Keamanan pada Sistem Terdistribusi

Budi Susanto
budsus@ukdw.ac.id

Pengantar

- Bagaimana kita dapat melindungi transaksi dalam suatu sistem terdistribusi?
  - Dapatkah kita melindungi pengiriman data?
  - Dapatkan kita mensetup saluran yang aman untuk komunikasi?
  - Dapatkan kita menentukan pengirim data?
- Kebutuhan untuk melindungi kesatuan dan rahasia informasi dan sesumber lain yang dimiliki oleh individu ataupun organisasi dapat meliputi keamanan fisik maupun data digital.
  - Kebutuhan ini muncul karena sesumber tersebut digunakan bersama

Perkembangan Kebutuhan Keamanan

	1965-75	*1975-89*	*1990-99*	*Current*
Platforms	Multi-user timesharing computers	Distributed systems based on local networks	The Internet, wide-area services	The Internet + mobile devices
Shared resources	Memory, files	Local services (e.g. NFS), local networks	Email, web sites, Internet commerce	Distributed objects, mobile code
Security requirements	User identification and authentication	Protection of services	Strong security for commercial transactions	Access control for individual objects, secure mobile code
Security management environment	Single authority, single authorization database (e.g. /etc/ passwd)	Single authority, delegation, repli-cated authorization databases (e.g. NIS)	Many authorities, no network-wide authorities	Per-activity authorities, groups with shared responsibilities

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3

Ancaman dan Serangan

- Tujuan utama dengan adanya keamanan adalah untuk membatasi akses informasi dan sesumber hanya untuk pemakai yang memiliki hak akses.
- Ancaman keamanan:
  - **Leakage** (Kebocoran) : pengambilan informasi oleh penerima yang tidak berhak
  - **Tampering** : pengubahan informasi yang tidak legal
  - **Vandalism** (perusakan) : gangguan operasi sistem tertentu. Si pelaku tidak mengharap keuntungan apapun.
- Serangan pada sistem terdistribusi tergantung pada pengkasesan ke saluran komunikasi yang ada atau membuat saluran baru yang menyamarkan (masquerade) sebagai koneksi legal
- **Penyerangan Pasive**, Hanya mengamati komunikasi atau data
- **Penyerangan Aktif,** Secara aktif memodifikasi komunikasi atau data
  - Pemalsuan atau pengubahan Email
  - TCP/IP Spoofing

Metode Penyerangan

Klasifikasi metode penyerangan :

- **Eavesdropping**, mendapatkan duplikasi pesan tanpa ijin
- **Masquerading,** Mengirim atau menerima pesanmenggunakan identitas lain tanpa ijin mereka
- **Message tampering**,
  - Mencegat atau menangkap pesan dan mengubah isinya sebelum dilanjutkan ke penerima sebenarnya. "*man-in-the-middle attack*" adalah bentuk message tampering dengan mencegat pesan pertama pada pertukaran kunci enkripsi pada pembentukan suatu saluran yang aman. Penyerang menyisipkan kunci lain yang memungkinkan dia untuk mendekrip pesan berikutnya seelum dienkrip oleh penerima
- **Replaying**, menyimpan pesan yang ditangkap untuk pemakaian berikutnya.
- **Denial of Service**, membanjiri saluran atau sesumber lain dengan pesan yang bertujuan untuk menggagalkan pengaksesan pemakai lain

Ancaman dari Mobile Code

- Mobile Code
  - Kode yang diload ke sebuah proses dari remote server dan dijalankan secara lokal
  - Contoh : Kode Java Applet
- JVM Security :
  - Setiap lingkungan eksekusi java mobile code diatur oleh Security Manager. Contoh : Applet tidak diijinkan untuk menulis atau membaca file lokal
  - Mobile code yang didownload disimpan secara terpisah dari kelas-kelas lokal, untuk mencegah penimpaan
  - Bytecode dicek untuk validasi. Instruksi pada bytecode juga dicek, apakah akan menyebabkan suatu kesalahan fatal, misal mengakses alamat memori yang tidak sah.
    - http://www.cs.cornell.edu/html/cs513-sp98/L17.html
    - http://www.cs.cornell.edu/html/cs513-sp98/L18.html

Mengamankan transkasi elektronis

- Keamanan sangat dibutuhkan pada kebanyak transaksi
  - E-commerce
  - Banking
  - E-mail
    - Bagaimana mengamankan isi email dan pengirim atau penerima pesan harus dapat memastikan satu sama lain
- Transaksi elektronik dapat aman jika dilindungi dengan kebijakan dan mekanisme keamanan.
- Contoh : Pembeli harus dilindungi terhadap penyingkapan kode credit number selama pengiriman dan juga terhadap penjual yang tidak bersedia mengirim barang setelah menerima pembayaran. Vendor harus mendapatkan pembayaran sebelum barang dikirim, sehingga perlu dapat memvalidasi calon pembeli sebelum memberi mereka akses

Perancangan Sistem yang Aman

- Merupakan tugas yang sulit!
  - Tujuannya adalah mencegah semua serangan yang saat ini diketahui ataupun yang akan datang
    - Rancangan mengikuti standard yang ada
    - Mendemokan validasi melawan ancaman yang diketahui
    - Audit terhadap kegagalan yang terdeteksi
  - Ada keseimbangan antara biaya terhadap serangan yang ada
- Beberapa rancangan yang buruk:
  - Antarmuka dibuka
  - Jaringan tidak aman
  - Membatasi waktu dan ruang lingkup setiap kunci rahasia
  - Algoritma dan kode program tersedia bagi penyerang
  - Penyerang memiliki akses ke sesumber
  - Meminimalkan komputer yang menjadi inti implementasi sistem

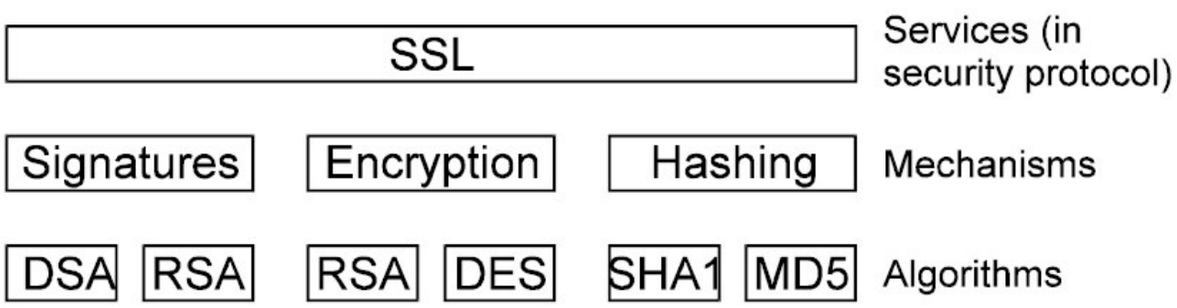
Kebijakan dan Mekanisme

- Pada keamanan fisik, akan menggunakan :
  - **Kebijakan/Layanan keamanan**
    - Aturan yang mengatur pengaksesan ataupun berbagi sesumber yang tersedia
    - Menyangkut apa saja yang diterapkan
    - Contoh : Kebijakan keamanan untuk suatu dokumen : hanya diperbolehkan sekelompok pegawai untuk mengakses
  - **Mekanisme keamanan**
    - Kebijakan dapat dijalankan dengan bantuan mekanisme keamanan
    - Menyangkut bagaimana menerapkannya
    - Contoh : mengakses dokumen dikontrol dengan distribusi yang terbatas dan tersembunyi
- Pemisahan antara kebijakan dan mekanisme keamanan akan membantu memisahkan kebutuhan implementasinya
  - Kebijakan menspesifikasikan kebutuhan
  - Mekanisme menerapkan spesifikasi kebijakan tersebut

