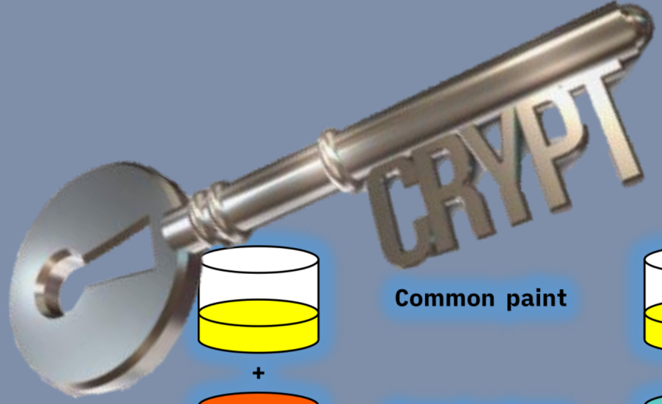
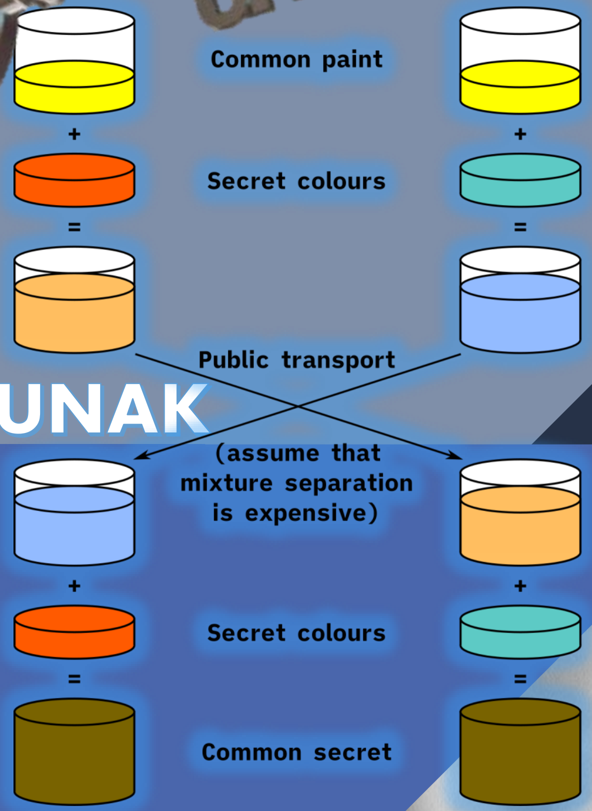
#24





