Implementasi Algoritma Checksum CRC32, MD5, dan SHA1 Untuk *File Sharing Verification*

Dery Rahman Ahaddienata

Teknik Informatika / Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia

13515097@std.stei.itb.ac.id

Abstak – Checksum merupakan sekumpulan bit yang merepresentasikan blok data digital untuk memastikan integritas dari sebuah data. Transmisi data pada jaringan sering kali menghasilkan error seperti perubahan posisi bits, hilangnya bits, ataupun duplikasi bits, sehingga perlu dilakukan mekanisme untuk memastikan integritas data seperti checksum. Terdapat beberapa algoritma checksum diantaranya seperti CRC32, MD5, dan SHA1. Ketiga algoritma ini akan digunakan untuk pembuatan API checksum sederhana untuk file sharing verification.

Kata kunci – checksum; CRC32; MD5; SHA1; file sharing verification;

I. Pendahuluan

Checksum sering sekali digunakan untuk keperluan verifikasi data yang telah diunduh dari internet. Ketika terjadi error dalam melakukan transmisi, data bisa saja mengalami *corrupt* sehingga data yang telah diunduh tidak dapat dibuka secara normal. Selain kegagalan transmisi, masalah pada penyimpanan juga dapat menjadikan data *corrupt*. Untuk mendeteksi adanya *corruption* dapat dilakukan dengan menggunakan *file verification*, yaitu dengan membandingkan 2 file (asli dan hasil download) pada level bit. Namun, untuk mendapatkan file asli, bisa saja juga mengalami *corrupt*. Pendekatan yang paling sering digunakan yaitu dengan menghitung checksum dan membandingkan checksum antara 2 file.

Pada dasarnya fungsi utama checksum adalah untuk mendeteksi adanya *random corruption*, yaitu perubahan bit yang disebabkan oleh error pada transmisi ataupun media penyimpanan. Perubahan bit yang sengaja dilakukan untuk membuat malicious file tidak menjadi prioritas utama fungsi checksum sehingga algoritma klasik checksum seperti CRC32, MD5, maupun SHA1 dapat digunakan untuk keperluan deteksi *random corruption*. Ketiga algoritma ini masih sering digunakan karena kecepatannya dalam melakukan perhitungan checksum.

CRC32 merupakan *error detection* yang sering digunakan untuk mendeteksi *noise* pada *channel* transmisi. CRC32 menghitung sisa pembagian polinomial dari sebuah bits data. Polinomial yang digunakan untuk CRC32 menggunakan 33 bits data yang digunakan pada perintah cksum Unix. MD5 dan SHA1 merupakan algoritma untuk fungsi hash yang dapat digunakan untuk keperluan verifikasi integritas data. Meskipun MD5 maupun SHA1 mempunyai celah keamanan karena adanya *hash kolisi*, namun kedua algoritma ini masih sering digunakan untuk mendeteksi adanya *random corruption* karena kecepatannya dalam menghitung checksum.

Beberapa *file sharing application* gratis seperti 4shared, tinyupload, dan uploadfiles tidak menyediakan fitur checksum pada setiap file yang diupload, sehingga integritas sebuah data yang didownload dari situs tersebut perlu dipertanyakan. Dengan adanya fungsi checksum CRC32, MD5, maupun SHA1, *file sharing application* dapat memanfaatkan mekanisme checksum untuk digunakan sebagai verifikasi integritas data.

II. Dasar Teori

Bab ini akan membahas mengenai dasar teori yang menunjang pembuatan ketiga macam algoritma checksum, CRC32, MD5, dan SHA1. Dasar teori yang dibahas adalah algoritma CRC32, fungsi hash satu arah, algoritma MD5, dan algoritma SHA1.