- **Security Service**, menurut definisi OSI :
  - **Access Control**, Perlindungan terhadap pemakaian tak legak
  - **Authentication**, Menyediakan jaminan identitas seseorang
  - **Confidentiality** (kerahasiaan), Perlindungan terhadap pengungkapan identitas tak legak
  - **Integrity**, Melindungi dari pengubahan data yang tak legak
  - **Non-repudiation** (penyangkalan), Melindungi terhadap penolakan terhadap komunikasi yang sudah pernah dilakukan
- Tiga dasar **Mekanisme Keamanan** yang dibangun :
  - **Enkripsi**
    - Digunakan untuk menyediakan kerahasiaan, dapat menyediakan authentication dan perlindungan integritas
  - **Digital Signature**
    - Digunakan untuk menyediakan authentication, perlindungan integritas, dan non-repudiation
  - **Algoritma Checksum/Hash**
    - Digunakan untuk menyediakan perlindungan integritas, dan dapat menyediakan authentication
  - Satu atau lebih mekanisme dikombinasikan untuk menyediakan security service

Service, Mekanisme dan Algoritma

- Sebuah protokol keamanan menyediakan satu atau lebih layanan (service)
- Service dibangun dari satu atau lebih mekanisme
- Mekanisme diterapkan dengan menggunakan suatu algoritma



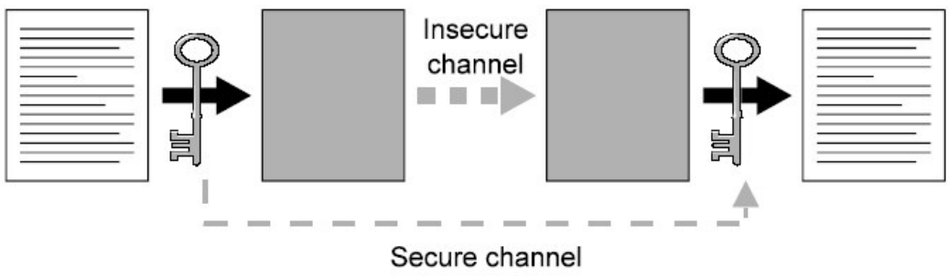
Teknik Keamanan

- Enkripsi adalah proses pengkodean pesan untuk menyembunyikan isi
- Algoritma enkripsi modern menggunakan kunci (*key*).
- Kunci kriptografi adalah parameter yang digunakan dalam algoritma enkripsi dimana hasil enkripsi tidak dapat didekripsi jika tanpa kunci yang sesuai
- Ada dua tipe algoritma enkripsi :
  - **Shared secret key**
    - Pengirim dan penerima harus berbagi kunci dan tidak diberikan kepada orang lain.
  - **public/private key pair**
    - Pengirim pesan menggunakan public key (kunci yang dipublikasikan ke penerima) untuk mengenkrip pesan
    - Penerima menggunakan private key yang cocok (miliknya) untuk mendekrip pesan.

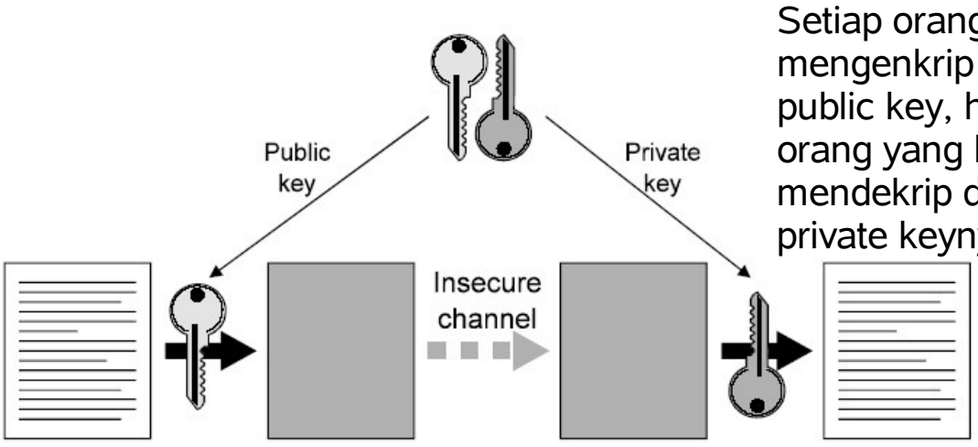
Manajemen Kunci

- Dua kelas kunci
  - **Short term session keys**
    - Disebut juga dengan ephemeral keys
    - Secara umum dibuat secara otomatis dan tidak tampak
    - Digunakan untuk satu pesan atau session dan dibuang
  - **Long term keys**
    - Dihasilkan secara eksplisit oleh pemakai
- Long term keys digunakan untuk dua tujuan
  - **Authentication**
    - Termasuk kontrol akses, integritas, dan non-repudiation
  - **Confidentiality (enkripsi)**
    - Membuat session key
    - Melindungi data yang tersimpan

Enkripsi Shared Key

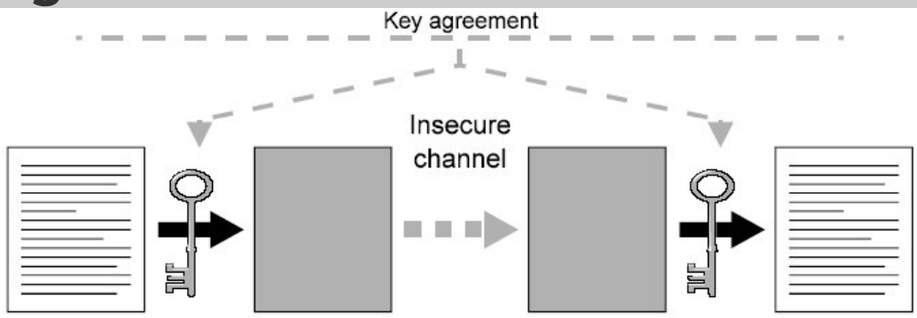


Enkripsi Public/Private Key

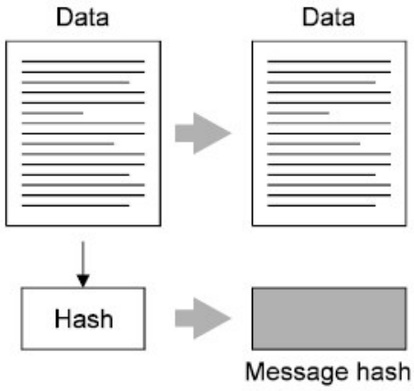


Setiap orang dapat mengenkrip dengan public key, hanya satu orang yang bisa mendekrip dengan private keynya

