

# Mekanisme/Metode Menjamin *Integrity* Pada File Digital dan Analisis Algoritmanya

Aaron Christopher Tanhar (07211940000055)  
Departemen Teknik Komputer  
Fakultas Teknologi Elektro dan Informatika Cerdas  
Institut Teknologi Sepuluh Nopember  
Surabaya, Indonesia 60111  
christopher.19072@mhs.its.ac.id

**Abstrak**—Integritas dari sebuah File Digital merupakan salah satu aspek penting dari sekuriti sistem komputer. Terdapat istilah yang dinamakan dengan File Integrity Monitoring (FIM) yang merupakan sebuah proses yang melakukan tindakan validasi dari sebuah File pada Operating System dan software aplikasi menggunakan metode verifikasi antara state atau keadaan yang terkini dengan state yang diketahui atau sebelumnya. Tentu saja hal tersebut penting untuk dilakukan sehingga file-file yang selalu kita transmisikan baik melalui lokal komputer dan melalui nirkabel akan selalu terjamin isinya sehingga tidak korup. Apabila tidak demikian maka mungkin dapat terjadi kerusakan atau pemalsuan data. Maka pada makalah ini saya melakukan riset terhadap metode yang dapat menjamin integritas pada file digital di sistem komputer. Dari segi security, terdapat beberapa penelitian yang juga melibatkan perkara *file integrity*. Penelitian dengan tema secure messaging juga memberikan kita penemuan yang relevan tentang usability dan security dari proses autentikasi pengguna layanan messaging tersebut. Parity Bit atau biasa juga disebut dengan *Check Bit* adalah bentuk sederhana dari *error detecting code*. Terdapat dua varian dari bit paritas, paritas genap dan paritas ganjil. Sebuah checksum merupakan suatu blok data berukuran kecil yang diperoleh dari blok data digital yang lainnya. Sebuah *MD5 message-digest algorithm* adalah hash function yang sering digunakan untuk mengecek integritas pada file. MD5 digest sudah digunakan secara luas pada dunia perangkat lunak untuk memberikan sebuah jaminan dimana file yang ditransmisikan telah tiba dan datanya sama dengan data yang asli. SHA-1 merupakan fungsi *hash cryptographic* yang menerima input lalu akan mengeluarkan output 160-bit (20 byte) nilai hash yang dikenal sebagai *message digest*. SHA-2 merupakan perubahan yang cukup signifikan dibandingkan pendahulunya, SHA-1. SHA-3 adalah subset dari Keccak yang didasari dari pendekatan baru yang dinamakan sponge construction. Sponge construction didasari dari fungsi random atau fungsi permutasi. Sebuah *Cyclic Redundancy Check* adalah *error-detecting code* yang biasa digunakan pada jaringan digital dan storage untuk mendeteksi adanya perubahan yang tidak diinginkan pada data. Komputasi dari CRC diturunkan dari polynomial division, modulo dua.

**Kata kunci**—*Integrity, Security, Komputer, File, MD5, Checksum, SHA, Digest, CRC*.

## I. PENDAHULUAN

Integritas dari File Digital merupakan salah satu aspek penting dari sekuriti sistem komputer. File adalah aspek penting dari sebuah OS karena digunakan untuk melakukan I/O serta interaksi user dengan OS [1]. Pada *UNIX-like OS* sendiri file adalah segalanya, dan segalanya adalah file, seperti yang di-

katakan oleh perintis Kernel Linux, Linus Torvalds [2]. Maka dari itu, file ini penting sekali untuk dilindungi untuk menjaga integritas dan *availability* dari *service* yang dilakukan oleh file yang bersangkutan tersebut. Menjaga integritas file penting dilakukan masa kini karena banyaknya instruksi dan data pada komputer. Terdapat beberapa faktor yang dapat menyebabkan terjadinya modifikasi data yang tidak terduga maupun yang tidak *authorized*. Sebuah data atau file dapat menjadi korup apabila terjadi malfungsi pada *hardware* atau *software*. Disk yang digunakan pada komputer bisa saja menjadi penyebab. Error pada disk merupakan sesuatu yang lumrah terjadi [3] dan biasanya *software* penyimpanan tidak didesain untuk menangani error tersebut. Bahkan dengan kesalahan integritas kecil sekalipun, yang tidak dapat dideteksi oleh *software* dengan tepat waktu, dapat membuat hilangnya data secara signifikan.

Terdapat beberapa cara atau metode untuk menjamin integritas dari sebuah file. *Parity Bit* atau biasa juga disebut dengan *Check Bit* adalah bentuk sederhana dari *error detecting code*. Error detecting code ini biasa digunakan untuk melakukan transmisi digital data yang *reliable* dengan menggunakan kanal komunikasi yang *unreliable*. Semua bentuk error detection akan menambahkan beberapa data ekstra pada pesan, yang dapat digunakan oleh penerima untuk melakukan verifikasi dari konsistensi pada pesan tersebut. Parity bit akan memastikan bahwa jumlah total dari 1-bit pada sebuah string adalah ganjil atau genap [4].

Terdapat beberapa perbedaan pada tiap-tiap implementasi algoritma untuk melakukan pengecekan integritas file. Pada masing-masing algoritma juga memiliki kekurangan dan kelebihan masing-masing, sehingga pemakaiannya juga perlu disesuaikan dengan kebutuhan. Maka dari itu penting untuk mengetahui perihal apa saja dan bagaimana karakteristik dari masing-masing algoritma untuk melakukan pengecekan integritas file ini. Ada algoritma yang dulunya cukup *reliable* untuk dijadikan sebagai lapisan pengamanan, namun seiring dengan makin cepatnya komputer, sudah dikatakan tidak aman lagi. Hal ini bukan berarti algoritma tersebut sudah tidak digunakan. Untuk persoalan mengecek integritas data masih cukup bisa diandalkan selama modifikasi data yang dilakukan tidak disengaja. Untuk aspek security, diperlukan juga pemahaman seberapa kuat algoritma tersebut dapat bertahan dari serangan *brute force*.

