shelf* (OTS), atau implementasi pengembang yang dipersipkan lebih dahulu. Sebagian besar lingkungan aplikasi web yang populer, seperti ASP dan PHP, menyediakan pengembang dengan rutinitas penanganan sesi bawaan. Beberapa jenis perusahaan identifikasi biasanya akan dikeluarkan, yang akan disebut sebagai ID Sesi atau *Cookie*. Ada beberapa cara di mana aplikasi web dapat berinteraksi dengan

pengguna. Masing-masing tergantung pada sifat situs, keamanan, dan persyaratan ketersediaan aplikasi.

***i) Input Validation Testing***

Kelemahan keamanan aplikasi web yang paling umum adalah tidak dapat memvalidasi input dengan benar yang berasal dari klien atau dari pengguna sebelum menggunakannya. Kelemahan ini mengarah ke hampir semua kerentanan utama dalam aplikasi web, seperti *scripting*, *SQL Injection*, injeksi juru bahasa, serangan lokal / *Unicode*, serangan sistem file, dan *buffer overflows*.



Data dari entitas eksternal atau pengguna tidak boleh dipercaya, karena dapat dirusak oleh penyerang secara sewenang-wenang. *"All Input is Evil"*, kata Michael Howard dalam bukunya yang terkenal *"Writing Secure Code"*. Itu aturan nomor satu. Sayangnya, aplikasi yang kompleks seringkali memiliki banyak *form input*, yang

menyulitkan pengembang untuk membuat aturan ini berjalan secara utuh. Pada dasarnya akan lebih bagus ketika validasi input dicek secara menyeluruh saat proses pengembangan. Ketika proses pengembangan akan lebih mudah untuk memetakan setiap *form input* yang dibuat dan menerapkan sanitasi-sanitasi yang diperlukan sehingga akan lebih mudah dalam membuat filter terhadap inputan dari pengguna. Ketika aplikasi sudah jadi dan sudah di*deploy* ke produksi sementara belum dilakukan proses sanitasi untuk memvalidasi *form input*, *tester* akan kesulitan untuk mengetes semua form berdasarkan setiap kemungkinan input yang diperlukan dan pengembang juga akan kesulitan dikarenakan harus bekerja dua kali untuk memperbaiki *form input* yang telah di *deploy* sebelumnya.



**j) Teknik-teknik dalam *Software Testing***

Pada bagian ini akan diberikan tinjauan dari berbagai teknik pengujian yang dapat digunakan saat membangun program pengujian.

Bagian ini tidak memberikan informasi mendalam secara teknis mengenai teknik-tenknik yang disajikan, namun akan membahas gambaran umum dari berbagai macam teknik yang mungkin dapat digunakan dalam pengembangan perangkat pengujian berdasarkan OWASP Testing Guide. Teknik-teknik tersebut berdasarkan OWASP Testing Guide (The OWASP Foundation, 2014) adalah sebagai berikut:

- *Manual Inspections & Reviews*

*Manual Inspections* atau inspeksi manual adalah ulasan langsung dari seseorang yang biasanya menguji implikasi keamanan dari berbagai subjek seperti manusia, kebijakan, dan proses. Inpeksi manual juga dapat mencakup pemeriksaan keputusan teknologi seperti desain arsitektur. Proses ini biasanya dilakukan dengan menganalisis dokumentasi atau melakukan wawancara dengan desainer atau pemilik sistem.

Meskipun konsep inspeksi manual sederhana, teknik ini bisa menjadi salah satu teknik paling kuat dan efektif diantara teknik-teknik yang tersedia. Dengan bertanya kepada seseorang bagaimana sesuatu bekerja dan mengapa itu diterapkan dengan suatu cara tertentu, penguji dapat dengan cepat menentukan apakah ada masalah keamanan yang mungkin terjadi. Inspeksi dan ulasan manual adalah salah satu dari beberapa cara untuk menguji proses siklus hidup pengembangan perangkat lunak dan untuk memastikan bahwa ada kebijakan atau keterampilan yang memadai yang ditetapkan.

Seperti banyak hal dalam hidup, ketika melakukan inspeksi dan ulasan manual, direkomendasikan untuk mengadopsi model *trust-but-verified*. Tidak semua yang ditunjukkan atau diberitahu oleh penguji akan akurat. Ulasan manual sangat baik untuk menguji apakah orang memahami proses keamanan, telah dibuat sadar akan kebijakan, dan

memiliki keterampilan yang sesuai untuk merancang atau mengimplementasikan aplikasi yang aman.

Kegiatan lain termasuk meninjau dokumentasi, kebijakan pengkodean yang aman, persyaratan keamanan, dan desain arsitektur, semua harus diselesaikan dengan menggunakan inspeksi manual. Teknik ini memiliki kelebihan dan kekurangan sebagai berikut:

Kelebihan:

- ➔ Tidak memerlukan teknologi pendukung
- ➔ Dapat diaplikasikan dalam berbagai macam situasi
- ➔ Fleksibel
- ➔ Mendorong kerja tim
- ➔ Diaplikasikan di awal pengembangan

Kekurangan:

➔ Membutuhkan waktu yang lama

➔ Material pendukung tidak selalu tersedia

➔ Membutuhkan SDM dengan skill yang bagus untuk hasil yang maksimal.

- *Threat Modeling*

*Threat Modeling* atau pemodelan ancaman merupakan teknik yang populer dalam membantu perancang sistem berpikir tentang ancaman keamanan yang mungkin dihadapi sistem dan aplikasi mereka. Oleh karena itu, pemodelan ancaman dapat dilihat sebagai penilaian risiko untuk aplikasi. Bahkan, teknik ini memungkinkan perancang untuk mengembangkan strategi mitigasi untuk kerentanan potensial dan membantu mereka memusatkan sumber daya mereka yang terbatas dan

perhatian pada bagian-bagian sistem yang paling membutuhkannya. Disarankan bahwa semua aplikasi memiliki model ancaman yang dikembangkan dan didokumentasikan. Model ancaman harus dibuat sedini mungkin dalam SDLC, dan harus ditinjau kembali ketika aplikasi berkembang dan berkembang.

Untuk mengembangkan model ancaman, OWASP Testing Gude (The OWASP Foundation, 2014) menyarankan untuk mengambil pendekatan sederhana yang mengikuti standar NIST 800-30 untuk penilaian risiko. Pendekatan ini melibatkan:

- Penguraian aplikasi – menggunakan proses inspeksi manual untuk dapat memahami bagaimana sebuah aplikasi dapat bekerja, aset-aset pendukungnya, fungsi-fungsi yang tersedia, dan konektivitas dari aplikasi yang diuji.

- ➔ Mendefinisikan dan mengklasifikasikan aset – dengan mendefinisikan aset menjadi aset berwujud dan tidak berwujud dapat digunakan untuk mendukung kepentingan bisnis sebuah organisasi.

- ➔ Potensi kerentanan – baik teknis, operasional dan manajemen.

- ➔ Ancaman yang potensial – mengembangkan pandangan realistis mengenai potensi-potensi ancaman yang mungkin ditemui dari perspektif penyerang dengan menggunakan skenario penyerangan.

- ➔ Membuat strategi mitigasi – membuat strategi mitigasi untuk dapat mengontrol ancaman-ancaman yang realistis.



Output dari permodelan ancaman dapat bervariasi tetapi biasanya kumpulan daftar dan diagram. Tidak ada cara benar atau salah untuk mengembangkan model ancaman dan melakukan penilaian risiko

informasi pada aplikasi. Setiap cara mungkin dapat diambil manfaatnya untuk mendukung proses keamanan yang sedang dilakukan.

Menggunakan teknik permodelan ancaman juga memiliki kelebihan dan kekurangan, diantaranya adalah sebagai berikut:

Kelebihan:

- ➔ Dapat melihat sistem dari sudut pandang penyerang
- ➔ Fleksibel
- ➔ Dilakukan diawal proses pengembangan



Kekurangan:

- ➔ Bisa dibilang merupakan teknik yang baru dan minim resource untuk ditiru sebagai template.

➔ Permodelan ancaman yang bagus tidak selalu mengartikan aplikasi yang diuji selalu bagus.