Syahrul Imardi, MT

STMIK Amik Riau

MATAKULIAH

KEAMANAN PERANGKAT LUNAK

Algoritma Diffie-Hellman & Knapsack

P24

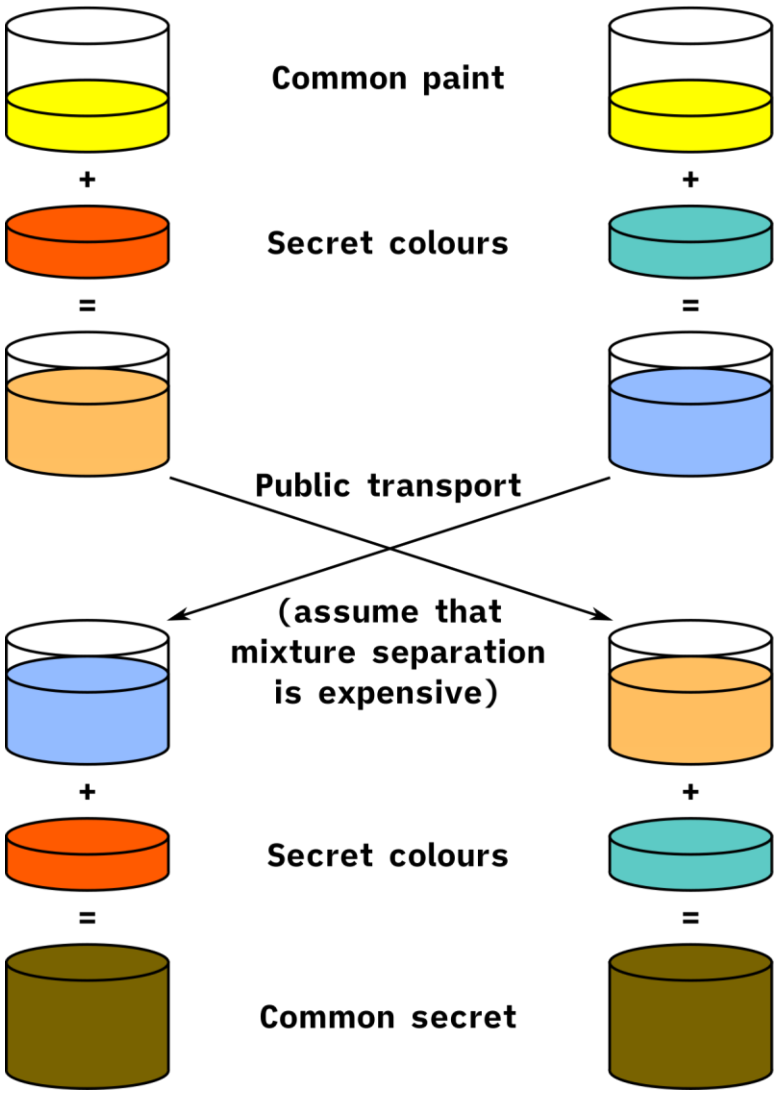
STMIK
Amik Riau

MATAKULIAH
KEAMANAN PERANGKAT LUNAK

Syahrul Imardi, MT

Public Key Cryptography

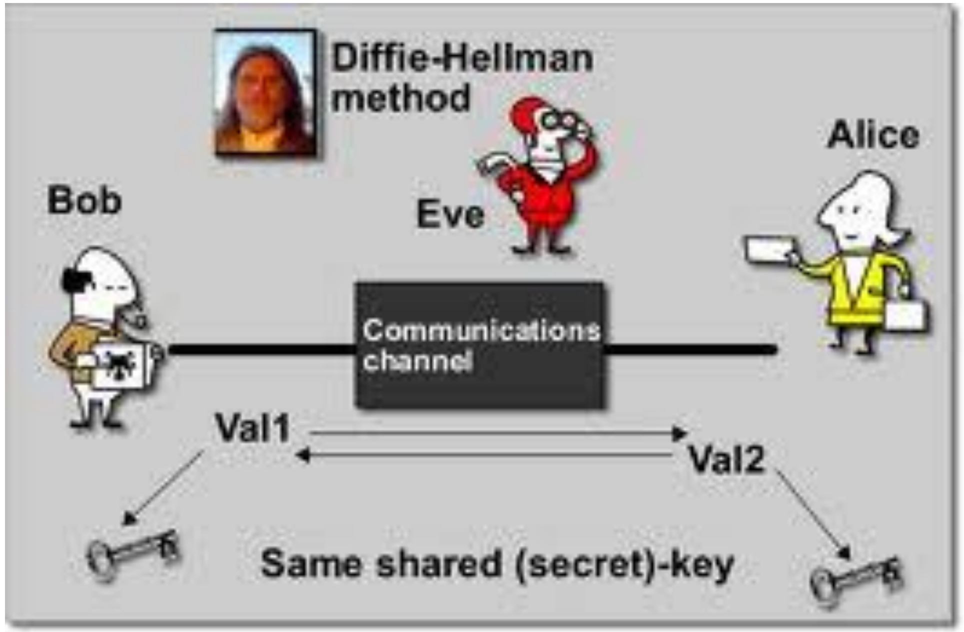
P24: **Algoritma Diffie-Hellman & Knapsack**



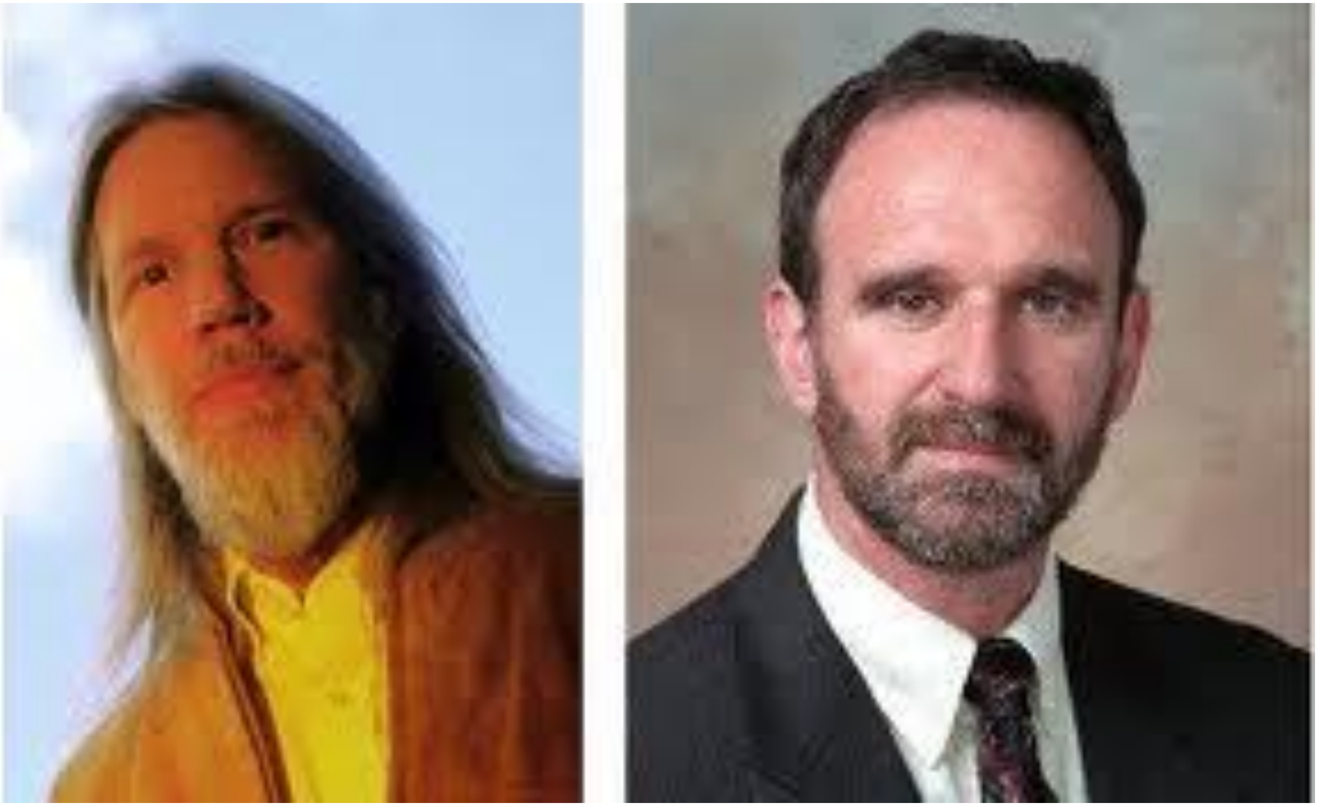
Algoritma Pertukaran Kunci Diffie-Hellman

Latar Belakang

- Kegunaan: untuk berbagi kunci rahasia yang sama antara dua entitas yang berkomunikasi. Kunci rahasia digunakan untuk mengenkripsi pesan dengan algoritma kriptografi kunci-simeteri (DES, AES, dll)



- Keamanan algoritmanya didasarkan pada sulitnya menghitung logaritma diskrit.