Key Agreement



- Memungkinkan dua pemakai menyetujui sebuah *shared key*
- Menyediakan bagian dari saluran aman untuk mempertukarkan sebuah kunci konvensional terenkripsi



Fungsi Hash

- Membuat sebuah "fingerprint" unik untuk sebuah pesan
- Tiap orang dapat mengubah data dan menghitung nilai hash baru
- Hash harus dilindungi

Notasi Kriptografi

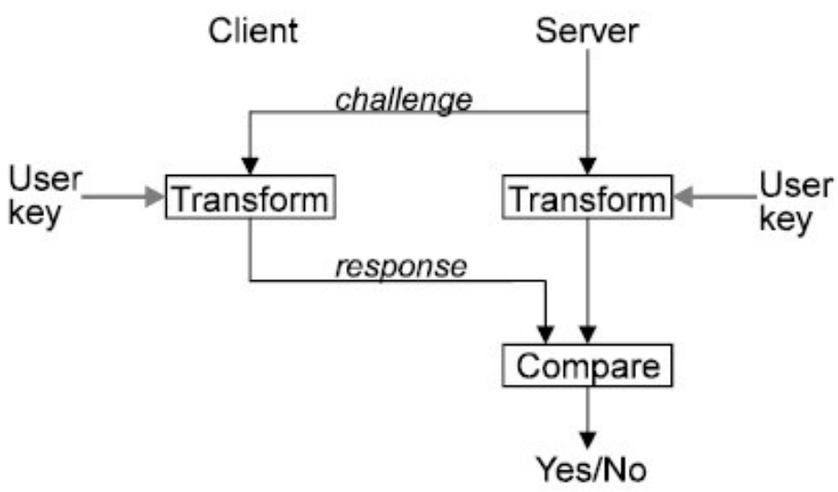
K_A	A secret key
K_B	B secret key
K_{AB}	Secret key shared between A and B
K_{Apriv}	A's private key (known only to A)
K_{Apub}	A's public key (published by A for all to read)
$\{M\}_K$	Message M encrypted with key K
$[M]_K$	Message M signed with key K

Peran Pokok Kriptografi

- **Kerahasiaan dan integritas**
  - komunikasi rahasia dengan shared secret key
  - A menginginkan mengirim informasi ke B menggunakan *shared secret key* K_{AB}
  - B membaca pesan terenkrip menggunaka fungsi dekrip yang sesuai $D(K_{AB}, M)$
  - Masalah yang dapat muncul:
    - Bagaimana A dapat mengirim shared key K_{AB} ke B dengan aman?
    - Bagaimana B mengetahui bahwa sembarang {M*i*} bukanlah duplikasi pesan terenkrip sebelumnya dari A yang diambil oleh M dan dikembalikan kemudian?

Peran Pokok Kriptografi

- **Authentication**
  - Kriptografi digunakan untuk mendukung mekanisme authentication antar sepasang node
  - Seorang pelaku jika berhasil mendekrip pesan menggunakan suatu kunci yang cocok, berarti pesan tersebut asli (*authentic*) dari pengirim yang sebenarnya.
  - Dikembangkan pertama kali oleh Roger Needham dan Michael Schroeder



Skenario 1 : Authentication dengan server

- **A** ingin mengakses file yang disimpan oleh **B** pada file server. **S** adalah *authentication server* yang memberikan A dan B password dan menyimpan *secret key* untuk semua pemakai dalam sistem.
- Sebagai contoh S mengetahui kunci A (K_A) dan B (K_B). Kunci ini dikenal dengan nama *tiket*. Sebuah tiket adalah item terenkrip yang dikeluarkan oleh *authentication server* yang berisi identitas pemakai dan *shared key* yang dihasilkan untuk *session* komunikasi.
  1. A mengirim sebuah pesan (tidak terenkrip) ke S untuk meminta tiket untuk mengakses B ($A \rightarrow S: A, B, N_A$)
  2. S mengirim ke A yang terenkrip dengan K_A yang berisi tiket yang dienkrip dengan K_B dan kunci baru K_{AB} untuk digunakan ketika berkomunikasi dengan B. $\{\{Tiket\}K_B, K_{AB}\}K_A$
     - (S -> A : $\{N_A, B, K_{AB}, \{K_{AB}, A\}K_B\}K_A$)

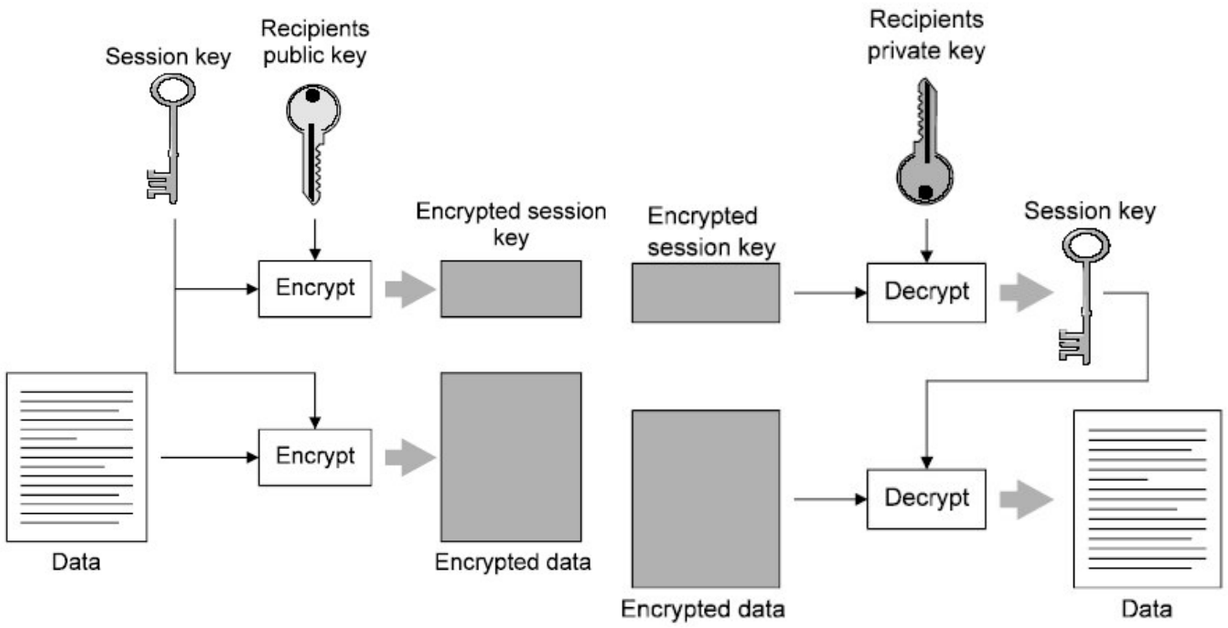
Skenario 1 : Authentication dengan server