- *Source Code Review*

*Source code review* atau ulasan *source code* adalah proses pengecekan secara manual *source code* suatu aplikasi untuk masalah keamanan. Banyak kerentanan keamanan yang serius tidak dapat dideteksi dengan bentuk analisis atau pengujian apa pun. Hampir semua pakar keamanan setuju bahwa tidak ada pengganti yang lebih baik dibandingkan dengan benar-benar melihat *source code* dari aplikasi. Semua informasi untuk mengidentifikasi masalah keamanan ada dalam kode di suatu tempat. Tidak seperti pengujian perangkat lunak tertutup pihak ketiga seperti sistem operasi, ketika menguji aplikasi web (terutama jika mereka telah dikembangkan sendiri) *source code* harus tersedia untuk tujuan pengujian.

Banyak masalah keamanan yang tidak disengaja tetapi signifikan juga sangat sulit ditemukan dengan bentuk analisis atau pengujian lain, seperti pengujian penetrasi. Dengan demikian teknik ini merupakan teknik yang sangat baik dan bagus sebagai pilihan pengujian keamanan. Dengan *source code review*, seorang *tester* dapat secara akurat menentukan apa yang sedang terjadi (atau yang seharusnya terjadi) dan menghapus pekerjaan menebak pengujian *blackbox*.

Metode ini meski pun bagus tetap memiliki kelebihan dan kekurangna. Diantara kelebihan dan kekurangan menggunakan teknik *source code review* adalah sebagai berikut:



Kelebihan:

- ➔ Pengujian dapat dilakukan secara utuh dan efektif
- ➔ Tingkat akurasi yang tinggi

➔ Cepat (jika dilakukan oleh orang yang berkompeten)

Kekurangan:

➔ Membutuhkan SDM (*security developer*) dengan skill yang tinggi

➔ Tidak dapat menemukan kerentanan pada library yang sudah di *compile*

➔ Tidak dapat mendeteksi *run-time error* dengan mudah

➔ Source code yang dianalisa mungkin ada perbedaan dengan *source code* yang diuji.



- *Penetration Testing*

*Pentration testing* telah menjadi teknik umum yang digunakan untuk menguji keamanan jaringan selama bertahun-tahun. Teknik ini

juga dikenal sebagai pengujian kotak hitam atau peretasan etis. Pengujian penetrasi pada dasarnya adalah seni pengujian aplikasi yang berjalan dari jarak jauh untuk menemukan kerentanan keamanan, tanpa mengetahui cara kerja aplikasi itu sendiri. Biasanya, tim uji atau *pentester* akan memiliki akses ke aplikasi seolah-olah mereka adalah pengguna. Penguji bertindak seperti penyerang dan berupaya menemukan dan mengeksploitasi kerentanan. Dalam banyak kasus, *tester* akan diberikan akun yang valid pada sistem.

Sementara pengujian penetrasi telah terbukti efektif dalam keamanan jaringan, teknik ini tidak serta merta bagus jika diaplikasikan untuk menguji sebuah aplikasi. Ketika pengujian penetrasi dilakukan pada jaringan dan sistem operasi, sebagian besar pekerjaan terlibat dalam menemukan dan kemudian mengeksploitasi kelemahan yang diketahui dalam teknologi tertentu. Karena aplikasi web hampir secara eksklusif dipesan lebih dahulu, pengujian penetrasi

di arena aplikasi web lebih mirip dengan penelitian murni. Alat pengujian penetrasi telah dikembangkan yang mengotomatiskan proses, tetapi dengan sifat aplikasi web efektivitasnya biasanya buruk.

Kelebihan dan kekurangn menggunakan penetration testing adalah sebagai berikut:

Kelebihan:

- ➔ Dapat dilakukan dengan cepat
- ➔ *Skill* yang dibutuhkan lebih rendah dibanding *source code review*
- ➔ Mengetes kode yang sesungguhnya yang telah di publikasikan



Kekurangan:

- ➔ Diaplikasikan setelah pengembangan

➔ Pengujian hanya berdasarkan dampak yang terlihat

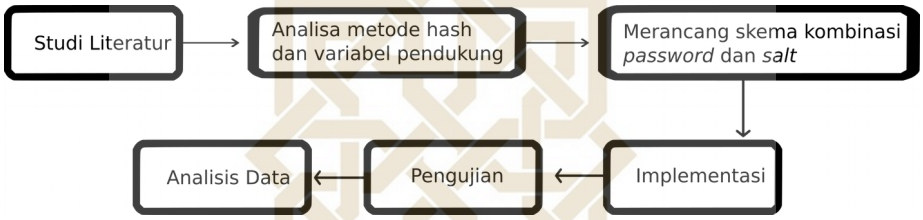
# BAB III

# METODE PENELITIAN

## A. Metode Penelitian

Metode penelitian adalah cara untuk mengumpulkan berbagai data yang akan diproses menjadi informasi. Informasi digunakan sebagai bahan untuk menyelesaikan permasalahan yang sedang diteliti. Langkah-langkah yang dilakukan dalam penelitian ini meliputi pengumpulan data, pengelompokkan data, proses analisis dan diakhiri dengan kesimpulan yang didapatkan.

Metode penelitian yang digunakan pada penelitian ini ditunjukan pada gambar 7:




Gambar 7: Metode Penelitian

1. Studi Literatur

Studi literatur dilakukan dengan cara mempelajari jurnal-jurnal/ penelitian terdahulu, buku-buku, internet dan sumber lain yang yang membahas mengenai keamanan sistem informasi, autentikasi, kriptografi, algoritma *hash*, dan topik-topik lain yang mendukung penelitian ini. Studi literatur juga merupakan proses

untuk menemukan teori-teori yang digunakan sebagai landasan dalam penelitian ini.

2. Analisa Metode *Hash* dan Variable Pendukung

Berbagai macam metode atau algoritma *hash* sudah banyak digunakan dalam pengaplikasian autentikasi pada sebuah sistem. Dari sekian banyak itu akan dipilih salah satu yang paling memungkinkan untuk digunakan sebagai objek penelitian. Parameter yang digunakan sebagai kriteria untuk menentukan algoritma *hash* yang dipilih adalah dari kesederhanaan algoritma tersebut. Dengan menggunakan algoritma *hash* yang sederhana, keterbatasan alat penelitian tetap akan mampu untuk menyelesaikan penelitian dengan baik.



Pada tahap ini algoritma *hash* akan ditentukan, selain dari tingkat kesederhanaannya, algoritma *hash* yang dipilih juga harus masih

banyak digunakan oleh banyak pengembang aplikasi. Meskipun tujuan penelitian ini adalah menemukan skema kombinasi yang dapat diaplikasikan pada berbagai macam algoritma, namun dengan menggunakan algoritma *hash* yang sederhana dan masih banyak digunakan, harapannya dapat membuka mata para pengembang yang masih menggunakan algoritma tersebut supaya sadar akan kebutuhan keamanan data, dan hasil dari penelitian dapat menjadi hasil yang aplikatif.

Dalam membuat sebuah skema penggunaan *password* dan *salt*, dibutuhkan variable-variable yang dapat mendukung tingkat keamanan sebuah kombinasi. Selain menentukan algoritma yang akan digunakan, pemetaan variable pendukung juga dilakukan untuk dapat digunakan sebagai bahan dalam merancang skema kombinasi *password* dan *salt* yang baik.

3. Merancang Skema Kombinasi *password* dan *Salt*

Perancangan skema kombinasi antara *password* dan *salt* digunakan sebagai bahan pengujian yang kemudian akan dibandingkan antara satu dengan yang lain. Bermodalkan algoritma dan variable pendukung yang telah ditentukan pada tahap sebelumnya, beberapa rancangan skema kombinasi akan ditentukan pada tahap ini.

Rancangan skema kombinasi tradisional dan rancangan skema kombinasi yang telah dipublikasikan oleh peneliti sebelumnya akan diikut sertakan dalam penelitian ini. Tujuan menyertakan skema kombinasi tradisional dan skema kombinasi peneliti sebelumnya adalah untuk mendapatkan gap atau jarak dari waktu yang dibutuhkan untuk melakukan cracking pada tahap pengujian. Dengan demikian, skema kombinasi tradisional dan skema

kombinasi yang telah dipublikasikan sebelumnya dapat digunakan sebagai pembanding dalam penelitian ini.