Whitfield **Diffie** and Martin **Hellman**

Parameter umum Diffie-Hellman

- Misalkan dua orang yang berkomunikasi: Alice dan Bob.

- Mula-mula Alice dan Bob menyepakati bilangan prima yang besar, n dan g, sedemikian sehingga $g < n$.

- Bilangan n dan g tidak perlu rahasia. Bahkan, Alice dan Bob dapat membicarakannya melalui saluran yang tidak aman sekalipun.

Algoritma Pertukaran Kunci Diffie-Hellman

1. Alice membangkitan bilangan bulat acak yang besar x dan mengirim hasil perhitungan berikut kepada Bob:

   $X = g^x \bmod n$

2. Bob membangkitkan bilangan bulat acak yang besar y dan mengirim hasil perhitungan berikut kepada Alice:

   $Y = g^y \bmod n$

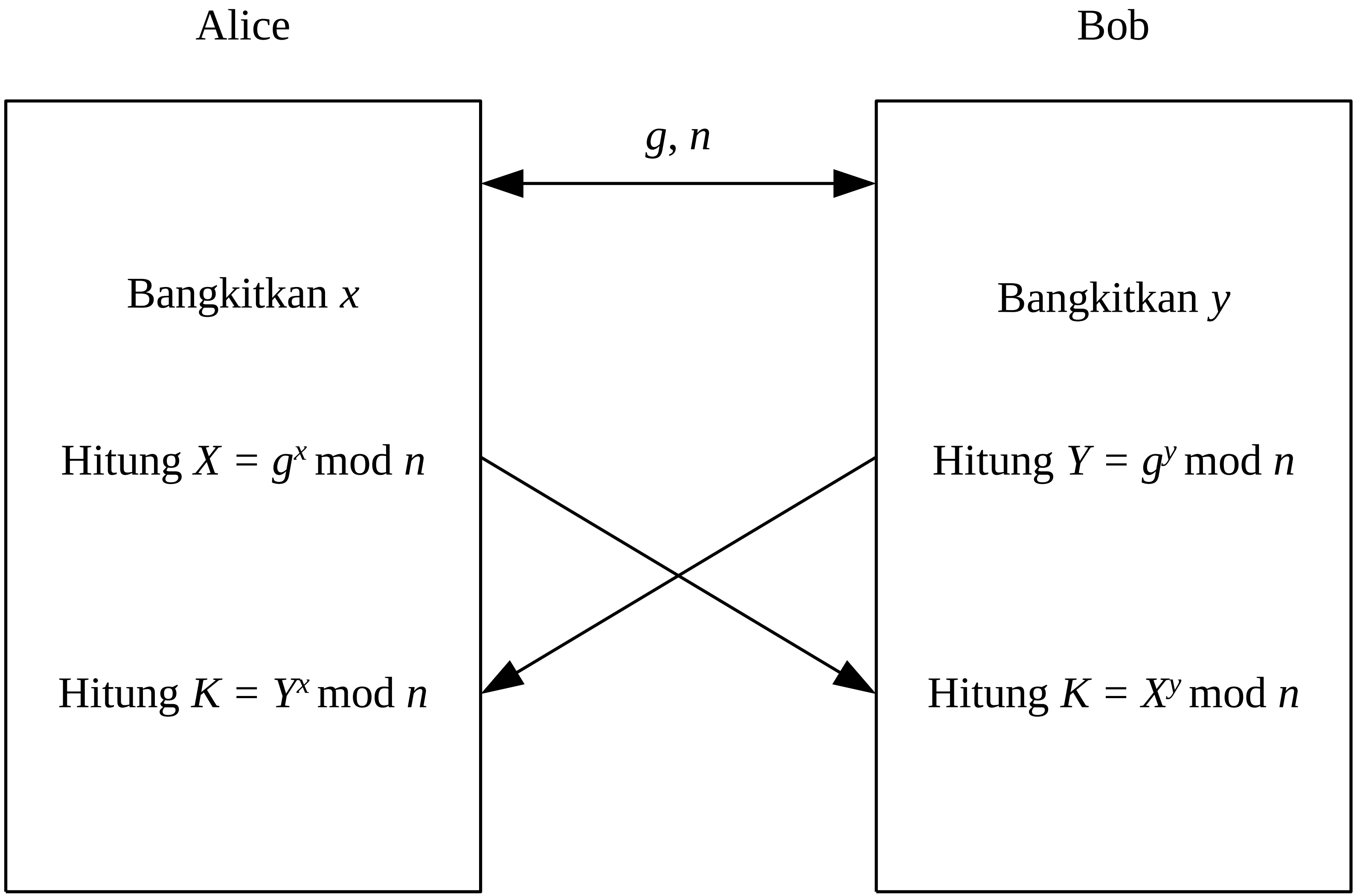
3. Alice menghitung

   $K = Y^x \bmod n$

4. Bob menghitung

   $K' = X^y \bmod n$

- Jika perhitungan dilakukan dengan benar, maka $K = K'$. Baik K dan $K' = g^{xy} \bmod n$.


Alice
Bob
g, n
Bangkitkan x
Bangkitkan y
Hitung $X = g^x \bmod n$
Hitung $Y = g^y \bmod n$
Hitung $K = Y^x \bmod n$
Hitung $K = X^y \bmod n$

Eve (seorang kriptanalis) yang menyadap pembicaraan antara Alice danBob tidak dapat menghitung *K*.