Pembahasan pada paper ini dimulai dengan presentasi mengenai penelitian lain (Bagian II). Kemudian dilanjutkan dengan penjelasan mengenai hal-hal apa saja yang menarik dari parity bit (Bagian III). Setelah itu dilanjutkan dengan pemahaman dari istilah checksum (Bagian IV). Pada bagian tersebut juga diklasifikasikan menjadi beberapa sub bagian seperti MD5, SHA-1, SHA-2 dan SHA-3. Dengan didasari bagian-bagian sebelumnya, maka dilanjutkan dengan Cyclic Redundancy Check (Bagian V) Terakhir, didapatkan kesimpulan dari penelitian yang telah dilakukan (Bagian VI).

## II. PENELITIAN TERKAIT

Dari segi security, terdapat beberapa penelitian yang juga melibatkan perkara *file integrity*. Beberapa diantaranya akan dibahas sekilas pada subbab berikut.

### A. File Integrity Verification

Beberapa penelitian dengan metode survei online telah mempelajari tentang security dan usability dari fingerprint berbeda untuk autentikasi maupun verifikasi integritas. Hsiao et al, sudah membandingkan kecepatan dan akurasi dari hash verification dengan representasi visual dan textual yang berbeda [5]. Pada penelitiannya disebut bahwa user kesulitan dalam membandingkan fingerprint yang panjang.

Penelitian dengan tema secure messaging juga memberikan kita penemuan yang relevan tentang usability dan security dari proses autentikasi pengguna layanan messaging tersebut. Unger et al, menekankan bahwa tingkat penggunaan dan usability yang terbatas dari verifikasi fingerprint secara manual [6]. Vaziripour et al, mengevaluasi usability dari proses autentikasi yang ada di tiga aplikasi messaging populer (WhatsApp, Viber, Facebook Messenger) melalui penelitian dua tahap yang melibatkan 36 pasang peserta [7]. Peserta melaporkan bahwa string fingerprint terlalu panjang, dan beberapa pengguna WhatsApp senang dengan fitur scan QR code daripada harus membandingkan string yang panjang.

### B. Otomatisasi Verifikasi Integritas

Kita dapat melakukan otomatisasi verifikasi checksum. Beberapa diantaranya sudah diaplikasikan. Pada package manager UNIX OS seperti brew (macOS) ataupun aptitude (Linux) telah diterapkan verifikasi checksum otomatis, yang mempermudah user untuk dapat langsung menginstall package tanpa perlu membandingkan checksum pada source dan yang diunduh secara manual karena pada package manager sudah otomatis melakukan hal tersebut. Akan tetapi, package manager seperti yang disebutkan sebelumnya biasanya hanya populer di UNIX system dan package manager biasanya digunakan untuk user yang sudah berpengalaman dengan terminal atau *command line interface*. Perlu diperhatikan bahwa package manager juga tidak luput dari serangan [8].

## III. PARITY BIT

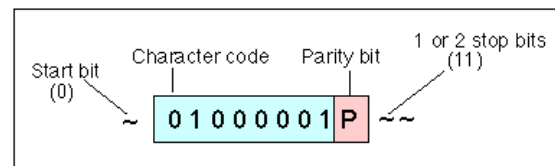
Pada ilmu Matematika, paritas merupakan sebuah properti dari sebuah *integer* yang menentukan apabila *integer* tersebut ganjil atau genap. Sebuah paritas dari bilangan bulat adalah

Tabel I  
CONTOH BIT PARITAS

7 bits of data	Count of 1 bits	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

genap apabila jika bilangan tersebut dapat dibagi dengan dua dan sisanya adalah nol. Sebaliknya jika sisanya satu maka paritasnya adalah ganjil. *Parity Bit* atau biasa juga disebut dengan *Check Bit* adalah bentuk sederhana dari *error detecting code*. Error detecting code ini biasa digunakan untuk melakukan transmisi digital data yang *reliable* dengan menggunakan kanal komunikasi yang *unreliable*. Semua bentuk error detection akan menambahkan beberapa data ekstra pada pesan, yang dapat digunakan oleh penerima untuk melakukan verifikasi dari konsistensi pada pesan tersebut. *Parity bit* akan memastikan bahwa jumlah total dari 1-bit pada sebuah string adalah ganjil atau genap [4].

Terdapat dua varian dari bit paritas, paritas genap dan paritas ganjil. Pada kasus paritas genap, untuk setiap bit dari string yang diberikan, jumlah bit yang bernilai 1 akan dihitung. Apabila jumlahnya ganjil, maka nilai dari bit paritas akan diatur menjadi 1, yang membuat total bit 1 menjadi bernilai genap. Sebaliknya, jika jumlah dari bit 1 pada awalnya sudah genap, maka bit paritas akan diatur menjadi 0. Pada kasus paritas ganjil, prosedurnya akan berkebalikan. Untuk setiap bit dari string yang diberikan, jumlah bit yang bernilai 1 akan dihitung. Apabila jumlahnya genap, maka nilai dari bit paritas akan diatur menjadi 1, yang membuat total bit 1 menjadi bernilai ganjil. Sebaliknya, jika jumlah dari bit 1 pada awalnya sudah ganjil, maka bit paritas akan diatur menjadi 0.