4. Implementasi

Pada tahap implementasi, setiap skema kombinasi akan diterapkan pada aplikasi berbasis web dengan bahasa pemrograman PHP. Aplikasi sederhana ini akan melakukan *generate salt*, melakukan proses *rearrangement*, melakukan proses *hashing*, dan menampilkan *output* yang berupa *chipertext*.

Pada dasarnya tujuan dibuat tahap implementasi disini adalah untuk mendapatkan data yang nantinya dapat digunakan sebagai bahan pengujian. Aplikasi yang dibuat bukan merupakan aplikasi yang dapat digunakan untuk keperluan tertentu seperti misalnya aplikasi *point-of-sale*, aplikasi *e-commerce* dan atau aplikasi semacamnya. Aplikasi yang dibuat sekedar aplikasi sederhana

yang didalamnya diimplementasikan skema-skema kombinasi yang telah dirancang, bukan sebagai bagian dalam proses autentikasi namun sekedar untuk mendapatkan nilai *chipertext* dari setiap skema yang telah dirancang pada tahap sebelumnya.

5. Melakukan Pengujian

Pengujian dilakukan terhadap beberapa skema yang telah ditentukan pada tahap sebelumnya. Pengujian dimaksudkan untuk mengetahui rentang waktu yang dibutuhkan untuk mengurai *chipertext* yang dihasilkan dari proses *hashing*. Semakin lama waktu yang diperlukan untuk menemukan nilai *password*, maka skema tersebut semakin baik dalam menguatkan kekuatan algoritma *hash*.

6. Analisis Data

Analisis data dilakukan untuk menarik kesimpulan dari hasil pengujian. Data yang didapat dari pengujian merupakan data yang belum informatif dan harus diolah agar menjadi data yang informatif.

Hasil dari proses pengujian merupakan nilai dari waktu yang dibutuhkan untuk mencari *plaintext* dari data *chipertext* yang sudah didapatkan dari tahap-tahap sebelumnya. Setiap nilai dari masing-masing proses akan diakumulasikan, dan dicari nilai rata-ratanya sehingga dapat dilihat selisih waktu dari setiap skema kombinasi. Hasil rata-rata dari setiap skema kombinasi akan ditampilkan dalam bentuk diagram sehingga menjadi lebih informatif dan dapat dibaca dengan lebih mudah.

**B. Alat Penelitian**

Alat penelitian merupakan perangkat komputer yang digunakan untuk melakukan proses penelitian. Pada penelitian ini, jenis dan spesifikasi perangkat komputer yang digunakan sangat berpengaruh terhadap nilai dan hasil penelitian. Semakin tinggi dan bagus spesifikasinya, akan semakin cepat melakukan proses pencocokan yang dilakukan pada tahap pengujian.

Alat penelitian yang digunakan pada penelitian ini adalah komputer laptop dengan spesifikasi sebagai berikut:



1. Merk/ Type : HP/ G 240 G7
2. Processor : Intel Core i7-8565U
3. RAM : 8 GB DDR 4

4. Storage : SSD 256 GB

5. Sistem Operasi : Linux Xubuntu 18.04

## C. Merancang Skema Kombinasi *Password* dan *Salt*

Beberapa rancangan skema kombinasi *password* dan *salt* diperlukan untuk dapat dilalukan perbandingan kekuatan dari setiap rancangan. Skema yang dibuat merupakan beberapa kombinasi dalam melakukan proses *hash* antara *password* dan *salt*. Skema yang akan digunakan pada penelitian ini dapat dilihat pada tabel 5 di bawah:

Tabel 5: Skema kombinasi *hashing password* dan *salt*



| No | Skema | Kombinasi |
|---|---|---|
| 1 | Skema 1 | Hash(*password*+salt) |
| 2 | Skema 2 | Hash(salt+*password*) |
| 3 | Skema 3 | Hash(**rearrangement**(*password*,salt)) |

| No | Skema | Kombinasi |
|---|---|---|
| 4 | Skema 4 | Hash(**rearrangement**(*password*,salt)+salt) |
| 5 | Skema 5 | Hash(*hash*(**rearrangement**(*password*,salt))) |

Pada tabel diatas telah ditampilkan 5 macam kombinasi antara *password* dan *salt* yang akan digunakan dalam penelitian ini. Skema 1 dan skema 2 merupakan skema tradisional yang sudah banyak digunakan oleh para pengembang aplikasi, yaitu dengan menjadikan salt sebagai prefix atau postfix dari sebuah *password*. Skema 3 merupakan skema yang pernah dipublikasikan oleh peneliti sebelumnya dan sangat berperan sebagai nilai pembanding terhadap skema baru yang ditambahkan. Skema 4 dan skema 5 adalah skema tambahan yang harapannya mendapatkan nilai waktu yang lebih tinggi dari skema-skema sebelumnya atau dalam arti lain menjadi skema kombinasi yang lebih kuat dan dapat direkomendasikan.

Fungsi *rearrangement* merupakan fungsi yang dibuat untuk mengacak komposisi antara *password* dan *salt*. Dengan menggunakan fungsi *rearrangement*, *salt* tidak diposisikan sebagai *prefix* atau pun *postfix*. Jika terdapat abcde sebagai *password*, dan 123 sebagai *salt*, dengan menggunakan fungsi *rearrangement* akan menghasilkan *string* baru berupa a1b2c3de, sehingga akan menyulitkan *attacker* untuk mendapatkan nilai *password* yang semakin sulit terbaca.

## D. Proses *Generating Salt*

Pada proses ini *salt* yang akan dipadukan dengan *password* dibuat. *Salt* dibuat dengan menggunakan fungsi *random* yang dimiliki oleh bahasa pemrograman PHP. *Salt* merupakan variabel pendukung yang akan diaplikasikan ke dalam sebagian dari skema kombinasi yang dibuat. *Salt* sangat berperan untuk membuat *hashed password* menjadi semakin kuat. Dengan menggunakan *salt* semakin unik dan semakin panjang, *password* yang sederhana akan menjadi lebih kuat.

Penggunaan salt dapat menambah jumlah karakter pada *string password* pada saat sebelum dilakukan proses *hashing*, tanpa menyulitkan user. Dengan adanya salt, user tidak perlu menghafal *password* yang panjang.

Pada penelitian ini *salt* yang di-*generate* dibatasi hanya 3 karakter angka untuk menyederhanakan proses pengujian. Meskipun demikian, dalam praktik pengembangan sistem seharusnya salt dibuat lebih dari 3 karakter sehingga dapat maksimal dalam menambah keamanan *password*.

## E. Proses Pengacakan *password* dan *Salt (Rearrangement)*



Proses ini merupakan proses yang akan dilakukan untuk memperkuat algoritma *hash*. Penggunaan *salt* pada umumnya hanya dikombinasikan sebagai *prefix* atau *postfix* terhadap *password*. Pada kasus lain penggunaan komposisi *salt* sangat bergantung kepada *programmer* yang membangun sebuah sistem. Dengan melakukan

pengacakan antara *password* dan *salt*, diharapkan sebuah *plaintext* yang akan dilakukan proses *hashing* akan menjadi lebih sulit untuk diuraikan oleh *attacker*, namun tanpa menambah karakter terlalu banyak sehingga tidak menambah waktu *hashing* menjadi lebih lama.

## F. Proses *Hashing*

Proses *hashing* merupakan proses dimana hasil kombinasi setiap skema akan diubah menjadi *chipertext*. Algoritma *hash* yang digunakan pada proses ini telah ditentukan sebelumnya pada tahapan analisa algoritma *hash*. Proses ini akan menggunakan algoritma MD5 sebagai objek pengujian dengan bahasa pemrograman PHP. Algoritma *hash* MD5 merupakan algoritma yang sudah tua namun masih banyak digunakan dalam pengembangan sistem. Banyaknya temuan celah keamanan pada algoritma *hash* menuntut pengembang sistem untuk dapat memanipulasi proses autentikasi sehingga dapat memperkuat keamanannya. Algoritma *hash* MD5 dipilih dalam penelitian ini

sebagai objek pengujian karena panjang bit-nya yang tidak terlalu panjang (32 bit) sehingga dapat diteliti dengan maksimal dengan perangkat komputer yang telah disiapkan untuk proses penelitian. Proses *hashing* akan menghasilkan *string* acak yang disebut dengan *chipertext*. *Chiptertext* hasil dari proses *hashing* kemudian akan digunakan sebagai bahan pengujian menggunakan *tools* oclHashcat pada tahap berikutnya.