- Eve hanya memiliki informasi *n*, *g*, *X* dan *Y* (yang tidak rahasia), tetapi ia tidak mempunyai informasi nilai *x* atau *y*.

- Untuk mengetahui *x* , Eve perlu melakukan perhitungan untuk menemukan *x* dari persamaan $X = g^x \bmod n$.

- Sekali *x* diketahui, maka selanjutnya Eve menggunakannya untuk menghitung kunci $K = Y^x \bmod n$.

- Kabar baiknya, logaritma diskrit sangat sulit dihitung.

Contoh: Alice dan Bob menyepakati $n = 97$ dan $g = 5$ ($g < n$)

1. Alice memilih $x = 36$ dan menghitung

   $X = g^x \bmod n = 5^{36} \bmod 97 = 50$

   Alice mengirim X kepada Bob.

2. Bob memilih y = 58 dan menghitung

   $Y = g^y \bmod n = 5^{58} \bmod 97 = 44$

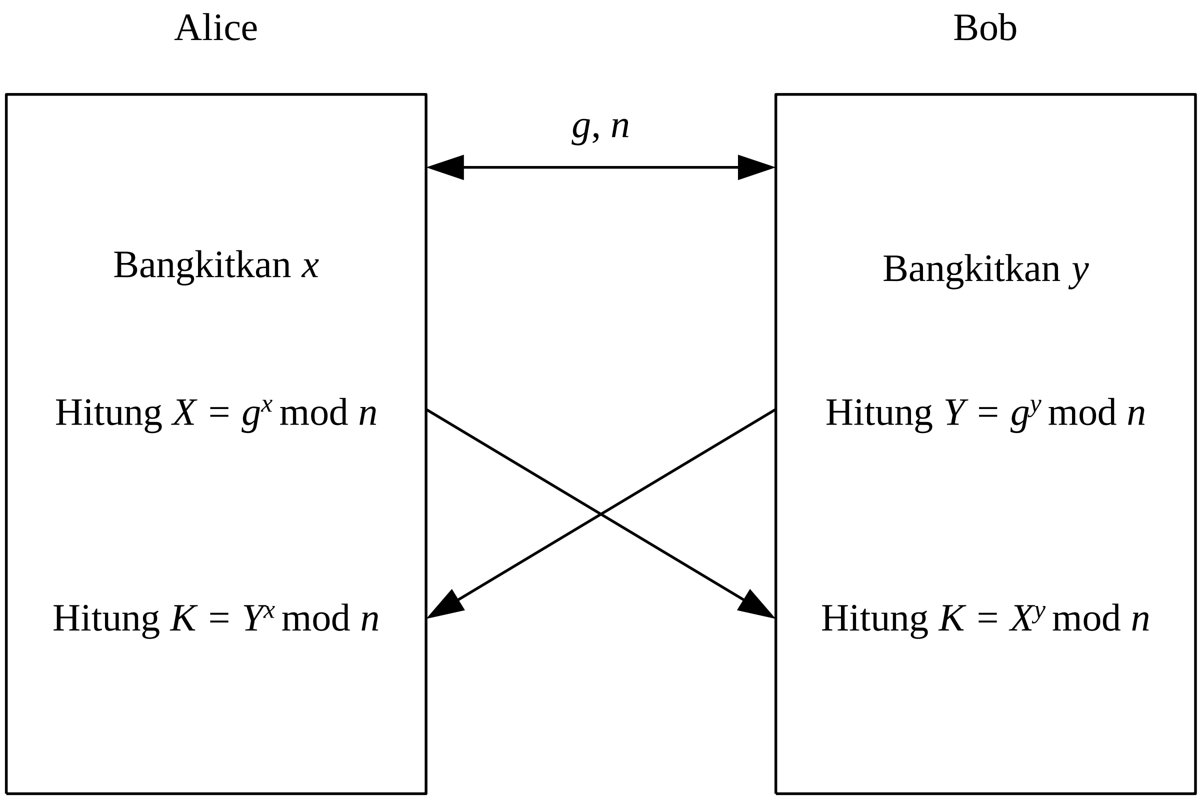
   Bob mengirim Y kepada Alice.