A. Algoritma CRC32

CRC32 merupakan salah satu varian CRC (Cyclic Redundancy Check) dengan menggunakan polinomial 33bit, yaitu 0xEDB88320. Sebuah algoritma CRC-n, mempunyai sisa pembagial n bits, sehingga polinomial yang dipilih

mempunyai panjang n+1 bits. Sebagai contoh, untuk CRC-4 yang mempunyai polinomial 0x9, dapat dinyatakan sebagai, $x^4 + x^1 + 1$, atau 10011. Polinomial ini dipilih untuk memaksimalkan deteksi error akibat adanya perubahan bit yang kecil, dan meminimalkan probabilitas *kolisi*. CRC32 yang digunakan pada cksum UNIX menggunakan polinomial 0xEDB88320.

Secara sederhana CRC menghitung checksum yang didapat dari hasil sisa pembagian data (divident) terhadap polinomialnya (divisor). Operasi dilakukan dengan menggunakan XOR. Contoh untuk 14 bits data (11010011101100) dengan menggunakan CRC-3, menggunakan polinomial 4 bits (1011), operasi yang dilakukan adalah sebagai berikut.

1. Tambahkan padding pada divident. Untuk polinomial n+1 bits, padding yang ditambahkan sebanyak n bits.
2. Pada setiap step, nilai 1 pertama pada divisor selalu berada diposisi 1 pertama pada divident. Lakukan operasi xor untuk menghasilkan divident yang baru.
3. Operasi dilakukan secara berulang kali, hingga divident bernilai 0.

Contoh operasi

```
11010011101100 -> 11010011101100000
```

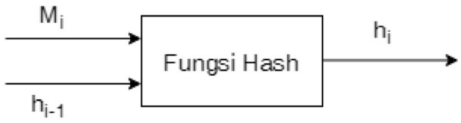
```
11010011101100 000
1011
-----------------
01100011101100 000
 1011
-----------------
00111011101100 000
  1011
-----------------
00010111101100 000
   1011
-----------------
00000011101100 000
       1011
-----------------
00000000110100 000
        1011
-----------------
00000000011000 000
         1011
-----------------
00000000001110 000
          1011
-----------------
00000000000101 000
           101 1
-----------------
00000000000000 100
```

```
Nilai 100 merupakan crc yang digunakan untuk
checksum
```

B. Fungsi Hash Satu Arah

Fungsi hash merupakan fungsi yang menerima masukan bytes dengan panjang sembarang lalu mentransformasikannya menjadi bytes dengan panjang yang tetap. Fungsi hash mempunyai sifat satu arah, yaitu pesan yang sudah dilakukan operasi hash tidak dapat dikembalikan menjadi pesan semula. Untuk setiap masukan x, sebuah fungsi hash H, tidak mungkin mencari $y \neq x$ sedemikian sehingga H(y) = H(x). Sifat ini dapat digunakan sebagai *message integrity check (MIC)* dimana perubahan pesan akibat adanya *random corruption* pada transmisi data atau media penyimpanan dapat menghasilkan nilai hash yang jauh berbeda.

Secara garis besar fungsi hash menerima 2 input, yaitu blok pesan (M_i) dan hasil hash pada blok pesan sebelumnya (h_{i-1}). Untuk pesan pertama (M_1), nilai hash sebelumnya (h_0) merupakan vektor awal yang bernilai tetap. Didalam fungsi hash terdapat operasi-operasi dasar seperti circular left shift dan operasi penjumlahan. Operasi-operasi ini dilakukan untuk menghasilkan fungsi hash yang mempunyai tingkat keamanan yang bagus.