**G. Pengujian**

Pengujian yang akan dilakukan dalam penelitian ini menggunakan suatu proses yang dapat menemukan *plaintext* dari *chipertext password*. Perlakuan untuk setiap skema akan berbeda berdasarkan kombinasinya masing-masing. Setiap kombinasi akan diuraikan, dan dicatat waktu yang diperlukan untuk dapat menguraikan *chipertext* dari *password* yang telah disimpan.

Proses penguraian (reverse rearrangement) pada tahap pengujian ini akan menggunakan algoritma sederhana yang dibuat dalam bahasa Python, sedangkan untuk proses penguraian *hash* akan digunakan sebuah *tools recovery password*, yaitu OclHashcat.

OclHascat merupakan program *recovery password* tercepat yang berbasis kepada GPU, dibangun oleh atom dan dapat berjalan pada GNU/Linux dan MS Windows, 32 dan 64 bit (Radix, 2011). Hashcat memiliki beberapa mode penyerangan yang digunakan untuk melakukan proses pencarian *plaintext*.



*Attack-Modes* atau mode penyerangan merupakan jenis serangan yang dapat dilakukan Hashcat dalam melakukan *pentration testing*. Untuk memilih mode yang akan digunakan, Hashcat menyediakan kode-kode untuk setiap modenya. Beberapa mode yang dapat dilakukan menggunakan Hashcat diantaranya:

a. *Straight / Dictionary Attack*

*Dictionary attack* atau sering disebut juga dengan *straight attack* dan *wordlist attack* merupakan mode penyerangan yang sederhana dengan memanfaatkan *wordlist*. *Wordlist* merupakan kumpulan dugaan *password* yang telah dihimpun dalam satu *list.*

Dalam mode ini, setiap kemungkinan akan dicoba satu demi satu dengan mencocokan hasil *hash* dari dugaan *password* dengan *chipertext* yang akan didekripsi. Penyerangan mode ini akan membutuhkan waktu lama, dan sangat bergantung kepada kelengkapan *wordlist*. Jika *password* yang dicari tersedia dalam *word*list maka proses dekripsi berhasil dan *password* ditemukan. Sebaliknya jika *password* yang dicari tidak tersedia

dalam *word*list, maka proses tidak berhasil dan *plaintext password* tidak akan ditemukan.

*b. Combination Attack*

*Combination attack* merupakan mode penyerangan Hashcat yang dapat membuat kombinasi-kombinasi dari beberapa kandidat *password* yang ada di *wordlist*. Sebagai contoh, pada Tabel 6, *wordlist* yang tersedia sebagai input dapat menghasilkan beragam kombinasi *password* yang dapat digunakan sebagai *wordlist* baru yang lebih banyak dan kombinatif.

Tabel 6: Contoh Combination-Attack

| Input | Output |
|---|---|
| pass | passpass |
| 12345 | pass12345 |
| omg | passomg |
| Test | passTest |
| | 12345pass |
| | 1234512345 |
| | 12345omg |
| | 12345Test |
| | omgpass |
| | omg12345 |
| | omgomg |
| | omgTest |
| | Testpass |
| | Test12345 |

| Input | Output |
| --- | --- |
| | `Testomg`<br>`TestTest` |

Seperti dapat dilihat pada tabel diatas, *output* merupakan hasil kombinasi dari *list password*, sehingga memiliki lebih banyak kandidat *password*. Dengan memiliki banyak kandidat *password* akan semakin meningkatkan kemungkinan *plaintext password* ditemukan, namun juga membutuhkan waktu yang semakin lama.



c. *Bruteforce – Mask Attack*

*Bruteforce* dan *Mask attack* pada dasarnya hampir sama, yaitu mencoba semua kombinasi dari *key* yang diberikan. Dalam

serangan *Brute-Force* tradisional dibutuhkan rangkaian karakter yang berisi semua huruf besar, semua huruf kecil dan semua digit *(mixalpha-numeric)*. Panjang kata sandi adalah 9, sehingga harus mengulangi kombinasi 62 ^ 9 (13.537.086.546.263.552), misalkan proses *cracking* dilakukan dengan kecepatan 100M / s, ini membutuhkan lebih dari 4 tahun untuk menyelesaikannya.

Dalam *Mask Attack* digunakan informasi terkait subjek dan dilakukan analisa bagaimana pengembang sistem tersebut mendesain *password*. Dengan menambahkan informasi-informasi personal sebagai tambahan *string* dalam *wordlist*, akan mempersingkat waktu yang dibutuhkan untuk mendekripsi *chipertext*-nya.

*d. Hybrid Attack*

Pada dasarnya serangan *hybrid* hanyalah serangan *combinator*. Satu sisi sebuah *dictionary list*, dan sisi lainnya adalah hasil dari serangan *Brute-Force*. Dengan kata lain, *keyspace Brute-Force* ditambahkan ke masing-masing kata dari kamus. Itu sebabnya ini disebut "*hybrid*".

*e. Rule-based Attack*

*Rule-based Attack* adalah salah satu jenis serangan yang paling rumit dari semua mode serangan. Mode ini seperti bahasa pemrograman yang dirancang untuk pembuatan kandidat *password*. Mode rule-based attack memiliki beragam fitur dan fungsi, diantaranya dapat digunakan untuk memodifikasi, memotong atau memperluas kata-kata dan memiliki operator

kondisional untuk melewati beberapa karakter yang tidak diinginkan dan berbagai macam fitur lain yang dapat didesain ketika melakukan serangan. Dengan menggunakan *rule-based attack* proses *cracking* menjadi fleksibel, akurat dan efisien.

**H. Analisis Data**

Analisis data merupakan proses mengolah dan menyajikan data yang telah dikumpulkan dari hasil pengujian ke dalam bentuk yang lebih informatif. Data didapatkan dari proses *cracking chipertext* hasil dari setiap skema yang telah ditentukan. Data-data tersebut masih berupa data mentah yang sukar untuk dipahami. Setiap skema kombinasi diberikan beberapa contoh *password* yang diambil dari tren *password* yang lemah namun masih banyak digunakan oleh sebagian orang. Dari hasil pengujian tersebut, akan dilakukan proses penjumlahan dan pembagian untuk mendapatkan nilai rata-rata waktu yang diperlukan dalam proses pengujian untuk masing-masing skema

kombinasi. Dengan demikian perbandingan antar skema kombinasi akan dapat dilihat dengan mudah. Hasil akhir akan disajikan dalam bentuk grafik sehingga menjadi lebih informatif.

**I. Penarikan Kesimpulan**

Kesimpulan diperlukan untuk menyampaikan hasil dari penelitian. Kesimpulan diambil berdasarkan hasil dari pengolahan data yang telah dilakukan dari proses sebelumnya. Pada kesimpulan dapat dilihat berapa selisih waktu yang dibutuhkan untuk mendapatkan *plaintext* dari *password*. Dari lama waktu tersebut, kekuatan dan keamanan algoritma *hash* akan dapat dilihat.

# BAB IV

# HASIL DAN PEMBAHASAN

Pada penelitian ini perancangan skema merupakan modal awal yang dibutuhkan setelah melakukan studi literatur. Skema kombinasi antara *password* dan *salt* yang dirancang selanjutnya akan dilakukan implementasi pada sebuah aplikasi web sederhana dengan bahasa pemrograman PHP untuk mendapatkan nilai dari *chipertext*. Nilai dari *chipertext* setiap skema kombinasi tersebut yang kemudian akan dilakukan dalam pengujian. Pengujian dilakukan untuk mendapatkan nilai dari waktu yang dibutuhkan dalam melakukan *cracking* untuk mendapatkan *plaintext* dari setiap *chipertext*.



## A. Rancangan Skema Kombinasi

Rancangan skema kombinasi diperlukan sebagai acuan dalam melakukan penelitian. Beberapa skema kombinasi yang disertakan dalam penelitian ini merupakan skema kombinasi tradisional yang telah banyak digunakan oleh para pengembang sistem. Salah satu

diantara skema kombinasi adalah skema kombinasi yang telah dipublikasikan oleh peneliti sebelumnya. Tujuan menyertakan skema kombinasi yang telah dipublikasikan oleh peneliti sebelumnya adalah untuk mendapatkan selisih waktu antara skema kombinasi terdahulu dengan skema kombinasi yang dirancang pada penelitian ini. Dengan demikian akan dapat terlihat apakah ada peningkatan keamanan dari skema kombinasi sebelumnya dengan skema kombinasi yang baru.