3. A mendekrip response tersebut menggunakan K_A (yang dihasilkan dari passwordnya, password tidak dikirimkan pada jaringan). Jika A memiliki kunci yang benar, dia mendapat tiket yang valid untuk mengakses B dan kunci enkripsi baru untuk berkomunikasi dengan B.
4. A mengirim tiket ke B bersamaan dengan identitas A dan meminta R untuk mengakses file : $\{K_{AB}, A\}K_B$
5. B dekrip tiket menggunakan kuncinya K_B. B mendapat identifikasi A yang sah dan *Shared Key* baru K_{AB} (*session key*) untuk berinteraksi dengan A. B akan mengenkrip pesan saat itu (N_B) dengan kunci K_{AB}. $\{N_B\}K_{AB}$.
6. A mengirim balasan ke B dengan menggunakan kunci K_{AB}. $\{N_B\text{-}1\}K_{AB}$.

Skenario 2 : Authentication dengan public key

- Diasumsikan B menghasilkan *public key* dan *private key*.
  - A mengakses layanan distribusi kunci untuk mendapatkan ***public key certificate*** yang mengandung public key B. Disebut *certificate* oleh karena adanya tanda tangan oleh yang berwewenang (seseorang atau organisasi yang diketahui menyediakan Public Key - pemiliknya). Setelah di cek tanda tangannya, A membaca public key B (K_{Bpub}) dari sertifikat.
  - A membuat shared key baru (K_{AB}) dan mengenkripnya menggunakan K_{Bpub} dengan algoritma public key. A mengirim hasilnya ke B, dengan suatu nama unik yang menunjukkan pasangan public/private key. Sehingga A mengirim *keyname,* $\{K_{AB}\}K_{Bpub}$ ke B.
  - B menggunakan private key K_{Bpriv} untuk mendekrip K_{AB}.
- Ilustrasi di atas menggambarkan kriptografi public key untuk mendistribusikan *shared secret key*. Teknik ini disebut *hybrid cryptographic protocol*.