Gambar 1. Struktur parity bit

Apabila sebuah string yang berisi bit-bit paritas ditransmisikan secara tidak benar atau korup, maka bit paritasnya akan salah. Hal tersebut mengindikasikan terjadinya *parity error* saat transmisi. Bit paritas ini hanya bisa mendeteksi adanya error; tidak dapat melakukan koreksi error pada file yang terjadi, karena tidak ada mekanisme yang memungkinkan bit paritas untuk mengetahui bit yang mana yang korup. Lalu apabila meninjau Tabel I, dapat diketahui bahwa bit paritas ini memiliki kelemahan. Bit paritas hanya mengecek jumlah bit ganjil atau genap tanpa meninjau jumlah asli bitnya. Hal ini mengakibatkan terjadinya kegagalan dalam menangkap error pada data. Sebagai contoh, jika A ingin mengirim data 1001 dengan paritas genap maka bit paritasnya adalah 0 sehingga

yang ditransmisikan ke B adalah 10010. Terjadi error pada transmisi, sehingga yang sampai ke B adalah 11011. Akan tetapi, B menghitung jumlah bit dan mendapati genap, sehingga B akan menganggap data tersebut benar walaupun sebenarnya salah.

```
def even_parity(x):
    parity = 0
    while x:
        parity ^= x & 1
        x >>= 1
    return parity
```

Listing 1. Program perhitungan bit paritas genap.

```
def odd_parity(x):
    parity = 1
    while x:
        parity ^= x & 1
        x >>= 1
    return parity
```

Listing 2. Program perhitungan bit paritas ganjil

Potongan kode pada Listing 1 diatas adalah contoh kode yang berfungsi menemukan bit paritas genap. Untuk bit paritas ganjil tinggal mengganti inisialisasi variabel parity menjadi 1. Inputnya adalah *integer* yang bisa kita jadikan biner dulu, seperti 0b10001.

#### IV. CHECKSUM

Sebuah checksum merupakan suatu blok data berukuran kecil yang diperoleh dari blok data digital yang lainnya. Checksum memiliki tujuan untuk mendeteksi error yang mungkin saja terjadi ketika melakukan transmisi data pada memori maupun internet. Checksum biasanya digunakan untuk verifikasi *integrity* dari sebuah data, dengan kata lain apakah datanya sudah dimodifikasi setelah checksum dibuat. Output dari sebuah algoritma fungsi *hash cryptographic*, yang biasa disebut *hashes* atau *digest* digunakan sebagai checksum. Checksum biasanya direpresentasikan dengan menggunakan string hexadecimal (e.g., 69fac420b...), ukurannya berkisar antara 32 sampai 128 digit. Fungsi *hash cryptographic* ini memiliki tiga properti utama, yakni pre-image resistance, second pre-image resistance, dan collision resistance [9]. Fungsi *hash cryptographic* yang sering digunakan adalah MD5, SHA-1, dan SHA-2. MD5 merupakan salah satu fungsi *hash cryptographic* pertama yang dicanangkan. Namun pada akhir 1990an, algoritma tersebut mulai ditinggalkan karena berhasil dirusak sehingga tidak baik untuk keamanan. SHA-1 direkomendasikan oleh National Institute of Standards and Technology (NIST) sampai pada tahun 2015, algoritmanya terbobol. Sekarang, SHA-2 menjadi algoritma terpopuler yang direkomendasikan NIST untuk melakukan verifikasi integritas file [10].

##### A. MD5

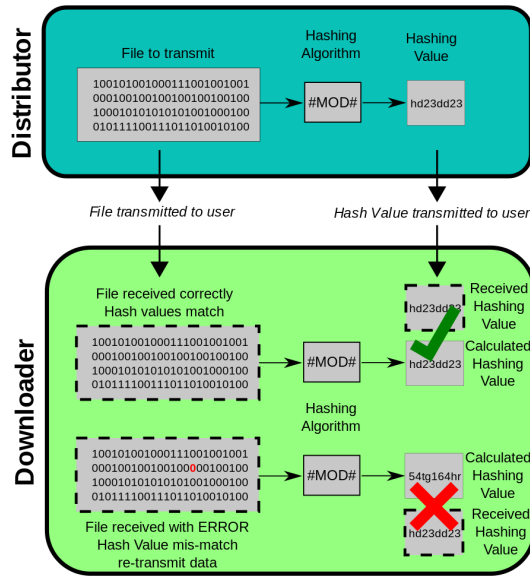
Sebuah *MD5 message-digest algorithm* adalah hash function yang *cryptographically broken* tapi masih sering digunakan untuk membuat nilai hash 128 bit. *Cryptographically broken*

maksudnya adalah secara security sudah tidak aman atau *vulnerable*. Pada awalnya, MD5 didesain sebagai fungsi *hash cryptographic*, namun ditemukan banyak vulnerability pada MD5, yang membuat algoritma ini ditinggalkan. MD5 masih bisa digunakan untuk melakukan verifikasi integritas data, namun hanya untuk mengecek kerusakan data yang tidak disengaja. MD5 masih cocok digunakan untuk hal-hal non kriptografik, seperti menentukan partisi dari kunci tertentu pada database yang sudah dipartisi, dan juga beberapa orang masih memilih menggunakan MD5 daripada *Secure Hash Algorithm* (SHA) karena beban komputasi yang lebih ringan [11]. MD5 didesain oleh Ronald Rivest tahun 1991 untuk menggantikan MD4, dan dispesifikasikan pada tahun 1992 sebagai RFC 1321. Salah satu kebutuhan dasar dari fungsi *hash cryptographic* adalah harus secara komputasional tidak mungkin ditemukan dua buah data yang berbeda dan memiliki nilai hash yang sama. MD5 gagal melakukan hal tersebut dan berdampak cukup fatal. Pada tahun 2019, MD5 tetaplah secara luas digunakan, meskipun kelemahan yang dimiliki sudah banyak didokumentasikan oleh para ahli security [12]. Sebuah *collision attack* dengan menggunakan komputer 2.6 GHz Pentium 4 dapat menemukan collision pada MD5 dalam hitungan detik. MD5 menggunakan *Merkle-Damgård construction*, jadi jika dua prefix dengan hash yang sama dapat dibuat, maka suffix umum dapat ditambahkan pada keduanya untuk membuat collision yang terjadi dapat diterima dan dianggap valid pada aplikasi yang menggunakannya.