Skema yang akan digunakan pada penelitian ini dapat dilihat pada tabel 7 di bawah:

Tabel 7: Skema kombinasi *hashing password* dan *salt*



| No | Skema | Kombinasi |
|---|---|---|
| 1 | Skema 1 | Hash(*password*+salt) |
| 2 | Skema 2 | Hash(salt+*password*) |
| 3 | Skema 3 | Hash(**rearrangement**(*password*,salt)) |

| No | Skema | Kombinasi |
|---|---|---|
| 4 | Skema 4 | Hash(**rearrangement**(*password*,salt)salt) |
| 5 | Skema 5 | Hash(salt(**rearrangement**(*password*,salt))) |

Pada tabel diatas dapat dilihat ada 5 rancangan skema kombinasi yang akan digunakan dalam penlitian ini. Skema 1 dan skema 2 merupakan skema tradisional yang menggunakan salt sebagai prefix dan postfix terhadap *password* untuk meningkatkan keamanan algoritma *hash*. Skema 3 merupakan skema kombinasi yang telah dipublikasikan oleh peneliti sebelumnya sedangkan skema 4 dan 5 merupakan skema baru yang akan dibandingkan dengan skema-skema yang lain.



Sebagian dari skema kombinasi diatas menggunakan fungsi rearrangement. Fungsi *rearrangement* merupakan fungsi yang dibuat untuk mengacak komposisi antara *password* dan *salt*. Dengan

menggunakan fungsi *rearrangement*, *salt* tidak diposisikan sebagai *prefix* atau pun *postfix*. Jika terdapat abcde sebagai *password*, dan 123 sebagai *salt*, dengan menggunakan fungsi *rearrangement* akan menghasilkan *string* baru berupa a1b2c3de, sehingga akan menyulitkan *attacker* untuk mendapatkan nilai *password* yang semakin sulit terbaca.

**B. Implementasi**

Implementasi merupakan proses penerapan skema yang telah dirancang ke dalam sebuah aplikasi. Aplikasi yang dibuat pada bagian ini merupakan aplikasi sederhana yang dapat menampilkan nilai *chipertext* dari setiap skema kombinasi yang sudah dirancang. Dalam implementasi ini penerapan skema kombinasi bukan diimplementasikan dalam proses autentikasi, namun sekedar menerapkan skema kombinasi ke dalam fungsi bahasa pemrograman

sehingga didapat data berupa *chipertext* yang akan dijadikan bahan dalam proses pengujian.

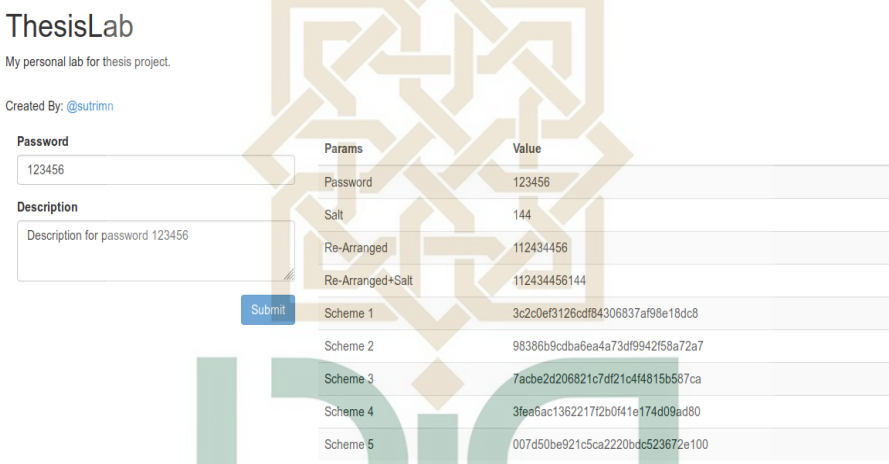
Pengimplementasian dilakukan dengan menggunakan framework CodeIgniter dan basis data MySQL. CodeIgniter merupakan framework yang sudah banyak digunakan dalam pengembangan system berbasis web yang menggunakan bahasa pemrograman PHP. Penggunaan basis data dalam implementasi sistem bertujuan untuk mendapatkan *record* dari setiap *result* yang dihasilkan selama proses pengumpulan data dengan aplikasi yang telah dibuat.

Pada dasarnya dalam proses ini bisa digunakan bahasa pemrograman apapun karena tujuan dari proses ini hanya untuk mengumpulkan data. Mengumpulkan data dengan melakukan proses *hashing* secara manual dengan menerapkan skema kombinasi yang telah dirancang pun memungkinkan untuk dilakukan. Namun dengan

membuat aplikasi sederhana semacam ini akan memudahkan proses pengumpulan data dan menghemat waktu penelitian.

**1. Tampilan Aplikasi**

Implementasi merupakan proses penerapan skema yang telah dirancang ke dalam sebuah aplikasi. Aplikasi yang dibuat adalah aplikasi sederhana yang dapat mensimulasikan penggunaan skema kombinasi dalam sebuah aplikasi berbasis website. Cara kerja dari aplikasi yang dibuat adalah menggunakan inputan berupa *password* untuk kemudian dikombinasikan dengan salt yang digenerate oleh fungsi yang dibuat didalam sistem. Kombinasi-kombinasi yang digunakan adalah skema kombinasi yang telah dirancang pada tahap sebelumnya. Hasil dari setiap proses berupa *chipertext* yang kemudian akan ditampilkan ke dalam view aplikasi dan disimpan ke dalam *database* sebagai catatan jika suatu saat dibutuhkan.




Gambar 8: Tampilan Aplikasi

Pada gambar diatas dapat dilihat pada sidebar terdapat *form input* yang digunakan untuk memasukkan *password*, dan form deskripsi untuk membantu menandai pengambilan data pada penelitian. Pada bagian sisi kanan merupakan result yang dihasilkan dari input yang diberikan. Result yang dihasilkan berupa nilai-nilai

dari skema kombinasi yang kemudian akan dilakukan proses pengujian.

**2. *Generating Salt***

*Salt* adalah serangkaian karakter yang dijadikan sebagai tambahan sebuah *password* atau *plaintext* sebelum dilakukan proses *hashing*. Pada penelitian ini *salt* yang digunakan diberikan batasan yaitu hanya menggunakan karakter berupa angka yang berjumlah 3 karakter. Pembatasan ini bertujuan untuk memudahkan proses pengujian pada tahap berikutnya. Dalam praktik yang sesungguhnya penggunaan *salt* yang sangat sedikit dan tidak menggunakan kombinasi berbagai macam jenis karakter sangat tidak disarankan, karena akan mempermudah *attacker* dalam memperoleh nilai *plaintext*.



Pada penelitian ini *salt* di-*generate* menggunakan fungsi mt_rand() yang sudah tersedia dalam bahasa pemrograman PHP.

Fungsi mr_rand() digunakan untuk menghasilkan nilai angka acak diantara *range* yang diberikan. Fungsi yang digunakan untuk melakukan *generate salt* secara utuh dapat dilihat pada kode program di bawah:

**Kode program 1** Fungi *Generate Salt*

```
1.  function randomSalt($length) {

2.    $result = '';
3.     for($i = 0; $i < $length; $i++) {
4.         $result .= mt_rand(0, 9);
5.     }
6.     return $result;
7.  }
```



Pada kode program diatas fungsi diberi nama randomSalt() dan menggunakan perulangan for. Perulangan dilakukan untuk mendapatkan sejumlah nilai acak yang dihasilkan oleh fungsi mt_rand(), karena fungsi mt_rand() hanya menghasilkan 1 nilai.