3. Alice menghitung kunci simetri K,

   $K = Y^x \bmod n = 44^{36} \bmod 97 = 75$

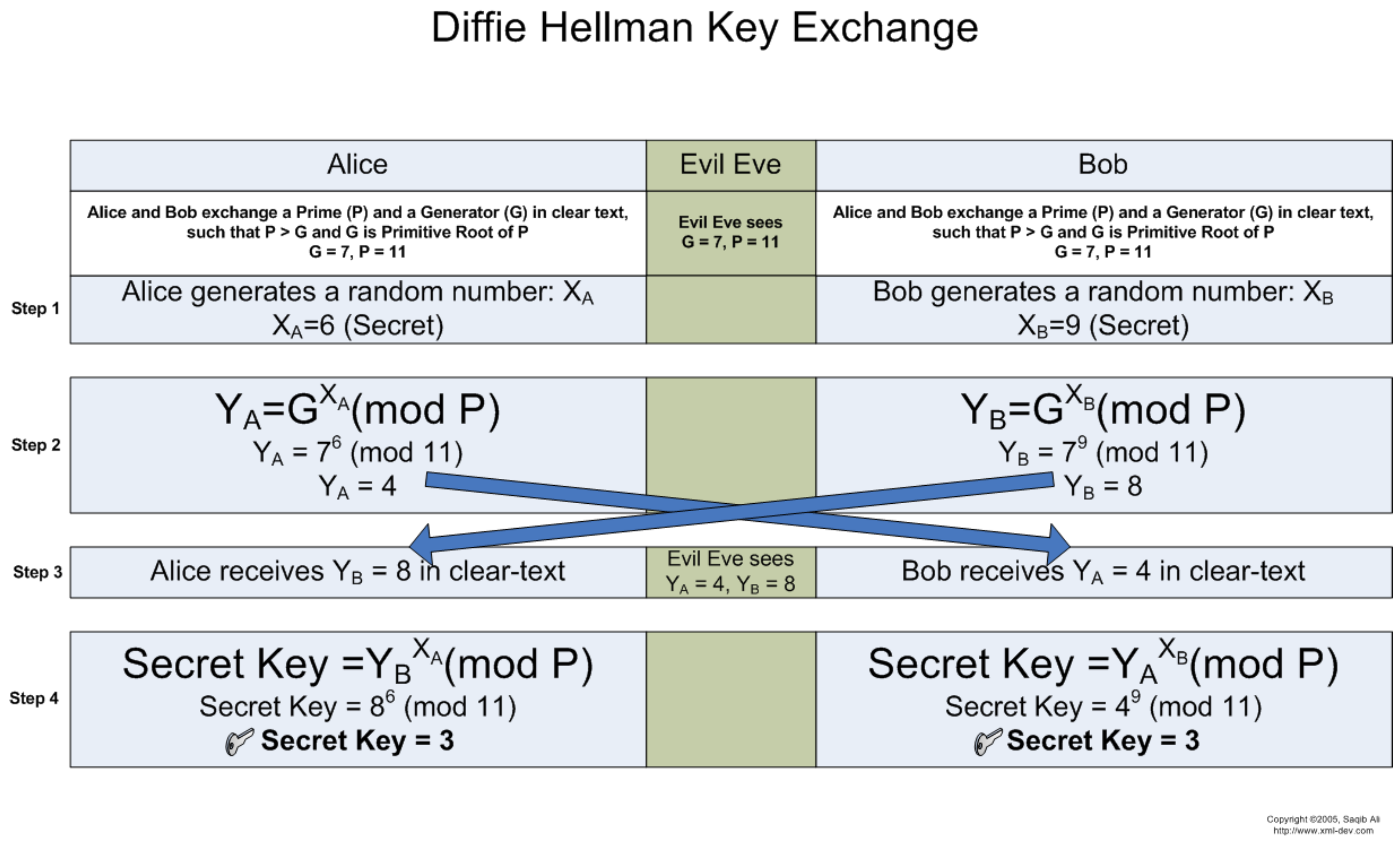
4. Bob menghitung kunci simetri K,

   $K = X^y \bmod n = 50^{58} \bmod 97 = 75$



Jadi, Alice dan Bob sekarang sudah mempunyai kunci enkripsi simetri yang sama, yaitu $K = 75$.

- Contoh lain:



Sumber: http://sspai.com/26497

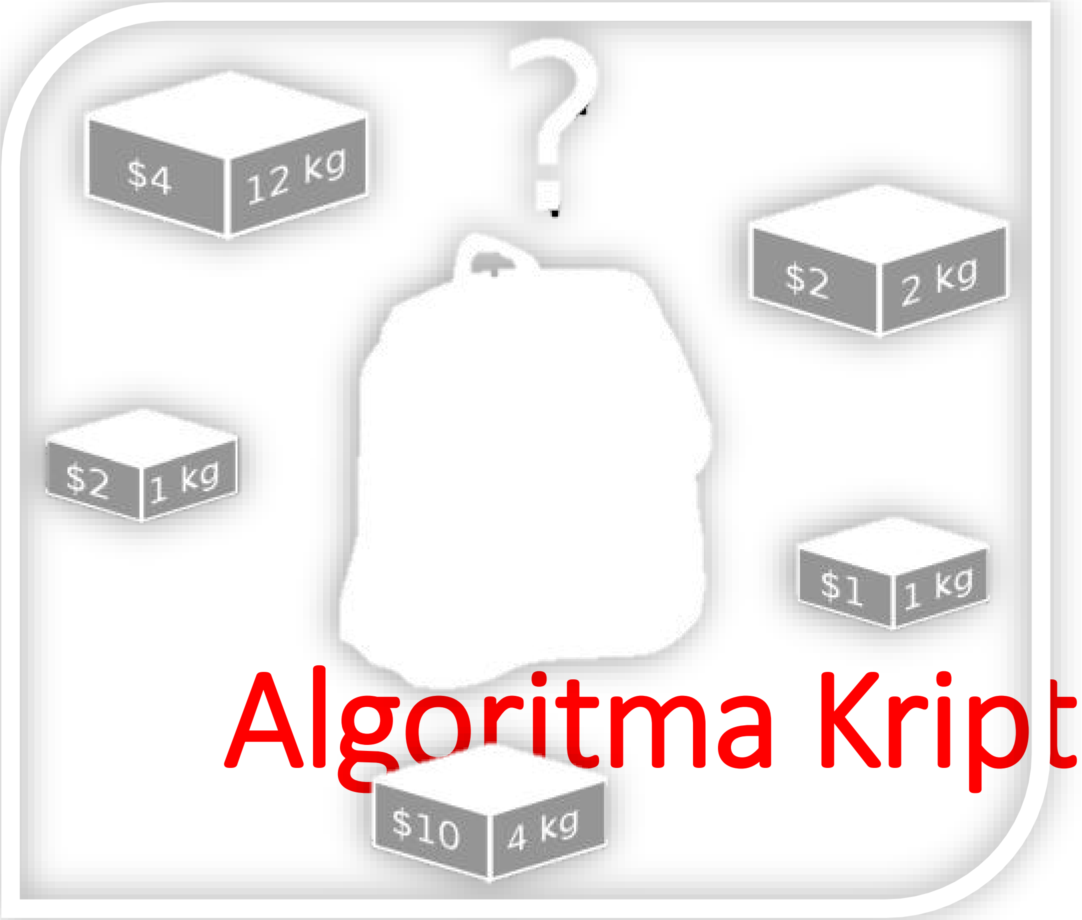
The IEEE Koji Kobayashi Computers and Communications Award

The 1999 award was given to Diffie, Hellman and Merkle for "For the revolutionary invention of public key cryptosystems which form the foundation for privacy, integrity and authentication in modern communication systems."

