Sistem Penyimpanan Kata Sandi Menggunakan Algoritma Shamir Secret Sharing

Karel Renaldi - 13519180 (*Author*)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519180@std.stei.itb.ac.id

***Abstract*—Kata sandi atau *password* merupakan kumpulan dari karakter-karakter atau yang sering disebut sebagai *string* yang umumnya digunakan pada komputer / jaringan dimana *password* ini digunakan sebagai alat / sarana untuk melakukan verifikasi pengguna terhadap suatu sistem didalam komputer / jaringan. Karena *password* ini memiliki peranan yang sangat penting atau bisa disebut juga sangat *critical*, maka *password* umumnya dibuat kompleks sehingga pihak yang tidak berwenang / *attacker* sulit menebak *password* dari pengguna. Namun karena kompleks maka *password* ini mudah untuk hilang dari ingatan oleh karena itu pada makalah ini akan dibahas suatu solusi penyimpanan kata sandi kompleks dengan cara melakukan partisi dari *password* menjadi beberapa bagian menggunakan algoritma *shamir secret sharing*.**

***Keywords—password*, algoritma *shamir secret sharing*, *attacker*, *critical*, kumpulan karakter.**

I. Pendahuluan

Kriptografi merupakan sebuah cabang dari kriptologi yang merupakan ilmu tentang menjaga sebuah kerahasiaan dari sebuah pesan. Secara *general* kriptografi merupakan sebuah teknik untuk melakukan rekonstruksi dan analisis protokol / aturan komunikasi sehingga sulit untuk dianalisis oleh *attacker*. Kata *Cryptography* sendiri berasal dari bahasa yunani yaitu "cryptós" (*secret*) dan gráphein(*writing*). Kriptografi sendiri memiliki 4 *service* yang disediakan yaitu :

a. Kerahasiaan pesan (*Confidentiality*)
b. Keaslian pesan (*Data integrity*)
c. Keaslian pengirim dan penerima pesan (*Authentication*)
d. Anti penyangkalan (*Non-repudiation*)

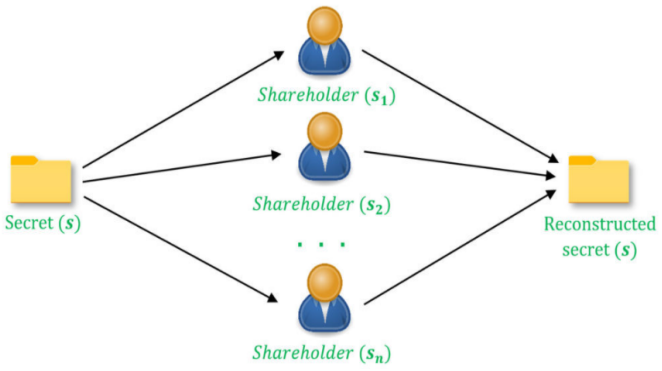
Kriptografi terbagi menjadi 2 jenis yaitu kriptografi klasik dan kriptografi modern. Kriptografi klasik sendiri umumnya digunakan sebelum zaman komputer modern sedangkan kriptografi modern digunakan pada zaman komputer modern sampai sekarang namun, kriptografi klasik juga masih sering digunakan karena kriptografi klasik sendiri merupakan landasan penting dari kriptografi modern sehingga kriptografi modern dan kriptografi klasik bisa disebut tidak terpisahkan. Selain itu kriptografi juga terbagi menjadi 2 jenis berdasarkan *flow* kerjanya yaitu kriptografi simetris dan kriptografi asimetris. Secara singkat, kriptografi simetris merupakan sebuah proses kriptografi dimana proses enkripsi dan dekripsi dilakukan dengan kunci yang sama sedangkan pada kriptografi asimetrik proses enkripsi dan dekripsi dilakukan dengan kunci yang berbeda dimana pada proses asimetrik ini sebenarnya jauh lebih aman karena terdapat 2 buah kunci yaitu kunci publik dan kunci privat. Secara sederhana, proses kriptografi asimetris adalah *receiver* akan men-*generate* 2 buah kunci yaitu kunci publik dan kunci privat dimana kunci publik nanti akan dikirim kepada *sender* sehingga *sender* akan melakukan enkripsi dengan kunci publik tersebut dan *receiver* akan membuka pesan tersebut menggunakan kunci privat. Jika dilihat dari *flow* kerjanya bisa disimpulkan bahwa memang kunci publik sesuai namanya dapat di-*publish* ke publik sedangkan kunci privat harus disimpan.Seiring berkembangnya dunia kriptografi dan matematika terdapat sebuah metode untuk menjaga kerahasiaan dari sebuah kunci dengan cara yang unik yaitu membagikan kunci tersebut kepada beberapa orang namun untuk membangkitkan kunci itu kembali membutuhkan sejumlah orang. Metode ini dinamakan metode *secret sharing scheme* yang diperkenalkan oleh salah satu dari pencipta algoritma kunci publik RSA yaitu Adi Shamir. Metode ini memungkinkan kunci utama dapat dipecah menjadi beberapa bagian namun untuk merekonstruksi kunci tersebut ke kunci utama awal diperlukan beberapa bagian yang sudah ditentukan dari awal.Metode ini juga didasarkan pada sebuah ilmu dari matematika yaitu persamaan polinomial. Banyak implementasi yang dapat digunakan dari metode ini salah satunya adalah yang dibahas pada makalah ini.yaitu pemecahan *password* sehingga *password* sendiri menjadi lebih aman. Selain itu skema ini juga memiliki manfaat yaitu membuat *password* menjadi lebih mudah diingat karena nantinya *password* akan dibagi ke beberapa orang dan untuk merekonstruksi *password* tersebut butuh beberapa orang juga sehingga bisa dibilang *password* menjadi lebih aman dan mudah untuk diingat.