### 3. Fungsi *Re-Arrangement*

*Rearrangement* merupakan proses untuk menata ulang posisi *password* dan *salt* sebelum dilakukan proses *hashing*. Pada pembahasan sebelumnya, sudah ditampilkan 5 skema kombinasi yang akan digunakan. Dari 5 skema kombinasi yang telah dirancang, 3 diantaranya menggunakan fungsi *rearrangement*. Kode program untuk fungsi *rearrangement* dapat dilihat pada kode program di bawah:

**Kode program 2** Fungi *Re-Arrangement*

```
1.  function reArrange($arr1, $arr2, $n1, $n2)
2.          {
3.                  $i = 0;
4.                  $j = 0;
5.                  $k = 0;
6.                  $arr3 = array();
7.
8.                  while ($i < $n1 && $j < $n2)
9.                  {
10.                         $arr3[$k++] = $arr1[$i++];
11.                         $arr3[$k++] = $arr2[$j++];
12.                 }
13.
14.
```

```
15.                 while ($i < $n1)
16.                         $arr3[$k++] = $arr1[$i++];
17.
18.                 while($j < $n2)
19.                         $arr3[$k++] = $arr2[$j++];
20.
21.                 $result = array();
22.                 for ($i = 0; $i < ($n1 + $n2); $i++)
23.                         array_push($result, $arr3[$i]);
24.
25.                 return implode("", $result);
26.  }
```

Fungsi reArrangement() diatas dipanggil dengan menyertakan 4 parameter. Parameter pertama dan kedua, yaitu $arr1 dan $arr2 merupakan *password* dan *salt*. Parameter ketiga dan keempat yaitu $n1 dan $n2 merupakan panjang dari *password* dan *salt*. Fungsi diatas bekerja dengan melakukan penataan ulang antara *password* dan *salt*. Misalnya, *password* yang dimiliki adalah abcdef dan *salt* 123, maka akan menghasilkan *string* a1b2c3def.

**4. Pengambilan Data**

Pengambilan data merupakan proses mencatat setiap nilai *chipertext* dari setiap skema kombinasi untuk setiap *password* yang telah ditentukan. *password* yang digunakan merupakan *weak password* yang diambil dari *SplashData's Top 100 Worst Passwords*. Dari kumpulan *password* yang ditemukan, yang digunakan dalam penelitian ini adalah yang sesuai dengan kriteria yaitu memiliki panjang 6 karakter. *password-password* yang digunakan dapat dilihat pada Tabel 8.

Tabel 8: Daftar *password*

| No | *password* |
|---|---|
| 1 | qwerty |
| 2 | monkey |
| 3 | abc123 |
| 4 | 123123 |
| 5 | dragon |

| No | *password* |
|---|---|
| 6 | qazwsx |
| 7 | 654321 |
| 8 | harley |

*password-password* tersebut kemudian digunakan sebagai masukan pada aplikasi sederhana yang telah dibuat. Berdasarkan output pada aplikasi, nilai *chipertext* untuk setiap skema kombinasi dapat dilihat pada tabel berikut:

Tabel 9: Tabel *Chipertext* Skema 1

| No | *password* | Salt | Chipertext |
|---|---|---|---|
| 1 | qwerty | 857 | f1caf8486e08c76bd42ddd8ee44e83b9 |
| 2 | monkey | 047 | f0b5cb46f1907e2c985f50a011891bc6 |
| 3 | abc123 | 565 | 7626dc5a4220dc732c61dd8cd72b75c1 |
| 4 | 123123 | 345 | f8044a5b0001b54bc9c58c1f641ae602 |
| 5 | dragon | 372 | 24e9a05aca46ff892ad810da3cddf1dd |
| 6 | qazwsx | 512 | 2c22920f3b1dc8f2eeb94d0cee96c6cc |
| 7 | 654321 | 782 | 4717db7095815be80b926437b66ae525 |
| 8 | harley | 456 | 09e9b2fc5449d2ca3c6107a77afd123c |

| | | | |
|---|---|---|---|

Tabel 10: Tabel *Chipertext* Skema 2

| No | *password* | Salt | Chipertext |
|---|---|---|---|
| 1 | qwerty | 857 | 2468e99ed002a65b033658df50852773 |
| 2 | monkey | 047 | dd771c7788afd35942f53ff8c1639166 |
| 3 | abc123 | 565 | 5023e0ef99a83232d2cc92e32e5acf07 |
| 4 | 123123 | 345 | 600559e3d8647c07d1ded17b41248f26 |
| 5 | dragon | 372 | 58fc5d364aaeb87d48ded43223d82b59 |
| 6 | qazwsx | 512 | 91138c81309eca2b698b08e2f96cccb3 |
| 7 | 654321 | 782 | 642dcfcd1eb848e0d75eb9a7a341686c |
| 8 | harley | 456 | 99ed649624fb943fe4c1d1a4849198de |



Tabel 11: Tabel *Chipertext* Skema 3

| No | *password* | Salt | Chipertext |
|---|---|---|---|
| 1 | qwerty | 857 | 5849622db6b506045f99b270929173c3 |
| 2 | monkey | 047 | 22fcf0835244f5d7bf71cee1b36de521 |

| No | *password* | Salt | Chipertext |
|---|---|---|---|
| 3 | abc123 | 565 | f8157addbae5b93b596fdba183f0de05 |
| 4 | 123123 | 345 | b179683f87e4b59895f8b809e13f1ffb |
| 5 | dragon | 372 | f104f88fb1a7531b06ffaef925e5d88c |
| 6 | qazwsx | 512 | 51b82023add5c9bb2b32352c8b7d3ca3 |
| 7 | 654321 | 782 | cc77b32aa843fa8b9ef85879b8881a0e |
| 8 | harley | 456 | 99ed649624fb943fe4c1d1a4849198de |

Tabel 12: Tabel *Chipertext* Skema 4

| No | *password* | Salt | Chipertext |
|---|---|---|---|
| 1 | qwerty | 857 | c667ccfe6000426afb9df55532fcb46d |
| 2 | monkey | 047 | 0d6b9fd61f525a5096ea1d062717c288 |
| 3 | abc123 | 565 | dc6e2d8a0ecd6eb12109e58af265809f |
| 4 | 123123 | 345 | 4ae5ae0e347a0d4f581c21cb2ecea4af |
| 5 | dragon | 372 | e2c7b5827dead02e02f6b12423f6c2da |
| 6 | qazwsx | 512 | 45587ec8e2332dc00f3e53f40582593f |
| 7 | 654321 | 782 | b6f797aa9d9a589b1873a0ed6ecb12e0 |
| 8 | harley | 456 | 4e0725b0bbb9e979db234a33f247c64a |

Tabel 13: Tabel *Chipertext* Skema 5

| No | *password* | Salt | Chipertext |
|---|---|---|---|
| 1 | qwerty | 857 | 0c6e60f6177f69bd6c725d8196f1f066 |
| 2 | monkey | 047 | 3b59c1f66c3e08e0390e8202753fdf93 |
| 3 | abc123 | 565 | f1be04dfba462ac7051484bf793045c0 |
| 4 | 123123 | 345 | 0ce098029382c32a22d78ff6648acb0d |
| 5 | dragon | 372 | 665da07e3e89bc6c12ca36d8b29e3943 |
| 6 | qazwsx | 512 | 6b9303a2d72962931abb9fe547fa924e |
| 7 | 654321 | 782 | 7834eca175d5d7776bc12571f73da909 |
| 8 | harley | 456 | 19803f9c5d2d6af68b3af56ea653cf72 |

Nilai *chipertext* dari data-data yang telah ditampilkan pada tabel diatas, selanjutnya akan diuji untuk menilai kekuatan skema kombinasi. Pengujian dilakukan dengan mengukur waktu yang diperlukan untuk mendapatkan nilai *plaintext* untuk setiap *chipertext*.

## C. Pengujian

Setelah melakukan simulasi penggunaan skema kombinasi antara *password* dan *salt* pada tahapan sebelumnya, didapatkan data-data berupa *chipertext* untuk setiap kombinasi dan *password* yang telah ditentukan. Data-data tersebut belum dapat diolah menjadi informasi yang dapat dipahami dan menjawab tujuan dari penelitian.

Proses pengujian diperlukan untuk mendapatkan nilai berupa waktu yang diperlukan untuk dapat mengetahui *plaintext* dari *chipertext* yang telah didapatkan pada tahapan sebelumnya. Hasil dari proses pengujian ini yang kemudian akan diolah menjadi informasi yang dapat ditarik menjadi kesimpulan.