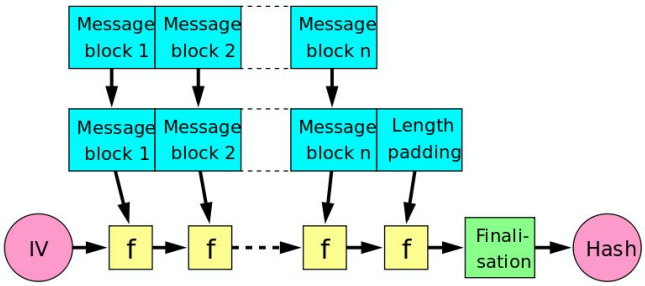
Gambar 1. Skema fungsi hash [1]

Secara keamanan fungsi hash yang bagus adalah fungsi hash yang dapat menghasilkan nilai yang mempunyai kemungkinan yang kecil untuk terjadinya kolisi. kolisi merupakan suatu kejadian saat nilai hasil dari fungsi hash mempunyai nilai yang sama untuk dua pesan yang berbeda.

C. Algoritma MD5

Saat ini algoritma MD5 (*Message Digest*) sudah tidak lagi digunakan karena telah ditemukan cara untuk menghasilkan nilai hash yang sama untuk 2 file yang berbeda. Meskipun demikian, MD5 tetap digunakan untuk mendeteksi adanya *random corruption* pada file karena kecepatannya dalam menghasilkan hash dengan cepat walaupun tidak secepat CRC32.

MD5 merupakan fungsi hash satu arah yang dibuat oleh Ron Rivest. Algoritma ini dapat menerima pesan dengan ukuran sembarang dan menghasilkan *message digest* dengan panjang 128 bits. Secara umum, MD5 menggunakan metode Merkle-Damgard dalam pembuatan *message digest*nya. Merkle-Damgard merupakan metode untuk membuat fungsi hash kriptografis yang bersifat resisten terhadap kolisi.



Gambar 2. Skema Merkle-Damgard [9]

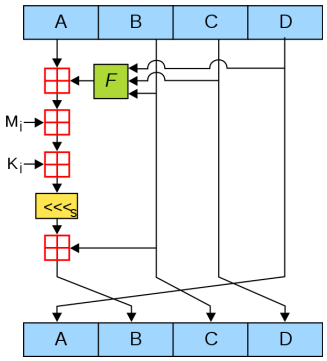
Semua *byte order* pada MD5 menggunakan *little-endian*. Langkah-langkah dalam pembuatan *message digest* secara garis besar sebagai berikut.

1. Penambahan bit-bit pengganjal. Pesan terlebih dahulu ditambahkan sejumlah bit pengganjal sedemikian sehingga panjang bit pesan kongruen dengan 448 (mod 512). Bit-bit pengganjal ini merupakan sekumpulan bit yang diawali dengan bit 1, diikuti dengan bit 0. Panjang bit-bit pengganjal ini antara 1 sampai 512.
2. Penambahan bit nilai panjang pesan. Pesan yang telah diberi bit-bit pengganjal akan mempunyai sisa 64 bit untuk menjadi bit sepanjang kelipatan 512. Sisa 64 bit ini digunakan untuk menyatakan panjang pesan semula. Jika panjang pesan lebih dari 2^{64} maka diambil panjang hasil dari modulo 2^{64}.
3. Inisiasi penyangga MD yang dimaksud merupakan *initialization vector* pada gambar 2. MD5 menggunakan 4 buah penyangga berukuran 32 bits (total panjang penyangga 128 bits). Keempat penyangga ini menjadi hasil antara dan hasil akhir *message digest*. Keempat penyangga tersebut dapat dilihat pada tabel 1.

Tabel 1. Penyangga MD5

	Nilai
A	0x67452301
B	0xEFCDAB89
C	0x98BADCFE
D	0x10325476

4. Pesan diolah pada blok berukuran 512 bits. Blok pesan yang diolah akan menghasilkan penyangga antara dengan panjang 128 bits. Fungsi f pada MD5 akan penerima 4 penyangga (A, B, C, D), dan subpesan (M_i). Fungsi f terdiri dari 4 putaran, dimana setiap putaran mempunyai 16 kali operasi dasar. Operasi dasar tersebut diantaranya adalah fungsi F, penjumlahan modulo 2^{32}, dan *circular left shift*.