II. Landasan Teori

A. Secret Sharing Scheme

Secret Sharing Scheme merupakan sebuah skema untuk membagi informasi rahasia ke dalam beberapa partisi dimana untuk melakukan rekonstruksi ke bentuk awal diperlukan jumlah tertentu yang sudah ditentukan sejak awal. Terdapat beberapa istilah dalam skema ini yaitu *Share* yang merupakan hasil dari partisi / pembagian dari informasi rahasia / *secret*,

Dealer yang merupakan pihak / orang yang melakukan pembagian *secret* dan *Participant / Shareholder* yang merupakan orang yang menerima *share*.



Secara umum, cara kerja dari *secret sharing scheme* ini adalah membagi *secret* kedalam beberapa *fragment*. Hal ini dilakukan agar mencegah hanya 1 pihak bisa mengakses secret namun dibutuhkan beberapa orang yang sudah ditentukan.Terdapat sebuah skema lain dalam skema ini yang harus dipahami terlebih dahulu yaitu *Threshold Schemes*. Untuk mudah memahami arti dari skema ini kita misalkan terlebih dahulu bahwa ada 2 buah bilangan bulat positif yaitu t dan w dimana t ≤ w dan terdapat 1 buah variabel yaitu M yang merepresentasikan sebuah *message* / pesan / *secret*. *Threshold Schemes* / Skema Ambang (t, w) adalah metode pembagian pesan M kepada w partisipan sedemikian rupa sehingga sembarang himpunan bagian yang terdiri dari t buah elemen / partisipan dapat merekonstruksi M, tetapi jika kurang dari t maka M tidak dapat direkonstruksi. Skema ambang ini ditemukan pada tahun 1979 oleh salah satu dari penemu algoritma kunci publik RSA yaitu Adi Shamir sehingga sering disebut juga skema ambang shamir (*Shamir threshold scheme*).Salah satu algoritma / implementasi dari *Secret Sharing Scheme* adalah *Shamir Secret Sharing*.*Shamir Secret Sharing* merupakan implementasi yang paling populer dalam algoritma *Secret Sharing Scheme* yang ditemukan / diciptakan oleh Adi Shamir salah satu kriptografi asal israel yang sangat terkenal yang mana orang ini juga merupakan salah satu penemu dari algoritma kunci publik RSA. *Shamir Secret Sharing* memungkinkan pesan dibagi kedalam beberapa *shares* yang bebas dan juga *threshold* yang bebas (selama *threshold* masih kurang dari jumlah total partisipan). *Shamir Secret Sharing* didasarkan pada sebuah konsep matematika yaitu interpolasi polinomial.Untuk mudah memahami mengapa *Shamir Secret Sharing* didasarkan pada konsep matematika interpolasi polinomial maka kita coba tinjau beberapa kasus:

$$y = \alpha_1 + \alpha_2 x$$

Bisa kita lihat bahwa untuk persamaan linier seperti persamaan diatas untuk dapat merekonstruksi persamaan tersebut kita membutuhkan minimal 2 buah titik, misalkan titik (x1, y1) dan (x2, y2) dimana x1, y1, x2, y2 merupakan bilangan real.

$$y = \alpha_1 + \alpha_2 x + \alpha_3 x^2$$

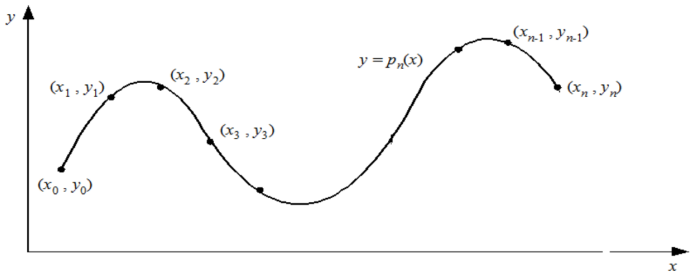
Untuk persamaan kuadrat seperti diatas diperlukan minimal 3 buah titik untuk merekonstruksi persamaan diatas, misalkan : (x1, y1), (x2, y2), (x3, y3) dimana x1, y1, x2, y2, x3, y3 merupakan bilangan real.

$$y = \alpha_1 + \alpha_2 x + \cdots + \alpha_n x^{n-1}$$

Untuk persamaan derajat n seperti persamaan diatas diperlukan minimal n+1 buah titik untuk melakukan rekonstruksi, misalkan (x1, y1), (x2, y2), …, (xn+1, yn+1) dimana x1, y1, x2, y2, …, xn+1, yn+1 merupakan bilangan real.

Dari penalaran kita diatas bisa kita simpulkan bahwa polinom interpolasi derajat n yang melakukan interpolasi titik-titik adalah :

$$y = p_n(x) = a_0 + a_1 x + a_2 x_2 + \ldots + a_n x^n$$



Jika kita lihat dari gambar di atas untuk mendapatkan persamaan polinomial yang sempurna kita perlu melakukan substitusi / penyulihan dari nilai-nilai / titik-titik yang sudah kita punya sebelumnya yaitu : (x0, y0), (x1, y1), …, (xn, yn). Untuk lebih jelasnya kita coba substitusikan nilai-nilai / titik-titik tersebut ke dalam persamaan polinomial general diatas.

$$\begin{aligned} a_0 + a_1 x_0 + \cdots + a_n x_0^n &= y_0 \\ a_0 + a_1 x_1 + \cdots + a_n x_1^n &= y_1 \\ &\vdots \\ a_0 + a_1 x_n + \cdots + a_n x^n &= y_n \end{aligned}$$

Bisa kita lihat dari gambar diatas bahwa kita memiliki n + 1 buah persamaan yang harus diselesaikan, teknik paling mudah untuk menyelesaikan n + 1 buah persamaan tersebut adalah dengan menggunakan metode eliminasi Gauss Jordan. Jika kita sudah berhasil melakukan metode tersebut maka kita

dapat mendapatkan nilai" yang diinginkan yaitu nilai a0, a1, a2, …, an.