Pada pengujian ini, diasumsikan penguji yang bertindak sebagai *attacker* telah mengetahui model skema kombinasi dan panjang serta type data yang digunakan untuk *password* dan *salt* yang

digunakan sebagai bahan pengujian. Hal ini bertujuan untuk membatasi penelitian, karena dalam proses attacking sangat bergantung pada informasi dan analisa yang didapatkan oleh *attacker*. Dengan demikian, yang diuji pada penelitian ini benar-benar murni untuk mengukur pengaruh skema kombinasi pada kekuatan algoritma *hash*.

**1. Pengujian menggunakan Hashcat**

Hashcat merupakan *tools penetration testing* yang dapat digunakan untuk melakukan decrypt pada *chipertext* hasil dari proses *hashing*. Hashcat memiliki berbagai macam metode penyerangan dengan berbagai model *hash* yang akan di *decrypt*.



Beberapa model penuerangan yang dimiliki Hashcat diantaranya adalah:

a. *Straight / Dictionary Attack*

b. *Combination Attack*

c. *Bruteforce – Mask Attack*

d. *Hybrid Attack*

e. Rule-based Attack

Dalam tahap pengujian pada penelitian ini, yang akan digunakan adalah model penyerangan *Disctionary Attack* dan *Mask Attack*. Model penyerangan *Dictionary Attack* memanfaatkan *wordlist* yang merupakan file yang barisi banyak kumpulan kemungkinan *password*.



*Wordlist* dapat di unduh dari internet atau dapat dibuat sendiri dengan berdasarkan informasi yang dapat diperoleh dari target, atau biasa disebut Social Engineering. Pada penelitian ini *wordlist* yang

digunakan adalah Rockyou.txt yang dapat diunduh dari url https://www.scrapmaker.com/view/dictionaries/rockyou.txt .

Perintah yang digunakan untuk melakukan dekripsi dengan mode *Dictionary Attack* dengan menggunakan *wordlist* adalah sebagai berikut:

**hashcat -a 0 -m 0 example.hash example.dict**

Keterangan:

a) Hashcat : Unik key yang digunakan untuk memanggil aplikasi
b) -a : Attack Mode ( 0 untuk mendefinisikan mode penyerangannya straight)
c) -m : Hash Type (0 untuk mendefinisikan tipe *hash* target adalah MD5)
d) example.*hash* : *chipertext* yang menjadi target

e) example.dict : *Wordlist* yang digunakan

*Dictionary attack* memungkinkan untuk melakukan dekripsi dengan mencocokkan hasil *hash* dari kemungkinan *password* yang tersedia dalam *wordlist* dengan *chipertext* target, namun tidak dapat menebak *plaintext* yang memiliki tingkat kombinasi rumit.

Pada skema kombinasi 4 dan 5 telah menggunakan fungsi *rearrangement* yang mengacak susunan karakter *password* dan *salt*. Penggunaan fungsi *rearrangement* membuat susunan *plaintext* sebelum di *hash* menjadi tidak mendekati kata-kata umum yang biasanya tersedia di *wordlist*, sehingga perlu menggunakan mode penyerangan *Mask Attack*.



*Mask Attack* merupakan mode penyerangan yang menggunakan gabungan dari karakter-karakter untuk menebak *plaintext* yang dicari. *Mask Attack* pada Hashcat pada dasarnya mirip dengan *Bruteforce*

*Attack*. Pada mode *Bruteforce* tradisional dibutuhkan rangkaian karakter yang berisi semua huruf besar, semua huruf kecil dan semua digit *(mixalpha-numeric)*, sehingga membutuhkan waktu yang lama dan tidak efisien.

Pada serangan *Mask Attack* rangkaian karakter dapat disusun berdasarkan informasi-informasi tentang target yang telah didapatkan. Pada pengujian ini tidak semua karakter *mixalpha-numeric* digunakan untuk melakukan serangan, namun dapat disesuaikan dengan pola serangan yang diinginkan berdasarkan informasi yang telah didapatkan.



Salah satu contoh perintah yang digunakan pada penyerangan *Mask Attack* menggunakan Hashcat adalah sebagai berikut:

**hashcat -a 3 -m 0 scheme3_monkey.txt ?l?d?l?d?l?d?l?l?l --force**

Keterangan:

a) Hashcat : Unik key yang digunakan untuk memanggil aplikasi
b) -a : Attack Mode ( 3 untuk mendefinisikan mode penyerangannya Mask Attack)
c) -m : Hash Type (0 untuk mendefinisikan tipe *hash* target adalah MD5)
d) scheme3_monkey.txt : *Chipertext* yang menjadi target
e) ?l?d?l?d?l?d?l?l?l : Karakter yang digunakan untuk menentukan pola serangan, ?l untuk huruf a-z, ?d untuk angka 0-9.

**2. Proses *Reverse Rearrangement***



Tahap ini dilakukan khusus untuk skema 3, 4 dan 5 dimana pada skema tersebut menggunakan fungsi *rearrangement* sebelum *plaintext* diubah menjadi *chipertext* pada proses *hashing*. Pada tahap

ini, dibuat skrip khusus dengan bahasa python yang dapat dilihat pada skrip berikut:

**Kode program 3** Fungi *Reverse Rearrangement (.py)*

```
1.  def main():
2.      input_string = raw_input("Input your string: ")
3.      start_time = time.time()
4.      start_time_print = time.strftime('%d/%m/%Y %H:%M:%S')
5.      plaintext_length = 6
6.      salt_length = 3
7.
8.      string_to_list = list(input_string)
9.      list_of_result = []
10.     for x in xrange(0,len(input_string)):
11.       new_index = x
12.       if x != 1 and x != 3 and x != 5:
13.         list_of_result.insert(new_index,string_to_list[x])
14.     result = ''.join(list_of_result)
```

```
15.     print result
16.
17.     print("\n\n\n--- start time: %s" % start_time_print)
18.     print("--- stop time: %s" % time.strftime('%d/%m/%Y %H:%M:%S'))
1.      print("--- time estimated %s seconds" % (time.time() - start_time))
```

Pada fungsi diatas, *string* yang menjadi masukan merupakan *string* kombinasi *password* dan *salt*. Fungsi ini bekerja dengan melakukan perulangan dengan parameter paebanjang *password* dan *salt* yang sebelumnya telah diketahui. Didalam proses perulangan, karakter yang diidentifikasi sebagai bagian dari *password* akan disimpan dalam list baru dan ditampilkan sebagai result.

### 3. Hasil Pengujian

Tujuan utama dari proses pengujian adalah waktu yang diperlukan untuk mendapatkan *plaintext* untuk setiap skema. Hasil pengujian ini yang kemudian akan di lakukan proses analisa pada tahap berikutnya. Hasil pengujian ditampilkan pada tabel berikut:

Tabel 14: Tabel Hasil Pengujian Skema 1

| No | *password* | Salt | Time | | |
|---|---|---|---|---|---|
| | | | Cracking Hash | Reverse Rearrangement | Total |
| 1 | qwerty | 857 | 18s | - | 18s |
| 2 | monkey | 047 | 4s | - | 4s |
| 3 | abc123 | 565 | 3s | - | 3s |
| 4 | 123123 | 345 | 2s | - | 2s |
| 5 | dragon | 372 | 2s | - | 2s |
| 6 | qazwsx | 512 | 2s | - | 2s |
| 7 | 654321 | 782 | 3s | - | 3s |
| 8 | harley | 456 | 3s | - | 3s |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |

Tabel 15: Tabel Hasil Pengujian Skema 2

| No | *password* | Salt | Time | | |
|---|---|---|---|---|---|
| | | | Cracking Hash | Reverse Rearrangement | Total |
| 1 | qwerty | 857 | 16s | - | 16s |
| 2 | monkey | 047 | 3s | - | 3s |
| 3 | abc123 | 565 | 3s | - | 3s |
| 4 | 123123 | 345 | 2s | - | 2s |
| 5 | dragon | 372 | 3s | - | 3s |
| 6 | qazwsx | 512 | 3s | - | 3s |
| 7 | 654321 | 782 | 2s | - | 2s |
| 8 | harley | 456 | 2s | - | 2s |