MD5 digest sudah digunakan secara luas pada dunia perangkat lunak untuk memberikan sebuah jaminan dimana file yang ditransmisikan telah tiba dan datanya sama dengan data yang asli. Sebagai contoh, file pada server-server umumnya menyediakan MD5sum yang sudah dikomputasi untuk file tertentu, sehingga user yang mengunduh file tersebut dapat membandingkan apakah file yang diterima sama dan utuh dengan file asli yang terdapat pada server. Kebanyakan UNIX-based system menggunakan MD5sum pada package manager-nya. ROM pada Android juga menggunakan checksum jenis MD5 ini. MD5 juga biasa digunakan untuk hashing password satu arah, namun NIST tidak merekomendasikan MD5 untuk keperluan ini

MD5 akan memproses pesan yang memiliki panjang bervariasi menjadi output dengan panjang 128 bit. Pesan masukan dipecah menjadi block-block 512 bit, dan pada pesan dilakukan padding agar panjangnya dapat dibagi dengan 512. Cara paddingnya adalah: pertama, satu bit, 1 ditambah pada akhir pesan. Kemudian bit ini diikuti dengan nol sebanyak yang dibutuhkan untuk membuat panjang message kurang dari 64 bit dari kelipatan 512. Bit sisa yang panjangnya 64 bit diisi dengan 64 bit yang merepresentasikan panjang pesan mulanya, modulo 2.

Algoritma inti dari MD5 bekerja pada keadaan 128-bit, dibagi menjadi 32-bit words, dinotasikan dengan *A, B, C*, dan *D*. Variabel tersebut diinisialisasi ke konstanta tertentu. Algoritma inti akan menggunakan tiap blok pesan berukuran 512-bit untuk memodifikasi statenya. Pemrosesan dari blok pesan terdiri dari 4 tahap, termed rounds; tiap tahap terdiri dari 16



Gambar 2. File transmisi dengan hashing

operasi yang mirip berdasarkan fungsi linear  $F$ , penjumlahan modular, dan rotasi ke kiri. Berikut persamaannya. Terdapat 4 fungsi yang mungkin terjadi, fungsi berbeda digunakan untuk tiap tahapan atau round.

$$F(B, C, D) = (B \wedge C) \vee (\neg B \vee D) \quad (1)$$

$$G(B, C, D) = (B \wedge D) \vee (B \vee \neg D) \quad (2)$$

$$H(B, C, D) = B \oplus C \oplus D \quad (3)$$

$$I(B, C, D) = C \oplus (B \vee \neg D) \quad (4)$$

```
import hashlib

file_name = 'filename.exe'

original_md5 = '6941402abc4b2a76b9719d911420c592'

with open(file_name) as file_to_check:
    data = file_to_check.read().encode()
    md5_returned = hashlib.md5(data).hexdigest()

if original_md5 == md5_returned:
    print('MD5_verified.')
else:
    print('MD5_verification_failed!')
```

Listing 3. Program python sederhana untuk cek md5sum

Pada Listing 3 adalah contoh program python sederhana untuk melakukan checksum menggunakan MD5. Program tersebut menggunakan library bawaan bernama hashlib.

## B. SHA-1

SHA-1 merupakan fungsi *hash cryptographic* yang menerima input lalu akan mengeluarkan output 160-bit (20 byte) nilai hash yang dikenal sebagai *message digest*, yang biasanya adalah angka hexadesimal dengan panjang 40 digit. SHA-1 dibuat oleh NSA, badan keamanan Amerika. Sejak tahun 2005, SHA sudah tidak dianggap aman oleh badan-badan keamanan [13]. NIST secara formal tidak memperbolehkan penggunaan SHA pada tahun 2013. Penggantian SHA-1 terbilang urgent pada penggunaannya dalam digital signature seperti SSL certificate. Semua web browser populer tidak menerima sertifikat SSL SHA-1 pada tahun 2017 [14, 15]. SHA-1 memproduksi message digest dan prinsipnya mirip dengan MD5, namun membuat nilai hash yang lebih besar (160 bit vs 128 bit). Version control system seperti Git, Mercurial, dan Monotone menggunakan SHA-1 bukan untuk sekuriti, namun untuk pengecekan revisi dan untuk menjamin datanya tidak berubah karena korup yang tidak disengaja.

Untuk hash function yang mana  $L$  adalah jumlah bit di message digest, menemukan message yang sesuai dengan message digest dapat selalu dilakukan dengan metode *brute force* yang kira-kira dilakukan  $2^L$  iterasi. Ini dinamakan preimage attack dan mungkin bekerja mungkin juga tidak tergantung dari  $L$  dan kekuatan komputasi dari komputer. Akan tetapi, sebuah *collision*, yang terdiri dari menemukan dua pesan berbeda yang menghasilkan message digest yang sama memerlukan kurang lebih  $1.2 \times 2^{L/2}$  iterasi apabila menggunakan birthday attack. Beberapa aplikasi yang menggunakan cryptographic hashes, seperti penyimpanan password, tidak terlalu terdampak oleh collision attack. Membuat password yang bekerja untuk suatu akun membutuhkan preimage attack, dan juga akses ke hash pada password originalnya. Pada kasus tanda tangan dokumen, attacker tidak dapat langsung memalsukan signature dari dokumen yang sudah ada. Attacker harus membuat sepasang dokumen, satu yang tidak merusak, dan satu untuk merusak, dan membuat pemegang *private key* untuk menandatangani dokumen yang tidak merusak.

```
import hashlib

file_name = 'filename.exe'

original_sha1 = ('d1e67b8819b009ec6942033'
                 'b6fc1928dd64b5df31bcde63'
                 '81b9d3f90488d25324049046'
                 '0c0a5a1a873da8236c12ef969')

with open(file_name) as file_to_check:
    data = file_to_check.read().encode()
    sha1_returned = hashlib.sha1(data).hexdigest()

if original_sha1 == sha1_returned:
    print('SHA-1_verified.')
else:
    print('SHA-1_verification_failed!')
```

Listing 4. Program python sederhana untuk cek shasum

Pada Listing 4 adalah contoh program python sederhana untuk melakukan checksum menggunakan algoritma SHA-1. Program tersebut menggunakan library bawaan bernama hashlib.