Kita sudah memahami motivasi dari *secret sharing scheme* ini selanjutnya penulis akan membahas tentang algoritma / pseudocode dari algoritma ini yaitu sebagai berikut (asumsi kita akan membuat skema(t, w):

1) Pilih sembarang bilangan bulat prima kita misalkan bilangan ini sebagai variabel p dimana variabel ini harus bernilai lebih besar dari semua kemungkinan nilai pesan M dan juga harus lebih besar dari jumlah w partisipan. Nantinya nilai p ini akan digunakan sebagai nilai modulus dalam komputasi.

2) Pilih bilangan bulat acak sebanyak t - 1 dalam modulus p kita misalkan bilangan ini sebagai variabel $s_1, s_2, \ldots, s_{t-1}$ dan buat persamaan polinomial dalam modulus p seperti gambar berikut :

$$S(x) \equiv M + s_1 x + s_2 x^2 + \ldots + s_{t-1} x^{t-1} \pmod p$$

sedemikian rupa sehingga S(0) kongruen dengan M (mod p).

3) Untuk w buah partisipan, kita pilih integer acak berbeda kita misalkan $x_1, x_2, \ldots, x_w \pmod p$. Setiap orang misalkan mendapatkan / memperoleh *share* yaitu share(x_i, y_i) dimana dalam hal ini nilai y_i adalah :

$$y_i \equiv S(x_i) \pmod p$$

Sebagai contoh kita bisa memisalkan bahwa untuk w buah orang kita memilih $x_1 = 1$, $x_2 = 2$, …, $x_w = w$.

III. Implementasi

Pada bagian ini kita akan melakukan implementasi, untuk mempermudah penulis akan melakukan analisis pada setiap bagian kode sehingga semakin mudah untuk dipahami, selain itu implementasi dilakukan dengan menggunakan bahasa pemrograman python.

```
def generate_shares(n, m, secret):
    """
    Melakukan pemecahan / partisi pada secret menjadi n buah
    bagian dimana minimum threshold yang harus dimiliki adalah
    m buah shares untuk dapat merekonstruksi ulang secret dimana
    menggunakan SSS algorithm.
    """
    coefficients = coeff(m, secret)
    shares = []

    for _ in range(1, n+1):
        temp = random.randrange(1, p)
        shares.append((temp, polinom(temp, coefficients)))

    return shares
```

Blok kode diatas merupakan blok kode untuk melakukan partisi secret ke dalam sejumlah shares yang telah ditentukan yang nantinya hasilnya merupakan potongan secret yang akan diterima oleh *shareholder* / participant.

```
def coeff(t, secret):
    """
    Secara random akan melakukan generate list koefisien untuk polinom
    berderajat t - 1 dimana nilai konstan adalah secret.
    """
    coeff = []
    for _ in range(t - 1):
        coeff.append(random.randrange(0, p))
    coeff.append(secret)
    return coeff
```

Blok kode diatas merupakan blok kode untuk melakukan generate koefisien berderajat t - 1 seperti yang sudah dituliskan sebelumnya dimana nilai konstanta nantinya merupakan *message* / *secret*.

```
def polinom(x, c):
    """
    Melakukan generate single point pada graf polinomial yang dimasukan
    nilai x. polinomial diberikan oleh parameter c.
    """

    point = 0
    for coefficient_index, coefficient_value in enumerate(c[::-1]):
        point += x ** coefficient_index * coefficient_value
    return point
```

Blok kode diatas sederhananya adalah menghitung sebuah point seperti yang sudah dituliskan sebelumnya dimana sebenarnya hasil list dari fungsi coeff akan dilakukan traversal secara reversed lalu dijumlahkan hasilnya lalu dilakukan return. Fungsi ini akan dipanggil pada saat partisi *secret* kedalam beberapa *shareholder*.

```
def reconstruct_secret(shares):
    sums = 0

    for j, share_j in enumerate(shares):
        xj, yj = share_j
        prod = float(1)
```

```
    for i, share_i in enumerate(shares):
        xi, _ = share_i
        if i != j:
            prod *= float(float(xi)/(xi-xj))

    prod *= yj
    sums += float(prod)

  return int(round(float(sums), 0))
```

Blok kode diatas merupakan blok kode untuk melakukan rekonstruksi awal dimana fungsi ini akan menerima parameter *shares* yang merupakan list dari informasi *shareholder* yang ingin untuk direkonstruksi ulang. Fungsi ini akan mengembalikan return *secret* jika informasi atau potongan *secret* dari *shareholder* memang benar nilainya dan jumlah dari *shareholder* sudah memenuhi batas ambang / *threshold* yang ditentukan.