Gambar 3. Operasi pada MD5[7]

Setiap operasi dasar $i \mid 0 \le i < 64$ mempunyai nilai K_i, M_i, s_i, dan nilai F yang berbeda. Nilai K_i dapat dilihat pada pseudocode 1. Sedangkan nilai s_i dapat dilihat pada pseudocode 2. Nilai M_i dapat dilihat pada pseudocode 3. Dan fungsi F dapat dilihat pada pseudocode 4.

pseudocode Ki

```
for i from 0 to 63
    K[i] := floor(232 × abs(sin(i + 1)))
end for
```

Pseudocode 1. Nilai pada K_i

pseudocode si

```
s[ 0..15] := { 7, 12, 17, 22 } * 4
s[16..31] := { 5,  9, 14, 20 } * 4
s[32..47] := { 4, 11, 16, 23 } * 4
s[48..63] := { 6, 10, 15, 21 } * 4
```

Pseudocode 2. Nilai pada s_i

pseudocode Mi

```
32bit pesan little-endian
dari blok pesan 512 bit
M[i], 0 ≤ i ≤ 15

for i from 0 to 63
    if i < 16:
        l = i
```

```
    elif i < 32:
        l = (5 * i + 1) % 16
    elif i < 48:
        l = (3 * i + 5) % 16
    else:
        l = (7 * i) % 16
    M[i] = M[l]
```

Pseudocode 3. Nilai pada M_i

```
pseudocode F

var int A := a0
var int B := b0
var int C := c0
var int D := d0

for i from 0 to 63

    if i < 16:
        F := (B and C) or ((not B) and D)
    elif i < 32:
        F := (D and B) or ((not D) and C)
    elif i < 48:
        F := B xor C xor D
    else:
        F := C xor (B or (not D))
```

Pseudocode 4. Nilai F

D. Algoritma SHA1

Algoritma SHA1 merupakan fungsi hash satu arah yang juga menggunakan metode Merkle-Damgard dalam pembuatan *message digest*nya. Perbedaan mendasar antara SHA1 dan MD5 terletak pada panjang nilai hash yang dihasilkan. SHA1 menghasilkan *message digest* sepanjang 160 bits. Dimana SHA1 menggunakan 5 buah penyangga.

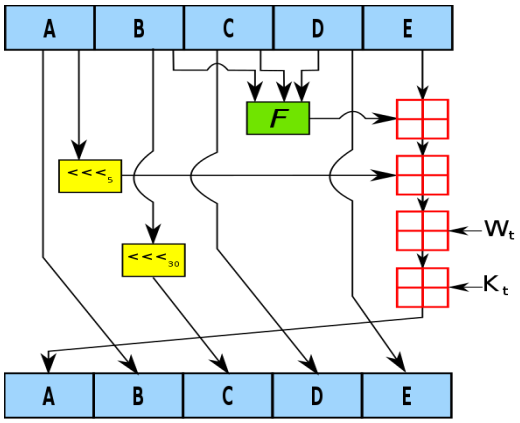
Semua *byte order* pada MD5 menggunakan *big-endian*. Langkah-langkan dalam pembuatan *message digest* untuk SHA1 kurang lebih sebagai berikut.

1. Penambahan bit-bit pengganjal yang prosesnya sama seperti MD5.
2. Penambahan bit nilai panjang pesan yang prosesnya sama seperti MD5
3. Inisiasi penyangga SHA1 dengan menggunakan 5 penyangga.

Tabel 2. Penyangga SHA1

	Nilai
A	0x67452301
B	0xEFCDAB89
C	0x98BADCFE
D	0x10325476
E	0xC3D2E1F0

4. Pesan diolah pada blok berukuran 512 bits. Blok pesan yang diolah akan menghasilkan penyangga antara dengan panjang 160 bits. Fungsi f pada SHA1 akan penerima 5 penyangga (A, B, C, D, E), dan subpesan (W_t). Fungsi f terdiri dari 4 putaran, dimana setiap putaran mempunyai 20 kali operasi dasar. Operasi dasar tersebut diantaranya adalah fungsi F, penjumlahan modulo 2^{32}, dan *circular left shift*.