### C. SHA-2

SHA-2 merupakan perubahan yang cukup signifikan dibandingkan pendahulunya, SHA-1. SHA-2 family terdiri atas enam buah hash function dengan digests (nilai hash) yang bernilai 2254, 256, 384 atau 512 bit: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. Hash function pada SHA-2 digunakan secara luas pada sejumlah aplikasi dan protokol sekuriti seperti TLS dan SSL, PGP, SSH, S/MIME dan IPsec. SHA-256 digunakan dalam proses autentikasi package software di OS Debian. SHA-256 dan SHA-512 direncanakan untuk digunakan pada DNSSEC [16]. Makin besar nilai hashnya maka artinya semakin aman dan sulit untuk menemukan collision, tetapi membutuhkan beban komputasi yang lebih tinggi.

### D. SHA-3

SHA-3 merupakan member terbaru dari family Secure Hash Algorithm (SHA), yang dirilis oleh NIST pada tahun 2015. Meskipun masih satu family, SHA-3 secara internal berbeda dengan SHA-1 dan SHA-2 yang strukturnya seperti MD5. SHA-3 merupakan subset dari *Keccak* [17].

Keccak didasari dari pendekatan baru yang dinamakan sponge construction. Sponge construction didasari dari fungsi random atau fungsi permutasi, dan menerima (atau "menyerap", dalam istilah sponge) input data sebanyak apapun, kemudian mengoutput ("memeras") berapapun jumlah datanya, sambil bertindak sebagai *pseudorandom function*. Pada state absorbing, blok pesan di-XOR menjadi subset dari state, yang kemudian ditransformasikan secara keseluruhan menggunakan fungsi permutasi  $f$ . Pada state "squeeze", blok-blok output dibaca dari subset yang sama dari state, berselang-seling dengan fungsi transformasi state  $f$ . Ukuran dari bagian pada state yang tertulis dan dibaca dinamakan "rate" (dinotasikan sebagai  $r$ ), dan ukuran dari bagian yang tidak tersentuh oleh input/output dinamakan capacity (dinotasikan sebagai  $c$ ). Kapasitas ini menentukan tingkat dari sekuritinya. Level maksimum dari sekuritinya adalah separuh dari kapasitasnya. Algoritma dari *Keccak* adalah sebagai berikut [18].

Fungsi theta terdiri dari tiga persamaan yang melibatkan operasi XOR sederhana dan bitwise cyclic shift.

$$C[X] = XOR(A[X, 0], A[X, 1], A[X, 2], A[X, 3], A[X, 4]) \quad (5)$$

*Circular shift* ke kiri pertama diaktifkan di lima jalur output. Setelah tahap ini, jalur terakhir menjadi yang pertama dan jalur kedua terakhir akan menjadi jalur terakhir. Kemudian circular shift ke kanan dilakukan pada jalur tersebut sehingga jalur pertama menjadi jalur terakhir dan jalur kedua menjadi jalur pertama. Sekarang circular shift ke kiri diterapkan pada setiap jalur untuk mengubah posisi dari bit pada tiap jalur [18, 19].

$$D[X] = ROT(C[X - 1], C[X + 1, 1]) \quad (6)$$

Input state matrix dan output jalur yang didapat dari (5)(6) di-XOR kan.

$$A[X, Y] = XOR(A[X, Y], D[X]), 0 \leq X; Y \leq 4 \quad (7)$$

Rho ( $\rho$ ) dan Pi ( $\Pi$ ) dapat diekspresikan untuk menghitung array B berukuran  $5 \times 5$  dari state array A. Operasi dari rho dan Pi, melakukan rotasi sirkular sebanyak nilai tetap yang bergantung pada koordinat  $X$  dan  $Y$ .

$$r[X, Y], B[Y, 2X + 3Y] = ROT(A[X, Y], r[X, Y]) \quad (8)$$

Jalur yang dirotasikan ditempatkan di array B. Ini disebut sebagai Pi step. Indeks yang diambil adalah modulo 5.

Jalurnya dioperasikan dengan fungsi chi ( $\chi$ ) seperti berikut.

$$A[X, Y] = B[X, Y]((\neg B[X + 1, Y]) \wedge B[X + 2, Y]) \quad (9)$$

Step Iota (10) adalah step paling sederhana dari algoritma Keccak. Step ini hanya melakukan operasi XOR pada konstanta bit RC dengan jalur pada lokasi  $[0, 0]$  dari matrix state baru A.

$$A[0, 0] = XOR(A[0, 0], RC) \quad (10)$$

```
import hashlib

file_name = 'filename.exe'

original_sha3 = ('3706a96a8fa96b3fc5ff30c'
                 'bca36ce666042e2d07762022'
                 'a78a2ec82439848fc3695e83'
                 '336ab71f47dddbc24b96454df2a43'
                 '7e343801a4e13faab89e8d0fda61')

with open(file_name) as file_to_check:
    data = file_to_check.read().encode()
    sha3_returned = hashlib.sha3_512(data).hexdigest()

if original_sha3 == sha3_returned:
    print('SHA-3_verified.')
else:
    print('SHA-3_verification_failed!')
```

Listing 5. Program python sederhana untuk cek sha3sum 512 bit

Program yang ada di listing 5 menunjukkan program python sederhana untuk melakukan pengecekan file integrity menggunakan algoritma SHA-3 512-bit.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consetetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet.