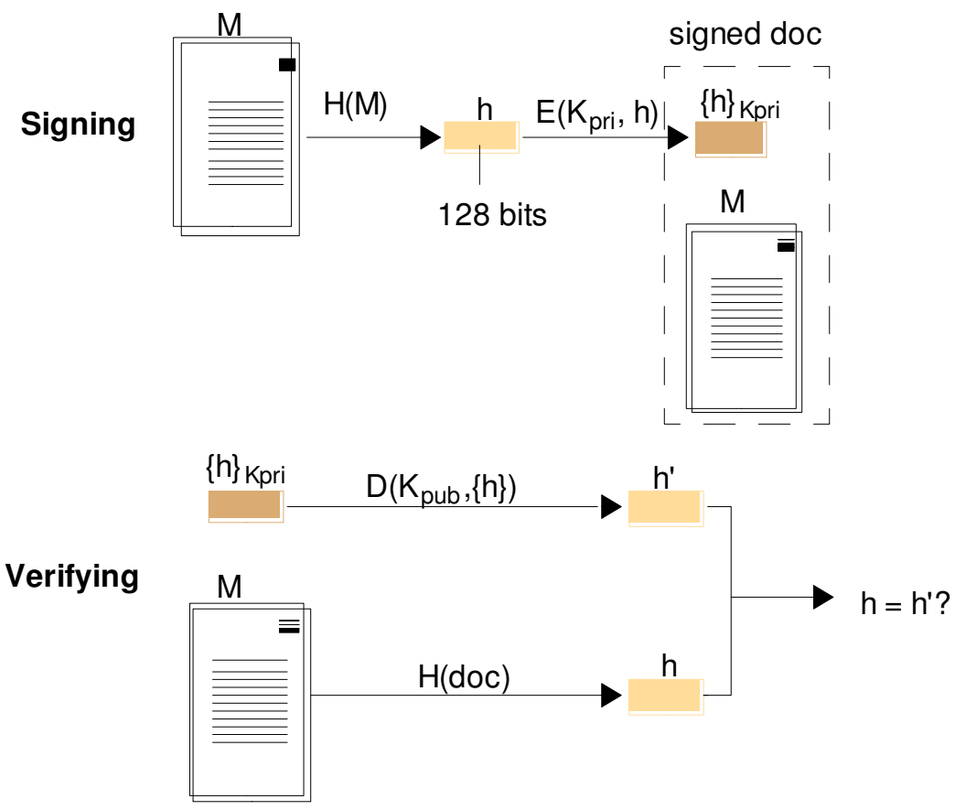
Enkripsi Data



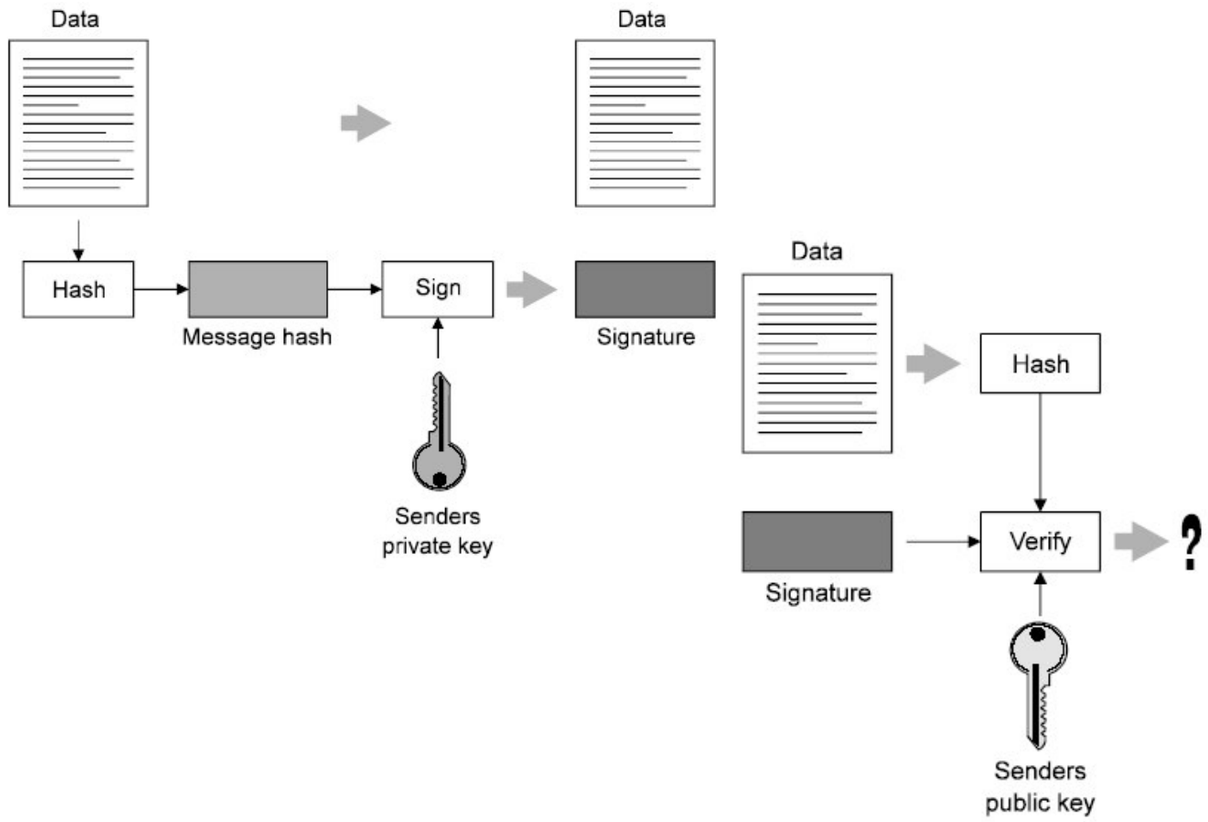
Peran Pokok Kriptografi

- **Digital Signature**
  - Didasarkan pada suatu ikatan tanda (yang tak dapat dirubah) ke suatu pesan atau dokumen yang hanya diketahui oleh si penandatangan.
  - Hal ini dapat dicapai dengan cara mengenkrip sebuah pesan terkompresi (***digest***) dengan menggunakan private key
  - Digest memiliki ukuran yang tetap yang dihasilkan dari sebuah *secure digest function*.
- A ingin menandatangani dokumen M, sehingga penerima dapat yakin bahwa M adalah berasal dari A.
  - A menghitung digest dokumen dengan fungsi *Digest(M)*.
  - A mengenkrip digest dengan private keynya, dan ditambahkan ke *M*, sehingga menghasilkan $\{Digest(M)\}K_{Apriv}$.
  - B menerima dokumen tersebut dan mengambil *M* dan menghitung *Digest(M)*.
  - B mendekrip dengan $\{Digest(M)\}K_{Apriv}$ menggunakan K_{Apub} dan membandingkan isinya dengan hasil perhitungan *Digest(M)*. Jika sama, tandatangan adalah valid.

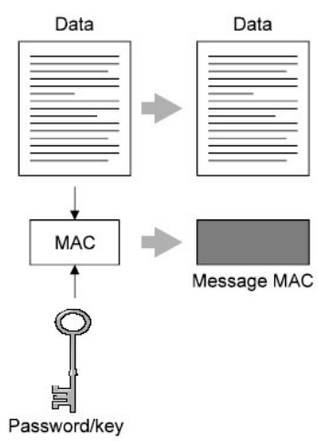
Digital Signature dengan public key

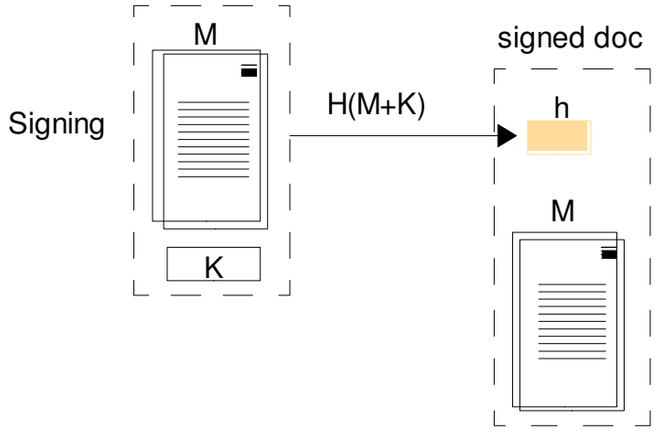


Digital Signature

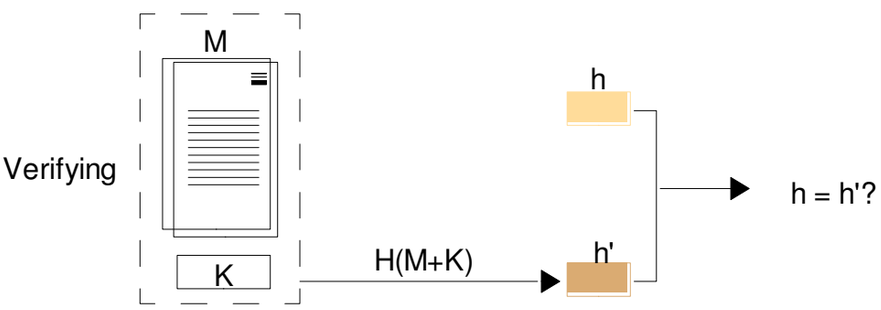


MAC (Message Authentication Code)





- Menghasilkan random password/key untuk suatu hash
- Hanya pemegang password yang dapat menghasilkan MAC



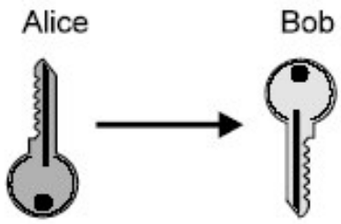
Digest Function

- Sebuah digest function yang aman harus (sering disebut dengan *one-way hash function*):
  - Diberikan *M*, mudah untuk menghitung *h*
  - Diberikan *h*, sangat sulit untuk menghitung *M*
  - Diberikan *M*, sangat sulit untuk menemukan pesan *M'* lain : *H(M) = H(M')*.
- MD5 (Message Digest 5) oleh Rivest (1991)
  - Dapat ditemukan di RFCs 1319-1321
  - Panjang digest : 128 bit
- SHA (Secure Hash Algorithm)
  - Panjang digest : 160 bit
  - Didasarkan pada algoritma MD4

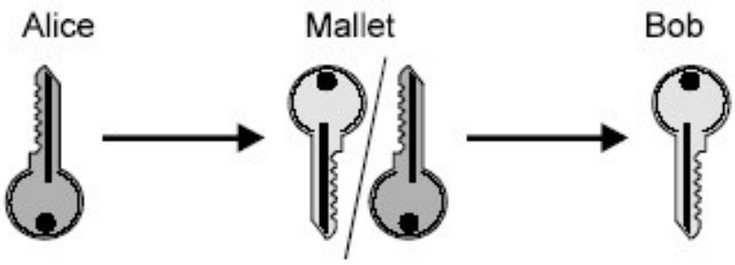

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class ContohDigest {
  public static void main(String args[])
    throws NoSuchAlgorithmException {
    MessageDigest sha =
         MessageDigest.getInstance("SHA");
    sha.update("Sistem Terdistribusi".getBytes());
    byte[] shaHash = sha.digest();
    System.out.println(new String(shaHash));