Gambar 4. Operasi pada SHA1 [8]

Setiap operasi dasar $t \mid 0 \le t < 80$ mempunyai nilai K_t, W_t, s_t, dan nilai F yang berbeda. Nilai K_t dapat dilihat pada pseudocode 5.. Nilai W_t dapat dilihat pada pseudocode 6. Dan fungsi F dapat dilihat pada pseudocode 7.

```
pseudocode Kt

for t from 0 to 79

    if t < 20:
        K[t] := 0x5A827999
    elif t < 40:
        K[t] := 0x6ED9EBA1
    elif t < 60:
        K[t] := 0x8F1BBCDC
    else:
        K[t] := 0xCA62C1D6
```

Pseudocode 5. Nilai pada K_t

```
pseudocode Wt

32bit pesan bit-endian
dari blok pesan 512 bit
```

```
W[t], 0 ≤ t ≤ 15

for t from 16 to 79
    w[t] := w[t-3]^w[t-8]^w[t-14]^w[i-16]
    w[t] := w[t] leftrotate 1
```

Pseudocode 6. Nilai pada W_t

pseudocode Wt

```
var int A := a0
var int B := b0
var int C := c0
var int D := d0

for t from 0 to 79
    if t < 20:
        F := (B & C) | ((~B) & D)
    elif t < 40:
        F := B ^ C ^ D
    elif t < 60:
        F := (B & C) | (B & D) | (C & D)
    else:
        F := B ^ C ^ D
```

Pseudocode 7. Nilai F

III. Implementasi

Bab ini membahas tentang implementasi dari CRC32, MD5, SHA1, API untuk *file sharing verification*, dan program CLI untuk menghitung checksum. Implementasi menggunakan python 3.6. Keseluruhan implementasi dapat dilihat pada https://github.com/deryrahman/api-checksum.git

A. Implementasi CRC32

Implementasi CRC32 menggunakan shift register. Shift register mempunyai panjang yang tetap dan dapat melakukan shift 1 bit terhadap isinya. CRC menggunakan shift register kiri, ketika melakuka shift, MSB (most significant bit) akan dikeluarkan dari register, bit pada posisi MSB-1 akan digeser ke kiri menjadi MSB, MSB-2 digeser menjadi MSB-1, dan seterusnya. Posisi bit terakhir LSB (least significant bit) akan diisi oleh input stream berikutnya. Pada implementasi CRC32 kali ini, menggunakan polinomial 0xEDB88320, mengikuti polinomial yang digunakan pada perintah cksum UNIX.

Kelas CRC32

```
class CRC32():
    def __init__(self):
        self.poly = 0xEDB88320

    def __call__(self, data):
        data = bytearray(data)
        crc = 0xFFFFFFFF
        for d in data:
            crc = crc ^ d
            for _ in range(8):
                crc = (crc >> 1)
                crc ^= (self.poly & -(crc & 1))
        res = 0xFFFFFFFF & ~crc
        res = struct.pack('>I', res)
        return res.hex()
```

Proses implementasi CRC adalah sebagai berikut:

1. Inisiasi register dengan 0xFFFFFFFF.
2. Shift bit data secara satu per satu. Jika MSB yang dikeluarkan dari register adalah 1, maka lakukan operasi XOR dengan polynomialnya. Jika tidak, biarkan.
3. Jika semua bit data telah diproses, nilai dari register merupakan nilai dari CRC yang akan digunakan sebagai checksum.

Pada CRC32 operasi XOR dengan polinomial yang tetap akan mempunyai hasil 2^{32} kali kemungkinan. Untuk meningkatkan performa, dapat digunakan tabel CRC32, dengan cara melakukan perhitungan 2^{32} kemungkinan operasi XOR yang terjadi.