Aliquam non quam. Aliquam portitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

## V. CYCLIC REDUNDANCY CHECK

Sebuah *Cyclic Redundancy Check* adalah *error-detecting code* yang biasa digunakan pada jaringan digital dan storage untuk mendeteksi adanya perubahan yang tidak diinginkan pada data. Blok data yang memasuki sistem CRC akan diberikan *check value*, berdasarkan sisa dari pembagian polinomial dari kontennya. Saat pengambilan data, kalkulasi tersebut diulang lagi dan apabila *check value* tidak sesuai maka dapat dilakukan koreksi untuk menghindari data yang korup. CRC dapat digunakan untuk *error-correction* [20]. Varian CRC-1 dikenal juga sebagai parity bit (III).

Sejatinya, CRC merupakan tipe dari checksum, dan memiliki konsep yang mirip dengan checksum. Akan tetapi, terdapat perbedaan diantaranya sehingga saya memutuskan untuk memberikannya bab tersendiri. CRC merupakan checksum yang secara spesifik adalah *position dependent checksum algorithm*. Dari namanya tersebut, CRC dapat mendeteksi perpindahan posisi, yang membuatnya menjadi integrity check yang umum digunakan. CRC juga populer dikarenakan kesederhanaannya dibandingkan algoritma checksum yang lainnya seperti MD5 dan SHA family. CRC juga lebih mudah untuk dianalisis secara matematis dan baik untuk mendeteksi error yang umum terjadi dikarenakan oleh noise pada transmission channel. CRC sendiri tidak didesain dengan tujuan kriptografik, karena CRC dapat direverse sehingga untuk alasan keamanan lebih dianjurkan untuk menggunakan SHA-2. CRC biasa digunakan dalam hal untuk menyalin atau memindahkan file serta kompres dan dekompres file.

### A. Integritas Data

Seperti yang sudah disebutkan sebelumnya, CRC didesain secara spesifik untuk keperluan error-checking, dimana CRC akan mendeteksi kesalahan dengan beban komputasi yang jauh lebih ringan dibandingkan dengan Cryptographic Hash Function. Maka dari itu, CRC tidak cocok untuk melindungi dari modifikasi data yang disengaja. Yang pertama, karena tidak ada autentikasi, *attacker* dapat memodifikasi pesan dan menghitung ulang CRCnya tanpa terdeteksi. Ketika disimpan bersama dengan data, baik CRC maupun Cryptographic Hash Function tidak melindungi dari perubahan data yang disengaja. Aplikasi yang memerlukan proteksi dari serangan tersebut harus menggunakan mekanisme autentikasi kriptografik, seperti *message authentication codes* (MAC) atau *digital signatures*. Yang kedua, tidak seperti MD5 maupun SHA, CRC dapat dengan mudah direverse, yang membuat CRC tidak cocok untuk digunakan sebagai *digital signatures* [21]. Yang ketiga, CRC memiliki hubungan yang mirip dengan fungsi linear [22].

$$CRC(x \oplus y) = CRC(x) \oplus CRC(y) \oplus c \quad (11)$$

Dimana  $c$  bergantung dari panjang  $x$  dan  $y$ . Persamaan 11 juga bisa dituliskan seperti berikut, dimana  $x$ ,  $y$ , dan  $z$  memiliki panjang yang sama.

$$CRC(x \oplus y \oplus z) = CRC(x) \oplus CRC(y) \oplus CRC(z) \quad (12)$$

Maka, bahkan ketika CRC dienkripsi dengan *stream cipher* yang menggunakan XOR sebagai operasi kombinasinya, baik pesan maupun CRC dapat dimanipulasi tanpa sepengetahuan dari *encryption key*, ini merupakan salah satu *design flaws* dari protokol Wired Equivalent Privacy (WEP) [23].

### B. Algoritma

Komputasi dari CRC diturunkan dari polynomial division, modulo dua. Ini menyerupai pembagian dari pesan string biner, dengan nol bit yang jumlahnya tetap diappend oleh string "generator polynomial", tetapi dengan menggunakan XOR, bukan pengurangan. Pembagian jenis ini sudah direalisasikan di hardware dengan shift register yang sudah dimodifikasi [24]. Contoh dari implementasi polynomial division pada hardware, misalnya kita mencoba untuk menghitung CRC 8-bit dari pesan 8-bit yang terdiri dari karakter ASCII "W", yang binernya adalah  $01010111_2$ , desimal  $87_{10}$  atau hexadecimal  $57_{16}$ . Sebagai ilustrasi, kita menggunakan CRC-8-ATM (HEC) polinomial  $x^6 + x^2 + x + 1$ . Menuliskan bit pertama yang ditransmisikan (koefisien dari pangkat tertinggi  $x$ ) di sebelah kiri, sesuai dengan string 9-bit "100000111". Nilai byte  $57_{16}$  dapat dikirimkan dalam dua urutan berbeda, bergantung dari konvensi urutan bit yang digunakan. Tiap urutan menghasilkan pesan polinomial  $M(x)$  yang berbeda. Msbit-first,  $x^6 + x^4 + x^2 + x + 1 = 01010111$ , bila lsbit-first,  $x^7 + x^6 + x^5 + 3 + 1 = 11101010$ . Nilai ini dapat dikalikan dengan  $x^8$  untuk menghasilkan dua pesan 16-bit polinomial  $x^8 M(x)$ . Menghitung sisanya terdiri dari mengurangi kelipatan dari generator polinomial  $G(x)$ . Ini hanya seperti pembagian desimal, tapi lebih sederhana karena kelipatan yang mungkin hanyalah 0 dan 1, dan pengurangannya meminjam dari "tak terhingga", bukan mengurangi dari digit yang lebih tinggi. Karena kita tidak peduli dengan hasil bagi, maka tidak perlu dituliskan.