    MessageDigest md5 =
         MessageDigest.getInstance("MD5");
    md5.update("Sistem Terdistribusi".getBytes());
    byte[] md5Hash = md5.digest();
    System.out.println(new String(md5Hash));
  }
}
```

Masalah pada distribusi kunci

- Alice memelihara private key dan mengirim public key ke Bob



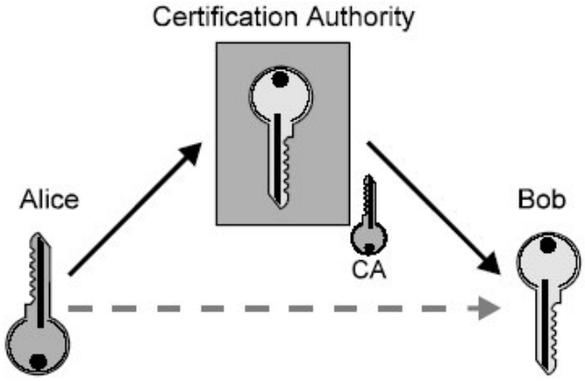
- Mallet mencegat kunci tersebut dan mengganti dengan miliknya



- Mallet dapat mendekrip dan menghasilkan tanda tangan palsu atau mengubah datanya

Solusinya...

- Sebuah Certification Authority (CA) dapat memecahkan masalah tersebut



- CA menandatangani kunci Alice untuk menyakinkan Bob. Mallet tidak dapat mengganti dengan kuncinya selama CA tidak bersedia menandatanganinya

Certificate

- Sertifikat digital adalah dokumen yang berisi pernyataan ditandatangani oleh pemegangnya. Contoh:
  1. B adalah bank. Nasabah ingin menyakinkan ketika menghubungi bahwa B adalah bank yang dimaksud. B perlu untuk mengesahkan (*authenticate*) pelanggannya sebelum memberikan nasabah akses ke account mereka
  2. A dapat menggunakan sertifikate tersebut ketika berbelanja untuk menjamin bahwa A punya rekening di bank B. Sertifikat ditandatangani dengan K_{Bpriv}.

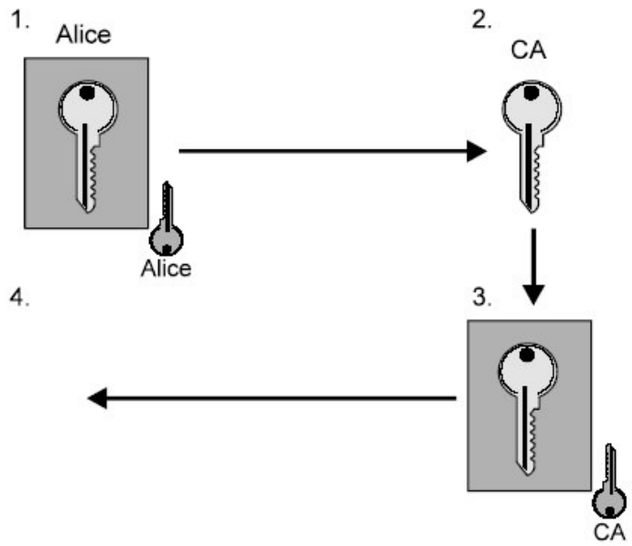
1. *Certificate type*	**Account number**
2. *Name*	**A**
3. *Account*	**6262626**
4. *Certifying authority*	**B Bank**
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

3. Sebuah toko C dapat menerima sertifikat untuk mendebet rekening A. Untuk itu C perlu K_{Bpub} untuk menyakinkan bahwa sertifikat tersebut sah, yaitu well-known dan terpercaya.
4. Misal, F adalah Bank Indonesia, sebagai pemberi otoritas (*certifying authority*), yang akan memberikan *public-key certificate* kepada B. Tentu saja keaslian sertifikat dari F tersebut didasarkan pada K_{Fpub}.

1. *Certificate type*	**Public key**
2. *Name*	**Bob's Bank**
3. *Public key*	K_{Bpub}
4. *Certifying authority*	**Fred – The Bankers Federation**
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

5. Sehingga C dapat yakin akan sertifikat dari F jika C memiliki K_{Fpub}. (di sini terjadi rekursi)

Mendapatkan Sertifikat



1. Alice menghasilkan pasangan kunci dan menandatangani public key dan informasi ID dengan private key
2. CA memeriksa tandatangan Alice pada public key dan informasi ID
   - Dapat melalui email, telepon
3. CA menandatangani public key dan informasi ID dengan kunci CA untuk membuat sertifikat
   - CA telah mensahkan public key dan ID
4. Alice memeriksa kunci, ID dan tandatangan CA
   - Menyakinkan bahwa CA tidak mengubah keu dan ID
   - Melindungi sertifikat selama pengiriman
5. Alice dan/atau CA mempublishkan sertifikat

Format Sertifikat Standard

- Untuk membuat sertifikat bermanfaat, diperlukan :
  - Format standard dan representasi sehingga pembuat sertifikat dan pemakai dapat menyusun dan menterjemahkannya
  - Persetujuan terhadap cara urutan pembuatan sertifikat
- Format sertifikat pertama kali yang digunakan X.509 untuk melindungi mekanisme pengaksesan server direktori (X.500 – saat ini dikenal LDAP)
- Untuk pemrograman Sertifikat pada Java, silahkan kunjungi tutorial di
  - http://www.onjava.com/pub/a/onjava/2001/05/03/java_security.html

Contoh dengan JSSE

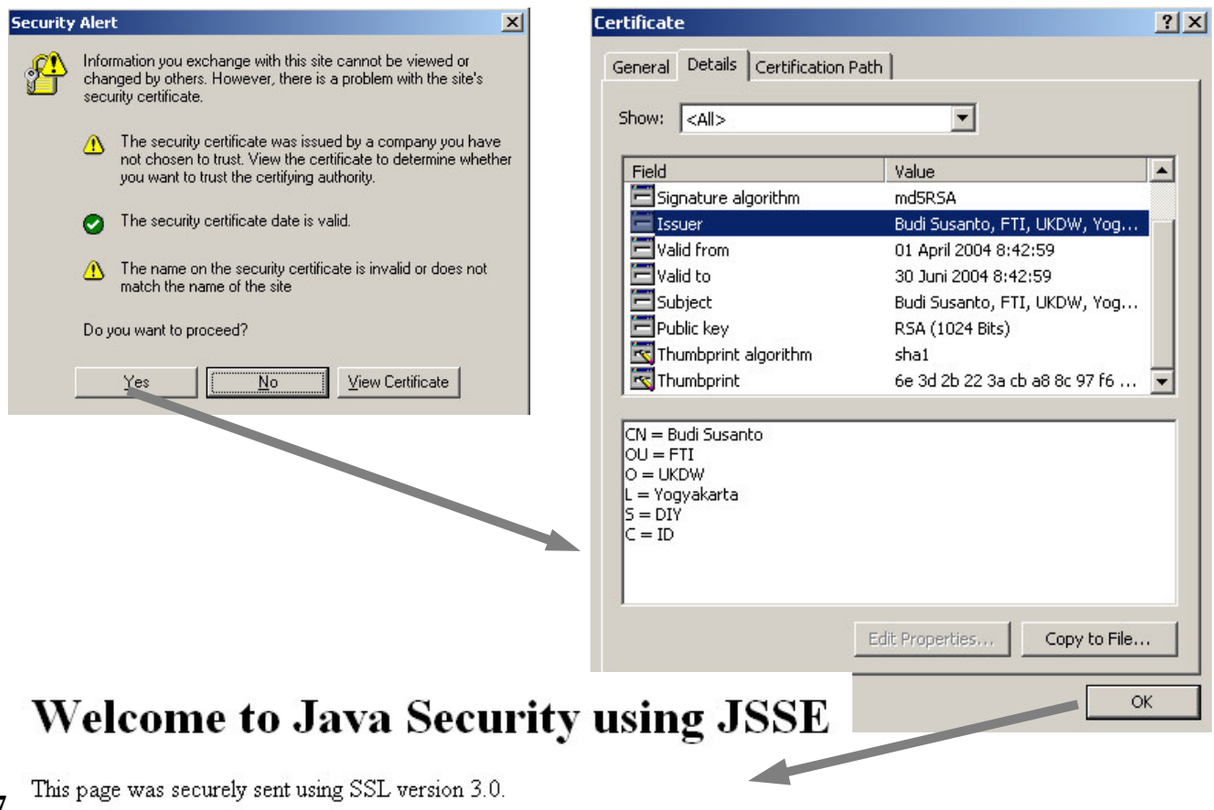
- Pembuatan sertifikat di server