B. Implementasi MD5

Algoritma MD5 kurang lebih diimplementasikan dengan kelas MD5. Kelas ini mempunyai 5 method penting, diantaranya adalah:

1. _expand: untuk menambahkan bit pengganjal, dan panjang pesan sehingga pesan mempunyai kelipatan 512
2. _basic_op: merupakan operasi dasar dari MD5 sesuai dengan gambar 3.
3. _h_md: merupakan fungsi f pada skema Merkle-Damgard yang digunakan pada algoritma MD5
4. _roll_left: merupakan implementasi *circular left shift*.
5. __call__: digunakan untuk membuat hash secara keseluruhan

Berikut adalah implementasi setiap method menggunakan python. Semua bit MD5 menggunakan byte order *little-endian*.

_expand

```
def _expand(self, msg):
    pad = b'\x80' + b'\x00'
    pad *= * ((56 - ((len(msg) + 1) % 64)) % 64)
    bits_len = struct.pack(b'<Q', len(msg) * 8)

    return msg + pad + bits_len
```

_basic_op

```
def _basic_op(self, a, b, c, d, wt, t, s, f):
    res = (a + f(b, c, d) + t + wt) & 0xFFFFFFFF
    res = self._roll_left(res, s)
    res = (res + b) & 0xFFFFFFFF

    return d, res, b, c
```

_h_md

```
def _h_md(self, a, b, c, d, block):
    w = []
    a_ori = a
    b_ori = b
    c_ori = c
    d_ori = d
    for i in range(64):
        if i < 16:
            wi = block[i * 4:i * 4 + 4]
            wi = struct.unpack(b'<I', )[0]
            w.append(wi)

        if i < 16:
            k = i
        elif i < 32:
            k = (5 * i + 1) % 16
        elif i < 48:
            k = (3 * i + 5) % 16
        else:
            k = (7 * i) % 16

        wi = w[k]
        ti = self.t[i]
        si = self.s[i]
        fi = self.f[i//16]
        op = self._basic_op
        res = op(a, b, c, d, wi, ti, si, fi)
        a, b, c, d = res

    a = (a_ori + a) & 0xFFFFFFFF
    b = (b_ori + b) & 0xFFFFFFFF
    c = (c_ori + c) & 0xFFFFFFFF
    d = (d_ori + d) & 0xFFFFFFFF

    return a, b, c, d
```

_roll_left

```
def _roll_left(self, x, n):
    res = ((x << n) | (x >> (32 - n)))
    return res & 0xFFFFFFFF
```

__call__

```
def __call__(self, msg):
    msg = self._expand(msg)
    a = self.buffer_md[0]
    b = self.buffer_md[1]
    c = self.buffer_md[2]
    d = self.buffer_md[3]
    for i in range(len(msg) // 64):
        m = msg[i * 64:i * 64 + 64]
        res = self._h_md(a, b, c, d, )
        a, b, c, d = res
    res = struct.pack('<IIII', a, b, c, d)
    return res.hex()
```

C. Implementasi SHA1

Algoritma SHA1 kurang lebih diimplementasikan dengan kelas SHA1. Sama halnya seperti MD5 kelas ini mempunyai 5 method penting, diantaranya adalah:

6. _expand: untuk menambahkan bit pengganjal, dan panjang pesan sehingga pesan mempunyai kelipatan 512
7. _basic_op: merupakan operasi dasar dari SHA1 sesuai dengan gambar 4.
8. _h_sha: merupakan fungsi f pada skema Merkle-Damgard yang digunakan pada algoritma SHA1
9. _roll_left: merupakan implementasi *circular left shift*.
10. __call__: digunakan untuk membuat hash secara keseluruhan

Implementasi method _roll_left sama seperti pada kelas MD5. Berikut adalah implementasi setiap method menggunakan python. Semua bit SHA1 menggunakan byte order *big-endian*.