Perhatikan bahwa untuk setiap pengurangan, bit-bitnya dibagi menjadi 3 bagian, grup yang berisi nol, grup yang tidak diubah dari originalnya, dan grup yang berwarna biru, yang "menarik". Grup yang menarik ini panjangnya 8-bit, menyamai pangkat dari polinomial. Setiap langkah, kelipatan yang benar dikurangi untuk membuat grup nol menjadi satu bit lebih besar, dan grup yang tidak berubah menjadi satu bit lebih pendek, hingga akhirnya meninggalkan satu sisa.

```
def crc_remainder(input_bitstring,
                  polynomial_bitstring,
                  initial_filler):
    """Calculate the CRC remainder of a string of
    bits using a chosen polynomial.
    initial_filler should be '1' or '0'.
    """
```



Most-significant bit first	Least-significant bit first
0101011100000000	1110101000000000
-00000000	-100000111
=0101011100000000	=0110100110000000
-100000111	-100000111
=00010101011000000	=0010100001000000
-00000000	-100000111
=0001010101100000	=0000100010100000
-100000111	-00000000
=00000110101110000	=0000100010100000
-00000000	-100000111
=0000011010111000	=0000000100111000
-100000111	-00000000
=00000001010101100	=0000000100111000
-100000111	-00000000
=00000000010100010	=00000000100111000
-00000000	-00000000
=00000000010100010	=00000000100111000

Gambar 3. Perhitungan polynomial division

```

polynomial_bitstring = polynomial_bitstring.lstrip('0')
len_input = len(input_bitstring)
initial_padding = (len(polynomial_bitstring) - 1) \
    * initial_filler
input_padded_array = list(input_bitstring \
    + initial_padding)
while '1' in input_padded_array[:len_input]:
    cur_shift = input_padded_array.index('1')
    for i in range(len(polynomial_bitstring)):
        input_padded_array[cur_shift + i] \
            = str(
                int(polynomial_bitstring\
                    [i] != input_padded_array\
                        [cur_shift + i]))
return ''.join(input_padded_array)[len_input:]

def crc_check(input_bitstring,
    polynomial_bitstring,
    check_value):
    """Calculate the CRC check of a string of bits using a
    chosen polynomial."""
    polynomial_bitstring = polynomial_bitstring.lstrip('0')
    len_input = len(input_bitstring)
    initial_padding = check_value
    input_padded_array = \
        list(input_bitstring + initial_padding)
    while '1' in input_padded_array[:len_input]:
        cur_shift = input_padded_array.index('1')
        for i in range(len(polynomial_bitstring)):
            input_padded_array[cur_shift + i] \
                = str(int(polynomial_bitstring\
                    [i] != input_padded_array\
                        [cur_shift + i]))

```

```

return ('1' not in ''.join(input_padded_array)[len_input:])

```

Listing 6. Program CRC.

Program dari listing 6 berisi fungsi yang akan mereturn nilai sisa awal dari CRC untuk input dan polinomial yang ditentukan, entah dengan 1 atau 0 sebagai padding awalnya.

```

>>> crc_remainder('11010011101100', '1011', '0')
'100'
>>> crc_check('11010011101100', '1011', '100')
True

```

Listing 7. Output dari listing 6.

```

#include <inttypes.h> // uint32_t, uint8_t

uint32_t CRC32(const uint8_t data[], size_t data_length) {
    uint32_t crc32 = 0xFFFFFFFFu;

    for (size_t i = 0; i < data_length; i++) {
        const uint32_t lookupIndex =
            (crc32 ^ data[i]) & 0xff;
        crc32 =
            (crc32 >> 8) ^ CRCTable[lookupIndex];
    }

    // CRCTable is an array of 256 32-bit constants

    // Finalize the CRC-32 value
    // by inverting all the bits
    crc32 ^= 0xFFFFFFFFu;
    return crc32;
}

```

Listing 8. Program CRC dalam bahasa C.

Listing 8 merupakan algoritma CRC-32 [25] dalam bahasa C. Variable `CRCTable` adalah memoization dari kalkulasi yang harus diulang untuk setiap byte dari pesan.

### C. Kompresi Data

CRC digunakan untuk melakukan kalkulasi dari semua data yang ada di dalam file yang terkompresi. Nilai CRC akan dikalkulasi setiap kali ada file baru yang ditambahkan kedalam archive. Ketika archive atau file terkompresi di dekomposisi, maka program akan mengkalkulasi nilai CRC kembali dan membandingkannya dengan yang ada di archive. Apabila terdapat perbedaan pada CRC value, maka biasanya akan ditampilkan pesan CRC error, yang mengindikasikan bahwa file yang terekstrak tidak sama dengan file yang awalnya di-kompresi. Hal ini biasanya terjadi ketika file yang dikompresi didalam archive rusak. Hal ini juga dapat terjadi walaupun hanya satu file didalam archive yang korup.

Nilai dari CRC itu sendiri tidak mengatakan bahwa file anda korup atau tidak. Maka, ketika kita mengecek metadata dari archive atau mengkompresi file dan menemukan nilai CRC-nya, bukan berarti archive kita telah rusak. Hal tersebut hanya menunjukkan nilai awal dari CRC file yang terkompresi. Kebanyakan software untuk melakukan kompresi seperti 7zip dan WinRAR sudah memiliki mekanisme ini [26].