```
$ keytool -genkey -keystore certs -keyalg rsa -alias
  budsus -storepass serverpwd -keypass serverpwd
What is your first and last name?
  [Unknown]:  Budi Susanto
What is the name of your organizational unit?
  [Unknown]:  FTI
What is the name of your organization?
  [Unknown]:  UKDW
What is the name of your City or Locality?
  [Unknown]:  Yogyakarta
What is the name of your State or Province?
  [Unknown]:  DIY
What is the two-letter country code for this unit?
  [Unknown]:  ID
Is CN=Budi Susanto, OU=FTI, O=UKDW, L=Yogyakarta, ST=DIY,
  C=ID correct?
  [no]:  y
```

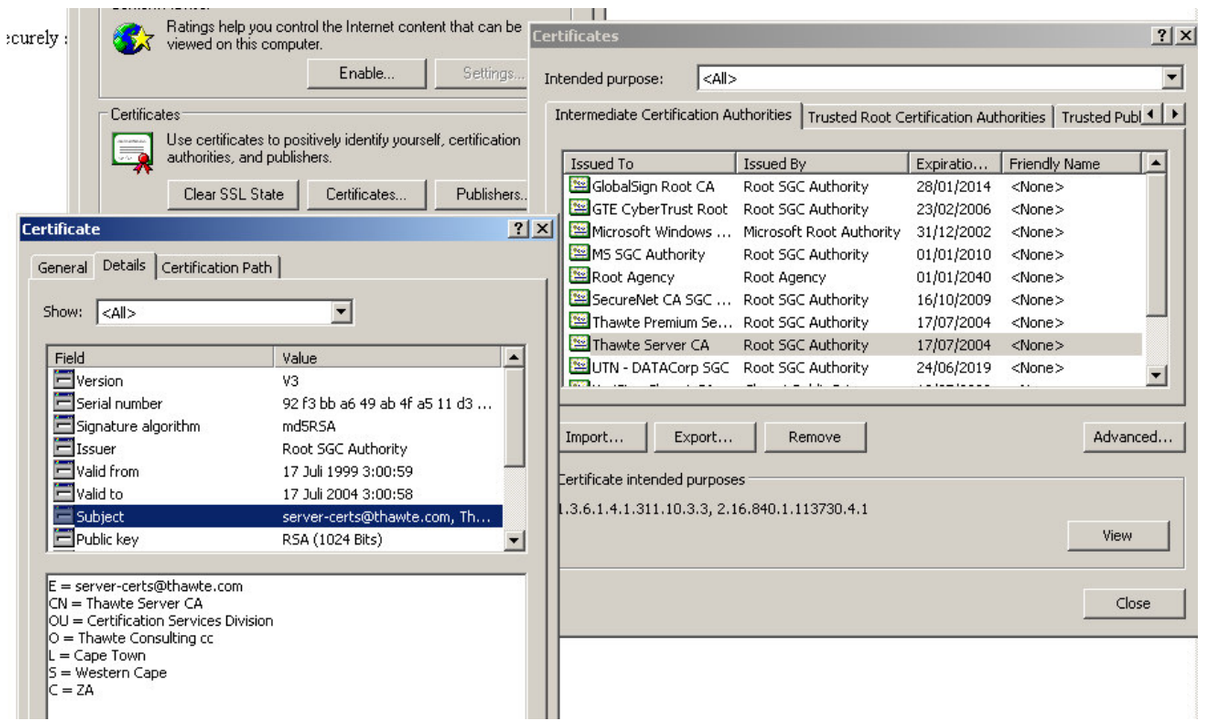

Contoh dengan JSSE

- Untuk mengimport sertifikat server