The 2000 award was given to Rivest, Shamir and Adleman "For the revolutionary invention of the RSA public key cryptosystem which is the first to be widely-adopted."

From left to right: Adi Shamir, Ron Rivest, Len Adleman, Ralph Merkle, Martin Hellman, and Whit Diffie (Picture courtesy of Eli Biham, taken at the presentation on Monday August 21 at Crypto 2000, an IACR conference

Sumber: http://www.merkle.com/merkleDir/KobayashiAward.html



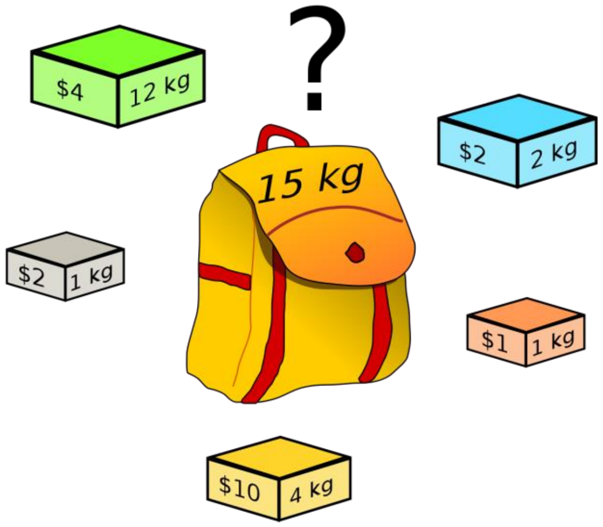
Algoritma Kriptografi Knapsack

Algoritma Kriptografi *Knapsack*

- Merupakan salah satu algoritma kriptografi kunci-publik awal yang ditemukan oleh Ralph Merkle dan Martin Hellman pada 1978.
- Disebut juga algoritma Merkle-Hellman

Merkle, Hellman, dan Diffie

- Algoritma ini didasarkan pada persoalan *Knapsack Problem:*



Diberikan bobot *knapsack* adalah M. Diketahui n buah objek

yang masing-masing bobotnya adalah w_1, w_2, …, w_n. Tentukan nilai b_i sedemikian sehingga

$$M = b_1w_1 + b_2w_2 + \ldots + b_nw_n \qquad (1)$$

yang dalam hal ini, b_i bernilai 0 atau 1. Jika $b_i = 1$, berarti objek i dimasukkan ke dalam *knapsack*, sebaliknya jika $b_i = 0$, objek i tidak dimasukkan.

- Dalam teori algoritma, persoalan *knapsack* termasuk ke dalam kelompok *NP-complete*.

- Persoalan yang termasuk *NP-complete* tidak dapat dipecahkan dalam orde waktu polinomial.

- Ide dasar dari algoritma kriptografi *knapsack* adalah mengkodekan pesan sebagai rangkaian solusi dari persoalan *knapsack*.

- Setiap bobot w_i di dalam persoalan *knapsack* merupakan kunci rahasia, sedangkan bit-bit plainteks menyatakan b_i.

Contoh 1: Misalkan n = 6 dan w_1 = 1, w_2 = 5, w_3 = 6, w_4 = 11, w_5 = 14, dan w_6 = 20.

Plainteks: 111001010110000000011000

Plainteks dibagi menjadi blok yang panjangnya 6, kemudian setiap bit di dalam blok dikalikan dengan w_i yang berkoresponden sesuai dengan persamaan (1):

Blok plainteks ke-1 : 111001

Kriptogram : $(1 \times 1) + (1 \times 5) + (1 \times 6) + (0 \times 11) + (0 \text{ x } 14) + (1 \text{ x } 20) = 32$

Blok plainteks ke-2 : 010110

Kriptogram : $(1 \times 5) + (1 \times 11) + (1 \times 14) = 30$

Blok plainteks ke-3 : 000000

Kriptogram : 0

Blok plainteks ke-4 : 011000

Kriptogram : $(1 \times 5) + (1 \times 6) = 11$

Jadi, cipherteks yang dihasilkan: 32 30 0 11

- Sayangnya, algoritma *knapsack* sederhana di atas hanya dapat digunakan untuk enkripsi, tetapi tidak untuk dekripsi.

- Misalnya, jika diberikan kriptogram = 32, maka tentukan $b_1, b_2, \ldots, b_6$ sedemikian sehingga

$$32 = b_1 + 5b_2 + 6b_3 + 11b_4 + 14b_5 + 20b_6 \qquad (2)$$

- Solusi persamaan (2) ini tidak dapat dipecahkan dalam orde waktu polinomial dengan semakin besarnya n (dengan catatan barisan bobot tidak dalam urutan menaik).

- Namun, hal inilah yang dijadikan sebagai kekuatan algoritma *knapsack*.

Superincreasing Knapsack

- *Superincreasing knapsack* adalah persoalan *knapsack* yang dapat dipecahkan dalam orde $O(n)$ (jadi, polinomial).
- Ini adalah persoalan *knapsack* yang mudah sehingga tidak disukai untuk dijadikan sebagai algoritma kriptografi yang kuat.