## VI. KESIMPULAN

Dari semua bagian yang sudah diberikan pembahasannya pada makalah ini, dapat diberi kesimpulan sebagai berikut. Parity Bit adalah salah satu cara untuk melakukan uji integritas file yang paling sederhana, tetapi pada penggunaannya terdapat kelemahan-kelemahan seperti jumlah bit yang sama-sama genap maupun ganjil tetapi berbeda dari yang aslinya akan dianggap tidak benar. Parity bit sendiri memiliki kelebihan seperti tidak memerlukan beban komputasi yang besar. Checksum sendiri merupakan istilah yang lebih luas dari algoritma-algoritma yang digunakan untuk melakukan pengecekan integritas. Terdapat banyak algoritma untuk melakukan checksum, salah satunya yang populer adalah MD5. MD5 merupakan algoritma hash function yang umum digunakan, meski sekarang hanya digunakan untuk sekedar melakukan cek integritas pada file yang tidak dimodifikasi secara sengaja. Hal ini terjadi karena komputer sekarang makin cepat sehingga metode brute force untuk menemukan collision dapat dengan mudah dilakukan. MD5 adalah varian dari message digest algorithm. SHA-1 merupakan generasi kedua dari keluarga SHA (Secure Hash Algorithm). SHA-1 juga sudah ditinggalkan karena alasan keamanan yang sama dengan MD5. SHA-2 merupakan hash function algorithm yang paling umum digunakan sekarang. NIST masih merekomendasikan SHA-2, namun disarankan menggunakan hash value yang besar agar semakin aman. SHA-2 juga dapat digunakan untuk melakukan pengecekan integritas file, namun apabila tidak memedulikan aspek sekuriti tidak disarankan karena beban komputasi yang lebih tinggi. SHA-3 merupakan terobosan terbaru dari keluarga SHA, karena menggunakan pendekatan yang berbeda dari varian sebelumnya, Keccak Algorithm. SHA-3 diyakini lebih aman dalam masalah anti collision, namun belum secara luas digunakan. CRC atau Cyclic Redundancy Check merupakan salah satu jenis checksum namun tidak diperuntukkan untuk keamanan. CRC bagus digunakan untuk melakukan verifikasi saat menyalin file ataupun pengarsipan file. Algoritma yang digunakan CRC juga terbilang tidak serumit MD5 dan SHA, sehingga pada *Integrated Circuit* biasanya sudah diaplikasikan dan siap digunakan.

## PUSTAKA

- [1] W. Stallings, *Operating Systems: Internals and Design Principles, Seventh Edition*. New York: Prentice Hall, 2012.
- [2] "Everything is a file descriptor or a process," 2015. [Online]. Available: [https://yarchive.net/comp/linux/everything\\_is\\_file.html](https://yarchive.net/comp/linux/everything_is_file.html)
- [3] V. Prabhakaran, N. Agrawal, L. N. Bairavasundaram, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Iron file systems," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, 2005.
- [4] W. H. RodgerE, Ziemer, Tranter, *Principles of communication : systems, modulation, and noise*. New Jersey: Hoboken, 2015.

- [5] H.-C. Hsiao, Y.-H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun, and B.-Y. Yang, "A study of user-friendly hash comparison schemes," in *2009 Annual Computer Security Applications Conference*, 2009, pp. 105–114.
- [6] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "Sok: Secure messaging," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 232–249.
- [7] E. Vaziripour, J. Wu, M. O'Neill, R. Clinton, J. Whitehead, S. Heidbrink, K. Seamons, and D. Zappala, "Is that you, alice? a usability study of the authentication ceremony of secure messaging applications," in *2017 ACM Symposium on Usable Privacy and Security (SOUPS)*, 2017.
- [8] J. Cappel, S. Samuel, S. Baker, and J. H. Hartman, "A look in the mirror: Attacks on package managers," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 565–574. [Online]. Available: <https://doi.org/10.1145/1455770.1455841>
- [9] B. Preneel, "Cryptographic hash functions," *Transactions on Emerging Telecommunications Technologies*, 1994.
- [10] "Nist policy on hash functions," 2015. [Online]. Available: <https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions>
- [11] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, 1st ed. O'Reilly Media, 2017.
- [12] "A quarter of major cmss use outdated md5 as the default password hashing scheme," 2019. [Online]. Available: <https://www.zdnet.com/article/a-quarter-of-major-cmss-use-outdated-md5-as-the-default-password-hashing-scheme/>
- [13] "Schneier on security: Cryptanalysis of sha-1," 2005. [Online]. Available: [https://www.schneier.com/blog/archives/2005/02/cryptanalysis\\_o.html](https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html)
- [14] "The end of sha-1 on the public web," 2019. [Online]. Available: <https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/>
- [15] "Google will drop sha-1 encryption from chrome by january 1, 2017," 2015. [Online]. Available: <https://venturebeat.com/2015/12/18/google-will-drop-sha-1-encryption-from-chrome-by-january-1-2017/>
- [16] "Use of sha-2 algorithms with rsa in dnskey and rrsig resource records for dnssec," 2009. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc5702.txt>
- [17] "Nist selects winner of secure hash algorithm (sha-3) competition," 2012. [Online]. Available: <https://www.nist.gov/itl/csd/sha-100212.cfm>
- [18] A. Arshad, D.-e.-S. Kundi, and A. Aziz, "Compact implementation of sha3-512 on fpga," in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, 2014.
- [19] N. Chandran and E. Manuel, "Performance analysis of modified sha-3," *Procedia Technology*, vol. 24, pp. 904–910, 12 2016.
- [20] "An algorithm for error correcting cyclic redundancy checks7," 2003. [Online]. Available: <https://www.drdoobs.com/an-algorithm-for-error-correcting-cyclic/184401662>
- [21] M. Stigge, H. Plotz, W. Muller, and J.-P. Redlich, "Reversing crc(theory and practice)," in *HU Berlin Public Report SAR-PR-2006-05*, 2006. [Online]. Available: [http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05\\_.pdf](http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05_.pdf)
- [22] "algorithm design - why is crc said to be linear?" 2003. [Online]. Available: <https://crypto.stackexchange.com/a/34013>
- [23] N. Cam-Winget, R. Housley, D. Wagner, and J. Walker, "Security flaws in 802.11 data link protocols," *Commun. ACM*, vol. 46, no. 5, p. 35–39, May 2003. [Online]. Available: <https://doi.org/10.1145/769800.769823>
- [24] E. Dubrova and S. S. Mansouri, "A bdd-based approach to constructing lfsrs for parallel crc encoding," in *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*, 2012, pp. 128–133.
- [25] "[lms-abs]: 32-bit crc algorithm | microsoft docs," 2019. [Online]. Available: <https://msdn.microsoft.com/en-us/library/dd905031.aspx>
- [26] "How to recover corrupted 7z archive." [Online]. Available: <https://www.7-zip.org/recover.html>