_expand

```
def _expand(self, msg):
    pad = b'\x80' + b'\x00'
    pad *= * ((56 - ((len(msg) + 1) % 64)) % 64)
    bits_len = struct.pack(b'>Q', len(msg) * 8)

    return msg + pad + bits_len
```

_basic_op

```
def _basic_op(self, a, b, c, d, e, w, k, f):
    res = (f(b, c, d) + e) & 0xFFFFFFFF
    res = (self._roll_left(a, 5) + res)
    res = res & 0xFFFFFFFF
    res = (w + res) & 0xFFFFFFFF
    res = (k + res) & 0xFFFFFFFF
    b = self._roll_left(b, 30)

    return res, a, b, c, d
```

_h_sha

```
def _h_sha(self, a, b, c, d, e, block):
    """
```

```
    block = 64 bytes
    """
    w = []
    a_ori = a
    b_ori = b
    c_ori = c
    d_ori = d
    e_ori = e
    for t in range(80):
        if t < 16:
            wi = block[t * 4:t * 4 + 4]
            wi = struct.unpack(b'>I', wi)[0]
            w.append(wi)
        else:
            wi = w[t-16]^w[t-14]^w[t-8]^w[t-3]
            wi = self._roll_left(wi, 1)
            w.append(wi)

        wt = w[t]
        kt = self.k[t // 20]
        ft = self.f[t // 20]
        op = self._basic_op
        res = op(a, b, c, d, e, wt, kt, ft)
        a, b, c, d, e = res

    a = (a_ori + a) & 0xFFFFFFFF
    b = (b_ori + b) & 0xFFFFFFFF
    c = (c_ori + c) & 0xFFFFFFFF
    d = (d_ori + d) & 0xFFFFFFFF
    e = (e_ori + e) & 0xFFFFFFFF
    return a, b, c, d, e
```

D. Implementasi API *File Sharing Verification*

API *file sharing verification* dibuat dengan menggunakan python 3.6 dan library Flask 1.0.2. Secara umum, API ini mempunyai 2 endpoint, yaitu:

1. **GET /checksum**. Endpoint ini digunakan untuk membandingkan antara checksum user, dengan checksum pada file yang terdapat pada server. Query pada endpoint ini antara lain adalah:
    a. **mode**: mode checksum; ‘crc32’, ‘md5’, ‘sha1’.
    b. **filename**: nama dari file yang akan diverifikasi.
    c. **checksum**: nilai checksum hasil dari program CLI checksum pada user. Atau bisa juga menggunakan perintah bawaan UNIX yang dapat menghitung checksum.
2. **POST /upload**. Endpoint ini hanya merupakan endpoint bantu untuk mengupload data kedalam server.

E. Implementasi Program CLI Checksum

Program CLI checksum diimplementasikan menggunakan python 3.6. Program ini akan menghitung checksum dari suatu file dan akan membangkitkan url yang akan menuju ke endpoint untuk melakukan pengecekan checksum pada server *file sharing*. Terdapat 2 argument utama pada program CLI checksum ini, yaitu :

1. **mode**: mode checksum; ‘crc32’, ‘md5’, ‘sha1’.
2. **path**: path lokasi file user yang akan dihitung nilai checksumnya.

IV. Pengujian dan Analisis

Pengujian yang dilakukan antara lain melakukan perbandingan kecepatan antar hasil implementasi. Akan digunakan file sebesar 200kb hingga 1800 kb, file bytes ini dibangkitkan secara random dengan menggunakan fungsi os.urandom pada python. Selain itu, pengujian kebenaran implementasi juga dilakukan dengan cara membandingkan hasil checksum dengan checksum yang dihasilkan oleh library yang sudah ada. Untuk CRC32, akan dibandingkan dengan library yang sudah ada yaitu binascii.crc32, untuk MD5, hashlib.md5, sedangkan SHA1, hashlib.sha1.