- Jika senarai bobot disebut barisan *superincreasing*, maka kita dapat membentuk *superincreasing knapsack*.

- Barisan *superincreasing* adalah suatu barisan di mana setiap nilai di dalam barisan lebih besar daripada jumlah semua nilai sebelumnya.
- Contoh: {1, 3, 6, 13, 27, 52} → barisan *superincreasing,*
  {1, 3, 4, 9, 15, 25} → bukan barisan *superincreasing*

- Solusi dari *superincreasing knapsack* (yaitu b_1, b_2, …, b_n) mudah dicari sebagai berikut (berarti sama dengan mendekripsikan cipherteks menjadi plainteks semula):

1. Jumlahkan semua bobot di dalam barisan.
2. Bandingkan bobot total dengan bobot terbesar di dalam barisan. Jika bobot terbesar lebih kecil atau sama dengan bobot total, maka ia dimasukkan ke dalam *knapsack*, jika tidak, maka ia tidak dimasukkan.
3. Kurangi bobot total dengan bobot yang telah dimasukkan, kemudian bandingkan bobot total sekarang dengan bobot terbesar selanjutnya. Demikian seterusnya sampai seluruh bobot di dalam barisan selesai dibandingkan.
4. Jika bobot total menjadi nol, maka terdapat solusi persoalan *superincreasing knapsack* , tetapi jika tidak nol, maka tidak ada solusinya.

Contoh 2: Misalkan bobot-bobot yang membentuk barisan *superincreasing* adalah {2, 3, 6, 13, 27, 52}, dan diketahui bobot *knapsack* (M) = 70. Kita akan mencari b_1, b_2, …, b_6 sedemikian sehingga

$$70 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_6$$

Caranya sebagai berikut:

1) Bandingkan 70 dengan bobot terbesar, yaitu 52. Karena $52 \leq 70$, maka 52 dimasukkan ke dalam *knapsack*. $\rightarrow b_6 = 1$

2) Bobot total sekarang menjadi 70 – 52 = 18. Bandingkan 18 dengan bobot terbesar kedua, yaitu 27. Karena 27 > 18, maka 27 tidak dimasukkan ke dalam *knapsack*. $\rightarrow b_5 = 0$

3) Bandingkan 18 dengan bobot terbesar berikutnya, yaitu 13. Karena $13 \leq 18$, maka 13 dimasukkan ke dalam *knapsack*. $\rightarrow b_4 = 1$

4) Bobot total sekarang menjadi 18 – 13 = 5.

5) Bandingkan 5 dengan bobot terbesar kedua, yaitu 6. Karena 6 > 5, maka 6 tidak dimasukkan ke dalam *knapsack*. $\rightarrow$ $b_3 = 0$

6) Bandingkan 5 dengan bobot terbesar berikutnya, yaitu 3. Karena $3 \leq 5$, maka 3 dimasukkan ke dalam *knapsack*. $\rightarrow$ $b_2 = 1$

7) Bobot total sekarang menjadi 5 – 3 = 2.

8) Bandingkan 2 dengan bobot terbesar berikutnya, yaitu 2. Karena $2 \leq 2$, maka 2 dimasukkan ke dalam *knapsack*. $\rightarrow$ $b_1 = 0$

9) Bobot total sekarang menjadi 2 – 2 = 0.

Karena bobot total tersisa = 0, maka solusi persoalan *superincreasing knapsack* ditemukan. Barisan bobot yang dimasukkan ke dalam *knapsack* adalah

$$\{2, 3, -, 13, -, 52\}$$

sehingga

$$70 = (1 \times 2) + (1 \times 3) + (0 \times 6) + (1 \times 13) + (0 \times 27) + (1 \times 52)$$

Dengan kata lain, plainteks dari kriptogram 70 adalah 110101.

Algoritma *Knapsack* Kunci-Publik

- Algoritma *superincreasing knapsack* adalah algoritma yang lemah, karena cipherteks dapat didekripsi menjadi plainteksnya secara mudah dalam waktu lanjar.

- Algoritma *non-superincreasing knapsack* atau *normal knapsack* adalah kelompok algoritma *knapsack* yang sulit (dari segi komputasi) karena membutuhkan waktu dalam orde eksponensial untuk memecahkannya.

- Namun, *superincreasing knapsack* dapat dimodifikasi menjadi *non-superincreasing knapsack* dengan menggunakan kunci publik (untuk enkripsi) dan kunci privat (untuk dekripsi).

- Kunci publik merupakan barisan *non-superincreasing* sedangkan kunci privat tetap merupakan barisan *superincreasing*.

- Modifikasi ini ditemukan oleh Martin Hellman dan Ralph Merkle.

- Prosedur membuat kunci publik dan kunci privat:

1. Tentukan barisan *superincreasing*.

2. Kalikan setiap elemen di dalam barisan tersebut dengan *n* (mod *m*)
   (Modulus *m* seharusnya angka yang lebih besar daripada jumlah semua elemen di dalam barisan, sedangkan pengali *n* seharusnya tidak mempunyai faktor persekutuan dengan *m*, atau PBB(n, m) = 1)

3. Hasil perkalian akan menjadi kunci publik sedangkan barisan *superincreasing* semula menjadi kunci privat.

Contoh 3: Misalkan barisan *superincreasing* adalah {2, 3, 6, 13, 27, 52}, dan m = 105, dan n = 31.