Jadi secara singkat, kita bisa merangkum bahwa program diatas akan melakukan pembagian / partisi *message* berdasarkan algoritma / pseudocode yang sudah dituliskan di atas lalu setelah itu juga program bisa melakukan rekonstruksi ulang pesan *secret* dari informasi potongan-potongan pesan yang dimiliki oleh *shareholder* dimana jumlah informasi *shareholder* minimal memenuhi dari batas ambang yang ditentukan. Untuk mempermudah berikut kode lengkap implementasi program diatas.

```
import random

p = 10**5

def reconstruct_secret(shares):
  sums = 0

  for j, share_j in enumerate(shares):
    xj, yj = share_j
    prod = float(1)

    for i, share_i in enumerate(shares):
      xi, _ = share_i
      if i != j:
        prod *= float(float(xi)/(xi-xj))

    prod *= yj
    sums += float(prod)

  return int(round(float(sums), 0))


def polinom(x, c):
  """
  Melakukan generate single point pada graf polinomial yang dimasukan
  nilai x. polinomial diberikan oleh parameter c.
  """

  point = 0
  for coefficient_index, coefficient_value in enumerate(c[::-1]):
    point += x ** coefficient_index * coefficient_value
  return point


def coeff(t, secret):
  """
  Secara random akan melakukan generate list koefisien untuk polinom
  berderajat t - 1 dimana nilai konstan adalah secret.
  """
  coeff = []
  for _ in range(t - 1):
    coeff.append(random.randrange(0, p))
  coeff.append(secret)
  return coeff


def generate_shares(n, m, secret):
  """
  Melakukan pemecahan / partisi pada secret menjadi n buah
  bagian dimana minimum threshold yang harus dimiliki adalah
  m buah shares untuk dapat merekonstruksi ulang secret dimana
  menggunakan SSS algorithm.
  """
  coefficients = coeff(m, secret)
  shares = []

  for _ in range(1, n+1):
    temp = random.randrange(1, p)
    shares.append((temp, polinom(temp, coefficients)))

  return shares

if __name__ == '__main__':
  t, n = 3, 5
  secret = 1234

  # Generate
  res = generate_shares(n, t, secret)
  print(res)

  # Reconstruct
  res = random.sample(res, t)
  print(res)
```

IV. Pengujian

Percobaan pertama adalah percobaan melakukan partisi kedalam 5 orang dimana *threshold* nya adalah 3 dan pesan *secret* adalah “54321”.

```
$ python test.py
secret = 54321
Nilai shareholder: (19578, 7573835086383), (41128,
33420455704233), (92427, 168777175858542), (4274,
361189618391), (69279, 94825446675786)
```

Setelah nilai dari setiap *shareholder* didapatkan maka kita akan mencoba untuk melakukan rekonstruksi ulang terhadap nilai yang diberikan. Untuk percobaan pertama kita akan mencoba memasukan 3 buah nilai yaitu sesuai nilai ambang.

```
temp = [(19578, 7573835086383), (41128,
33420455704233), (92427, 168777175858542)]
   print(f'Menggabungkan shares: {", ".join(str(share) for
share in temp)}')
   print(f'Hasil rekonstruksi: {reconstruct_secret(temp)}')
```

```
$ python test.py
Menggabungkan shares: (19578, 7573835086383), (41128,
33420455704233), (92427, 168777175858542)
Hasil rekonstruksi: 54321
```