Tabel 16: Tabel Hasil Pengujian Skema 3

| No | *password* | Salt | Time | | |
|---|---|---|---|---|---|
| | | | Cracking Hash | Reverse Rearrangement | Total |
| 1 | qwerty | 857 | 1h7m57s | 0.000262975692749s | 1h7m57.0003s |
| 2 | monkey | 047 | 1h4m57s | 0.00019907951355s | 1h4m57.0003s |
| 3 | abc123 | 565 | 7s | 0.000290155410767s | 7.0003s |
| 4 | 123123 | 345 | 5s | 0.000843048095703s | 5.0008s |
| 5 | dragon | 372 | 1m16s | 0.000288963317871s | 1m16.0003s |
| 6 | qazwsx | 512 | 3h35m17s | 0.000334024429321s | 3h35m17.0003s |
| 7 | 654321 | 782 | 8s | 0.000181913375854s | 8.0002s |
| 8 | harley | 456 | 1h6m14s | 0.000261068344116s | 1h6m14.0003s |

Tabel 17: Tabel Hasil Pengujian Skema 4

| No | *password* | Salt | Time | | |
|---|---|---|---|---|---|
| | | | Cracking Hash | Reverse Rearrangement | Total |
| 1 | qwerty | 857 | 2h12m36s | 0.000263929367065s | 2h12m36.0003s |
| 2 | monkey | 047 | 2h9m1s | 0.000308990478516s | 2h9m1.0003s |
| 3 | abc123 | 565 | 17s | 0.000260829925537s | 17.0003S |
| 4 | 123123 | 345 | 16s | 0.000319004058838s | 16.0003s |
| 5 | dragon | 372 | 12m33s | 0.000279903411865s | 12m33.0003s |
| 6 | qazwsx | 512 | 4h18m40s | 0.000226020812988s | 4h18m40.0002s |
| 7 | 654321 | 782 | 35s | 0.000211000442505s | 35.0002s |
| 8 | harley | 456 | 1h50m22s | 0.000138044357 3s | 1h50m22.0001s |

Tabel 18: Tabel Hasil Pengujian Skema 5

| No | *password* | Salt | Time | | |
|---|---|---|---|---|---|
| | | | Cracking Hash | Reverse Rearrangement | Total |
| 1 | qwerty | 857 | 1h48m39s | 0.000224113464355s | 1h48m39.0002s |
| 2 | monkey | 047 | 1h57m11s | 0.000231027603149s | 1h57m11.0002s |
| 3 | abc123 | 565 | 18s | 0.000243902206421s | 18.0002s |
| 4 | 123123 | 345 | 16s | 0.000253200531006s | 16.0002s |
| 5 | dragon | 372 | 12m18s | 0.000326156616211s | 12m18.0003s |
| 6 | qazwsx | 512 | 4h8m59s | 0.000308990478516s | 4h8m59.0003s |
| 7 | 654321 | 782 | 29s | 0.000277996063232s | 29.0003s |
| 8 | harley | 456 | 1h49m40s | 0.000302791595459s | 1h49m40.0003s |

## D. Analisis Data

Pada tahap ini hasil pengujian akan diolah menjadi lebih informatif. Sampel data yang berjumlah 8 untuk masing-masing skema kombinasi akan dicari nilai rata-ratanya, sehingga dapat dengan mudah dibandingkan dengan skema kombinasi yang lain.

### 1. Mencari nilai rata-rata setiap skema kombinasi

Sebelum dilakukan proses untuk menemukan nilai rata-rata, nila total waktu yang masih dalam bentuk jam, menit dan detik harus disamakan menjadi dalam bentuk detik/ secon. Rata-rata ditemukan dengan membagi jumlah nilai dari keseluruhan data pada satu skema kombinasi, terhadap jumlah data, atau jumlah sampel yang digunakan. Rumus yang digunakan sebagai berikut:

$$\bar{x} = \frac{\sum x}{N}$$

dimana,

$\bar{x}$ = Rata-rata

$\sum x$ = Jumlah nilai $x$

$N$ = Jumlah data

Perhitungan rata-rata pada skema 1 dan 2 akan menjadi seperti berikut:

Skema 1

$$Skema\ 1\ \bar{x} = \frac{(18+4+3+2+2+2+3+3)s}{8} = \frac{37\,s}{8} = 4.6\,s$$

Skema 2

$$Skema\ 2\ \bar{x} = \frac{(16+3+3+2+3+3+2+2)s}{8} = \frac{34\,s}{8} = 4.2\,s$$



Setelah setiap data yang didapatkan untuk masing-masing skema kombinasi ditemukan nilai rata-ratanya akan lebih mudah untuk

dapat melihat bagaimana perbedaan antara setiap skema kombinasi. Hasil nilai rata-rata untuk setiap skema dapat dilihat pada Tabel 19:

Tabel 19: Tabel Rata-rata Waktu Penyerangan Setiap Skema

| No | Skema | Rata-rata Waktu (s) |
|---|---|---|
| 1 | 1 | 4.6 |
| 2 | 2 | 4.2 |
| 3 | 3 | 3120.12 |
| 4 | 4 | 4832.50 |
| 5 | 5 | 4483.75 |

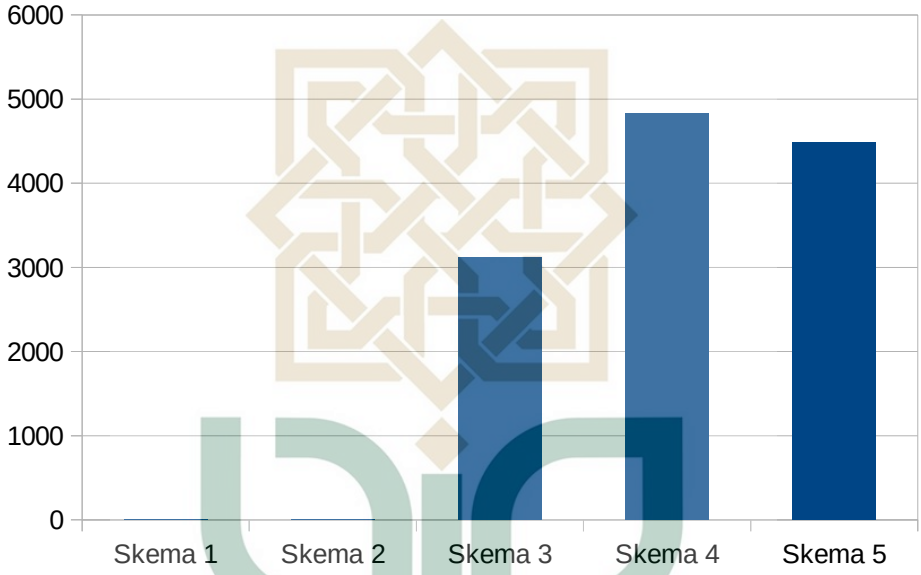
Pada tabel 19 dapat dilihat perbedaan waktu yang dibutuhkan untuk dapat menemukan *plaintext* dari setiap skema kombinasi. Pada skema 1 rata-rata waktu yang dibutuhkan adalah sebesar 4.6 detik dan pada skema 2 rata-rata waktu yang dibutuhkan adalah sebesar 4.2 detik. Jumlah waktu yang berbeda jauh dibutuhkan untuk menemukan *plaintext* dari skema 3, 4 dan 5. Skema 1 dan 2 merupakan skema

tradisional yang sudah sejak dulu digunakan oleh banyak pengembang aplikasi untuk mengamankan *password*. Hal tersebut tentu dimanfaatkan oleh *tools* semacam Hashcat untuk menciptakan formula khusus sehingga waktu yang diperlukan untuk menemukan *plaintext* dapat begitu cepat. Dengan demikian keberadaan skema kombinasi baru yang lebih kompleks diperlukan sehingga tidak mudah ditebak *plaintext* dari *password* yang digunakan.

**2. Grafik Rata-rata**

Pada Tabel 19 sudah dapat dilihat perbandingan rata-rata waktu yang diperlukan untuk mendapatkan *plaintext* dari setiap skema. Rata-rata waktu tersebut mengindikasikan seberapa kuat skema yang telah dirancang. Semakin tinggi waktu yang diperlukan, dapat diartikan semakin kuat pula skema tersebut.

Grafik rata-rata waktu dari Tabel 19 dapat dilihat di bawah ini:




Grafik 1: Rata-rata waktu pengujian (*s*)