Barisan *non-superincreasing* (atau normal) *knapsack* dihitung sbb:

$2 \cdot 31 \bmod 105 = 62$

$3 \cdot 31 \bmod 105 = 93$

$6 \cdot 31 \bmod 105 = 81$

$13 \cdot 31 \bmod 105 = 88$

$27 \cdot 31 \bmod 105 = 102$

$52 \cdot 31 \bmod 105 = 37$

Jadi, kunci publik adalah {62, 93, 81, 88, 102, 37}, sedangkan kunci privat adalah {2, 3, 6, 13, 27, 52}.

Enkripsi

- Enkripsi dilakukan dengan cara yang sama seperti algoritma *knapsack* sebelumnya.

- Mula-mula plainteks dipecah menjadi blok bit yang panjangnya sama dengan kardinalitas barisan kunci publik.

- Kalikan setiap bit di dalam blok dengan elemen yang berkoresponden di dalam barisan kunci publik.

Contoh 4: Misalkan

Plainteks: 011000110101101110

dan kunci publik adalah hasil dari Contoh 3,

Kunci publik = {62, 93, 81, 88, 102, 37},
Kunci privat adalah {2, 3, 6, 13, 27, 52}.

Plainteks dibagi menjadi blok yang panjangnya 6, kemudian setiap bit di dalam blok dikalikan dengan elemen yang berkorepsonden di dalam kunci publik:

Blok plainteks ke-1 : 011000

Kunci publik : 62, 93, 81, 88, 102, 37

Kriptogram : $(1 \times 93) + (1 \times 81) = 174$

Blok plainteks ke-2 : 110101

Kunci publik : 62, 93, 81, 88, 102, 37

Kriptogram : $(1 \times 62) + (1 \times 93) + (1 \times 88) +$
$(1 \times 37) = 280$

Blok plainteks ke-3 : 101110

Kunci publik : 62, 93, 81, 88, 102, 37

Kriptogram : $(1 \times 62) + (1 \times 81) + (1 \times 88) +$
$(1 \times 102) = 333$

Jadi, cipherteks yang dihasilkan : 174, 280, 333

Dekripsi

- Dekripsi dilakukan dengan menggunakan kunci privat.
- Mula-mula penerima pesan menghitung n^{-1} , yaitu balikan dari n modulo m, sedemikian sehingga

$$n \cdot n^{-1} \equiv 1 \pmod{m}.$$

- Kalikan setiap kriptogram dengan n^{-1} , lalu nyatakan hasil kalinya sebagai penjumlahan elemen-elemen kunci privat untuk memperoleh plainteks dengan menggunakan algoritma pencarian solusi *superincreasing knapsack*.

Contoh 5: Kita akan mendekripsikan cipherteks dari Contoh 4 dengan menggunakan kunci privat {2, 3, 6, 13, 27, 52}. Di sini, n = 31 dan m = 105.

Nilai 31^{-1} (mod 105) diperoleh sbb:

$$n \cdot n^{-1} \equiv 1 \pmod{m} \rightarrow 31 \cdot n^{-1} \equiv 1 \pmod{105} \rightarrow n^{-1} = (1 + 105k)/31$$

coba k = 0, 1, 2, …, diperoleh n^{-1} = 61

Cipherteks dari Contoh 4 adalah 174, 280, 333. Hasil dekripsi:

$174 \cdot 61 \bmod 105 = 9 = 0 \cdot 2 + 1 \cdot 3 + 1 \cdot 6 + 0 \cdot 13 + 0 \cdot 27 + 0 \cdot 52 \rightarrow 011000$

$280 \cdot 61 \bmod 105 = 70 = 1 \cdot 2 + 1 \cdot 3 + 0 \cdot 6 + 1 \cdot 13 + 0 \cdot 27 + 1 \cdot 52 \rightarrow 110101$

$333 \cdot 61 \bmod 105 = 48 = 1 \cdot 2 + 0 \cdot 3 + 1 \cdot 6 + 1 \cdot 13 + 1 \cdot 27 + 0 \cdot 52 \rightarrow 101110$

Jadi, plainteks yang dihasilkan kembali adalah:

011000110101101110

Implementasi Knapsack

- Ukuran cipherteks yang dihasilkan lebih besar daripada plainteksnya, karena enkripsi dapat menghasilkan kriptogram yang nilai desimalnya lebih besar daripada nilai desimal blok plainteks yang dienkripsikan.

- Untuk menambah kekuatan algoritma *knapsack*, kunci publik maupun kunci privat seharusnya paling sedikit 250 elemen, nilai setiap elemen antara 200 sampai 400 bit panjangnya, nilai modulus antara 100 sampai 200 bit.

- Dengan nilai-nilai *knapsack* sepanjang itu, dibutuhkan 10^{46} tahun untuk menemukan kunci secara *brute force*, dengan asumsi satu juta percobaan setiap detik.

Keamanan Knapsack

- Sayangnya, algoritma knapsack dinyatakan sudah tidak aman, karena *knapsack* dapat dipecahkan oleh pasangan kriptografer Shamir dan Zippel.

- Mereka merumuskan transformasi yang memungkinkan mereka merekonstruksi *superincreasing knapsack* dari *normal knapsack*.

Referensi utama :

>> Michael Felderer , Riccardo Scandariato (editor) - Exploring Security in Software Architecture and Design, 2018.

>> Nancy R. Mead, Carol Woody - Cyber Security Engineering_ A Practical Approach for Systems and Software Assurance-Addison-Wesley Professional (2016)

>> James Helfrich - Security for Software Engineers-CRC Press (2019)

>> Pete Loshin - Simple Steps to Data Encryption_ A Practical Guide to Secure Computing-Syngress (2013)

>> Tevfik Bultan,Fang Yu,Muath Alkhalaf,Abdulbaki Aydin (auth.) - String Analysis for Software Verification and Security (2017

Ada pertanyaan?