Hasil kecepatan dari implementasi diperlihatkan pada tabel 3, dalam satuan ms.

Tabel 3. Hasil kecepatan CRC32, MD5, dan SHA1

Ukuran (KB)	CRC32	MD5	SHA1
200	46.69	282.96	587.06
400	77.77	567.73	1141.70
600	114.63	845.36	1664.83
800	147.50	1188.00	2235.24
1000	186.77	1396.68	2767.55
1200	221.03	1687.96	3374.87
1400	256.99	1966.91	4093.06
1600	297.26	2281.80	4419.61
1800	329.19	2611.28	5076.39

Tabel 4, 5, dan 6 menunjukan hasil pengujian kebenaran antara algoritma hasil implementasi dengan library yang sudah ada.

Tabel 4. Hasil CRC32

Pesan	CRC32 implementasi	CRC32 binascii.crc32
Lorem ipsum dolor sit amet, consectetur adipiscing elit	6c8ada71	6c8ada71

Tabel 5. Hasil MD5

Pesan	MD5 implementasi	MD5 hashlib.md5
Lorem ipsum dolor sit amet, consectetur adipiscing elit	fc10a08df7fafa3871166646609e1c95	fc10a08df7fafa3871166646609e1c95

Tabel 6. Hasil SHA1

Pesan	SHA1 implementasi	SHA1 hashlib.sha1
Lorem ipsum dolor sit amet, consectetur adipiscing elit	4045ed3c779e3b27760e4da357279508a8452dcb	4045ed3c779e3b27760e4da357279508a8452dcb

Selain itu pengujian kebenaran juga dilakukan dengan melakukan pembangkitan secara random menggunakan fungsi os.urandom, sebanyak 100 kali, dan menghasilkan nilai hash yang 100% sama dengan hasil hash oleh library.

VI. Kesimpulan

Adanya kemungkinan untuk terjadinya error pada mekanisme transmisi data ataupun penyimpanan data dapat dideteksi dengan menggunakan checksum. Salah satu algoritma checksum yang sering digunakan adalah CRC32. Namun, pada dasarnya algoritma hash seperti MD5 dan SHA1 juga dapat digunakan untuk melakukan verifikasi integritas data. CRC32, MD5, dan SHA1 sering digunakan untuk mendeteksi *random corruption* pada data karena performanya yang cepat.

Ucapan Terimakasih

Penulis mengucapkan terimakasih kepada Allah SWT, karena atas berkat karuniaNya penulis dapat menyelesaikan makalah ini. Tidak lupa penulis mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir selaku dosen mata kuliah kriptografi. Atas berkat ilmu beliau, penulis dapat memahami konsep dari kriptografi dalam pembuatan makalah ini.

References

[1] Munir, Rinaldi. 2019. Slide Kuliah IF4020 Kriptografi: Fungsi Hash

[2] Munir, Rinaldi. 2019. Slide Kuliah IF4020 Kriptografi: Algoritma MD5

[3] Munir, Rinaldi. 2019. Slide Kuliah IF4020 Kriptografi: Secure Hash Algorithm SHA

[4] http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html terakhir diakses pada 9 Mei 2019

[5] Rivest, R. (April 1992). The MD5 Message-Digest Algorithm. doi:10.17487/RFC1321

[6] Chaitya B. Shah. (Oktober 2014). Secured Hash Algorithm-1: Review Paper. Volume 2, Issue X, Oct 2014 ISSN 2320-6802

[7] https://en.wikipedia.org/wiki/MD5 terakhir diakses pada 9 Mei 2019

[8] https://en.wikipedia.org/wiki/SHA-1 terakhir diakses pada 10 Mei 2019

[9] https://en.wikipedia.org/wiki/Merkle–Damgård_construction terakhir diakses pada 9 Mei 2019

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Mei 2019

Dery Rahman Ahaddienata