Bisa kita lihat bahwa memang benar bahwa hasil rekonstruksi berhasil untuk jumlah *shares* yang memenuhi atau sebanyak nilai ambang. Percobaan berikutnya adalah dengan mencoba untuk memasukan *shares* dengan jumlah yang kurang dari batas ambang berikut hasil percobaannya :

```
$ python test.py
Menggabungkan shares: (19578, 7573835086383), (41128,
33420455704233)
Hasil rekonstruksi: -15907609853583
```

Bisa kita lihat bahwa memang jika jumlah *shares* kurang dari jumlah ambang maka akan berakibat mendapatkan hasil rekonstruksi yang tidak sama dengan *secret* awal. Percobaan berikutnya adalah dengan mencoba memasukkan jumlah *shares* yang lebih banyak dari jumlah ambang seperti berikut :

```
  temp = [
      (19578, 7573835086383),
      (41128, 33420455704233),
      (92427, 168777175858542),
      (4274, 361189618391),
      (69279, 94825446675786)
  ]
  print(f'Menggabungkan shares: {", ".join(str(share) for
share in temp)}')
  print(f'Hasil rekonstruksi: {reconstruct_secret(temp)}')
```

```
$ python test.py
Menggabungkan shares: (19578, 7573835086383), (41128,
33420455704233), (92427, 168777175858542), (4274,
361189618391), (69279, 94825446675786)
Hasil rekonstruksi: 54321
```

Dari hasil di atas bisa dilihat bahwa pada saat jumlah *shares* lebih besar dari jumlah *threshold* atau nilai ambang maka hasil rekonstruksi akan selalu berhasil, sehingga dapat kita simpulkan bahwa program dapat berjalan dengan baik.

Dari hasil diatas sebenarnya *secret* merupakan *password* yang akan kita bagikan kepada beberapa orang agar informasi dari *password* kita lebih aman dan berdasarkan percobaan diatas bisa kita simpulkan bahwa memang berhasil untuk menerapkan *password sharing* dengan skema *secret sharing*.

V. Analisis dan Kesimpulan

Algoritma *shamir secret sharing* dan konsep dari *secret sharing scheme* sangat berguna dan memang setelah dilakukan beberapa percobaan diatas bahwa *secret sharing scheme* ini juga sangat aman karena kita butuh beberapa *shareholder* dari *shareholder* total untuk melakukan rekonstruksi ulang terhadap pesan *secret*. Dari percobaan diatas juga sudah berhasil menunjukan bahwa memang jika jumlah *shareholder* pada saat melakukan rekonstruksi kurang maka terjadi kegagalan pada saat melakukan rekonstruksi yaitu menghasilkan nilai yang berbeda.

Pada percobaan kali ini dapat disimpulkan bahwa berhasil dan juga bisa digunakan untuk tujuan *password management*. Implementasi dalam makalah ini belum sempurna yang mana memang masih bisa ditambahkan untuk mendapatkan *password management* yang lebih baik lagi yaitu dengan cara membuat implementasi yang tidak hanya menerima angka namun juga huruf yaitu dengan cara melakukan konversi karakter ke dalam angka selain itu juga *password* bisa dilakukan enkripsi terlebih dahulu sebelum dilakukan partisi sehingga lebih aman.

VI. Ucapan Terima Kasih

Puji syukur pertama-tama saya panjatkan kepada Tuhan Yang Maha Esa karena berkat dan rahmat-Nya saya dapat menyelesaikan makalah ini dengan baik dan benar. Saya juga mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku pengajar mata kuliah Kriptografi dan pembimbing dalam pembuatan makalah ini. Tidak lupa juga

saya ucapkan terimakasih kepada keluarga dan teman-teman tercinta yang tidak henti-hentinya memberikan dukungan moral serta bantuan yang tidak bisa disebutkan satu persatu.

VII. Referensi

[1] https://www.geeksforgeeks.org/implementing-shamirs-secret-sharing-scheme-in-python/

[2] https://id.wikipedia.org/wiki/Kriptografi

[3] https://www.cs.bgu.ac.il/~beimel/Papers/Survey.pdf

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, 19 Desember 2021

Karel Renaldi 13519180