```
$ keytool -import -keystore jssecacerts -alias budsus
  -file server.cer
Enter keystore password:  12345678
Owner: CN=Budi Susanto, OU=FTI, O=UKDW, L=Yogyakarta, ST=DIY, C=ID
Issuer: CN=Budi Susanto, OU=FTI, O=UKDW, L=Yogyakarta, ST=DIY, C=ID
Serial number: 406aff78
Valid from: Thu Apr 01 00:27:20 GMT+07:00 2004 until: Wed Jun 30
  00:27:20 GMT+07
:00 2004
Certificate fingerprints:
         MD5:  09:2B:EF:29:9C:F6:97:ED:9D:80:A5:2C:D0:D1:4A:B3
         SHA1:
  1F:DC:E9:72:DD:4F:51:71:6A:A1:E1:F4:BB:A1:1C:3C:AA:44:13:99
Trust this certificate? [no]:  y
Certificate was added to keystore
```



Penanganan Sertifikat pada IE



Algoritma Kriptografi

- Pesan M (*plaintext*) di enkodekan dengan fungsi E dan sebuah kunci K untuk menjadi *ciphertext*.

  $E(K,M) = \{M\}_K$

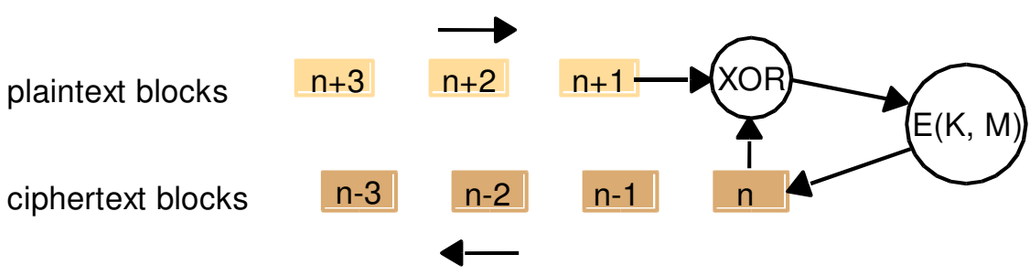
- Pesan didekripsi dengan menggunakan fungsi D dan kunci L

  $D(K,E(K,M)) = M$

- Algoritma kriptografi :
  - **Symmetric (secret-key)**
    - Kunci yang sama digunakan di kedua algoritma
    - Menggunakan secret key ketika algoritma dipublikasikan
    - One-way function
  - **Asymmetric (public-key)**
    - Menggunakan pan public/private key
    - Pola public key dimunculkan pertama oleh Diffie Hellman (1976)
    - Dasar public key : *trap-door function* adalah one-way function yang dapat dibalikkan dengan hanya adanya secret key

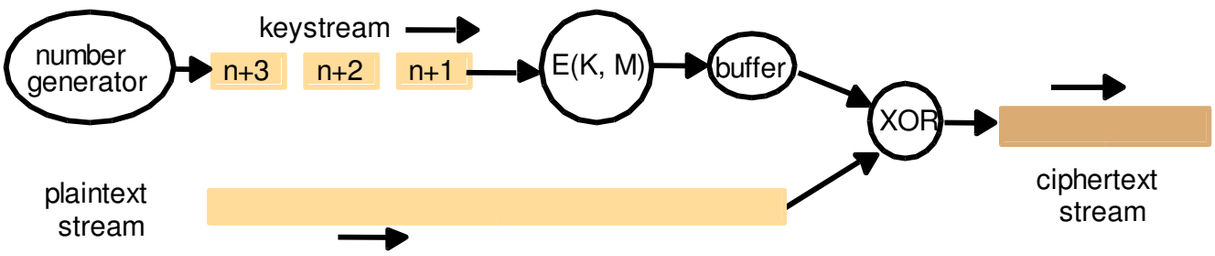
Chiper Block Chaining

- Untuk ukuran block data yang tetap, yang populer adalah 64 bit
- Pesan dibagi ke dalam block, dan block terakhir di *padding* ke ukuran standard yang digunakan, dan setiap block dienkrip secara independent
- Block pertama tersedia untuk transmisi setelah enkripsi selesai
- Chiper Block Chaining
  - Mengirim block terenkrip **XOR** block terenkrip sebelumnya
  - Mendekodekan ulang menggunakan operasi **XOR** dengan blok terenkrip sebelumnya
  - Jika ada kegagalan akan dihentikan prosesnya



Stream Chiper

- Menggunakan fasilitas incremental encryption
- Keysteram generator
  - Menghasilkan stream bit yang digunakan untuk memodifikasi isi data
  - XOR diterapkan sebelum di kirimkan
  - XOR diterapkan untuk mendekode ulang
  - Keysteram harus aman
  - Chiper block chaining mungkin digunakan untuk keamanan yang lebih baik



Contoh DSA,DH,RSA di Java

```
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;

public class getPublicPrivateKey {
    public static void main(String[] args) {
        try {
            // Generate a 1024-bit Digital Signature Algorithm (DSA) key pair
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
            keyGen.initialize(1024);
            KeyPair keypair = keyGen.genKeyPair();
            PrivateKey privateKey = keypair.getPrivate();
            PublicKey publicKey = keypair.getPublic();
            System.out.print("Public Key : " + publicKey);
            System.out.print("Private Key : " + privateKey);

...
```

```
            // Generate a 576-bit Diffie-Hellman (DH) key pair
            keyGen = KeyPairGenerator.getInstance("DH");
            keyGen.initialize(576);
            keypair = keyGen.genKeyPair();
            privateKey = keypair.getPrivate();
            publicKey = keypair.getPublic();
            System.out.print("Public Key : " + publicKey);
            System.out.print("Private Key : " + publicKey);

            // Generate a 1024-bit RSA key pair
            keyGen = KeyPairGenerator.getInstance("RSA");
            keyGen.initialize(1024);
            keypair = keyGen.genKeyPair();
            privateKey = keypair.getPrivate();
            publicKey = keypair.getPublic();
            System.out.print("Public Key : " + publicKey);
            System.out.print("Private Key : " + privateKey);
        } catch (java.security.NoSuchAlgorithmException e) {}
    }
}
```


DesEncrypter

```
import java.security.*;
import java.security.spec.*;
import javax.net.ssl.*;
import javax.crypto.*;
import java.io.*;
import javax.crypto.spec.*;

public class DesEncrypter {
    Cipher ecipher;
    Cipher dcipher;

    DesEncrypter(SecretKey key) {
        try {
            ecipher = Cipher.getInstance("DES");
            dcipher = Cipher.getInstance("DES");
            ecipher.init(Cipher.ENCRYPT_MODE, key);
            dcipher.init(Cipher.DECRYPT_MODE, key);

        } catch (javax.crypto.NoSuchPaddingException e) {
        } catch (java.security.NoSuchAlgorithmException e) {
        } catch (java.security.InvalidKeyException e) {
        }
    }
```

DesEncrypter

```
    public String encrypt(String str) {
        try {
            // Encode the string into bytes using utf-8
            byte[] utf8 = str.getBytes("UTF8");

            // Encrypt
            byte[] enc = ecipher.doFinal(utf8);

            // Encode bytes to base64 to get a string
            return new sun.misc.BASE64Encoder().encode(enc);
        } catch (javax.crypto.BadPaddingException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (UnsupportedEncodingException e) {
        } catch (java.io.IOException e) {
        }
        return null;
    }

....
```


DesEncrypter

```
    public String decrypt(String str) {
        try {
            // Decode base64 to get bytes
            byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str);

            // Decrypt
            byte[] utf8 = dcipher.doFinal(dec);

            // Decode using utf-8
            return new String(utf8, "UTF8");
        } catch (javax.crypto.BadPaddingException e) {
        } catch (IllegalBlockSizeException e) {
        } catch (UnsupportedEncodingException e) {
        } catch (java.io.IOException e) {
        }
        return null;
    }
}
```

tryDesEncrypter

```
import javax.crypto.*;
import java.io.*;

public class tryDesEncrypt {
    public static void main(String[] args) {
        try {
            // Generate a temporary key. In practice, you would save this key.
            // See also e464 Encrypting with DES Using a Pass Phrase.
            SecretKey key = KeyGenerator.getInstance("DES").generateKey();

            // Create encrypter/decrypter class
            DesEncrypter encrypter = new DesEncrypter(key);

            // Encrypt
            String encrypted = encrypter.encrypt("Budi Susanto");
            System.out.println(encrypted);
            // Decrypt
            String decrypted = encrypter.decrypt(encrypted);
            System.out.println(decrypted);
        } catch (Exception e) {
        }
    }
}
```