Enterprise Java Security

Atul Kahate

Project Manager
i-flex solutions limited, Pune, India

akahate@gmail.com

Benjamin Franklin once said …

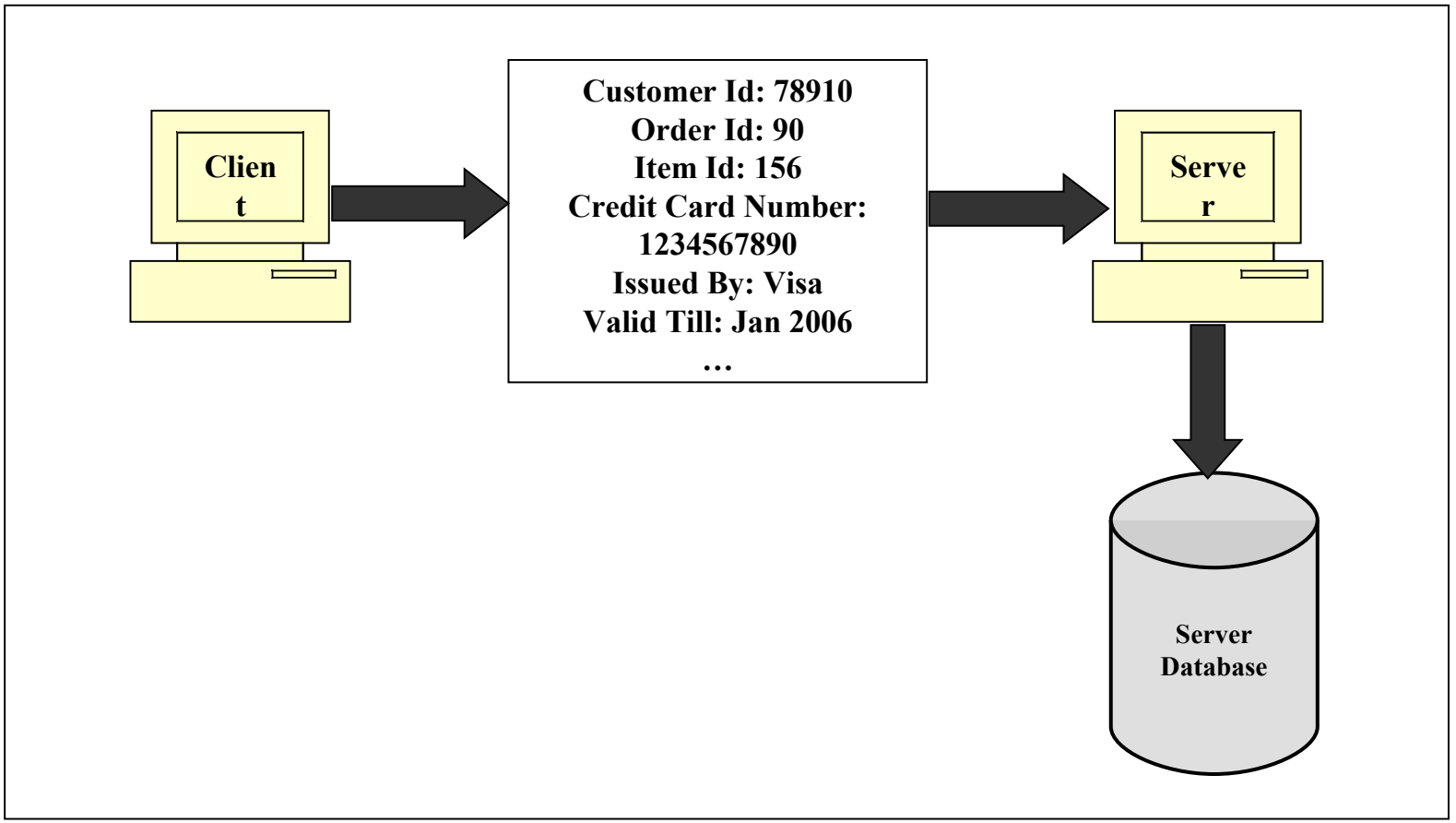
Three people can keep a secret …

… if two of them are dead!

Security is not easy to achieve

- Human tendency
- Problems of storage and communication
- Trust in all the parties

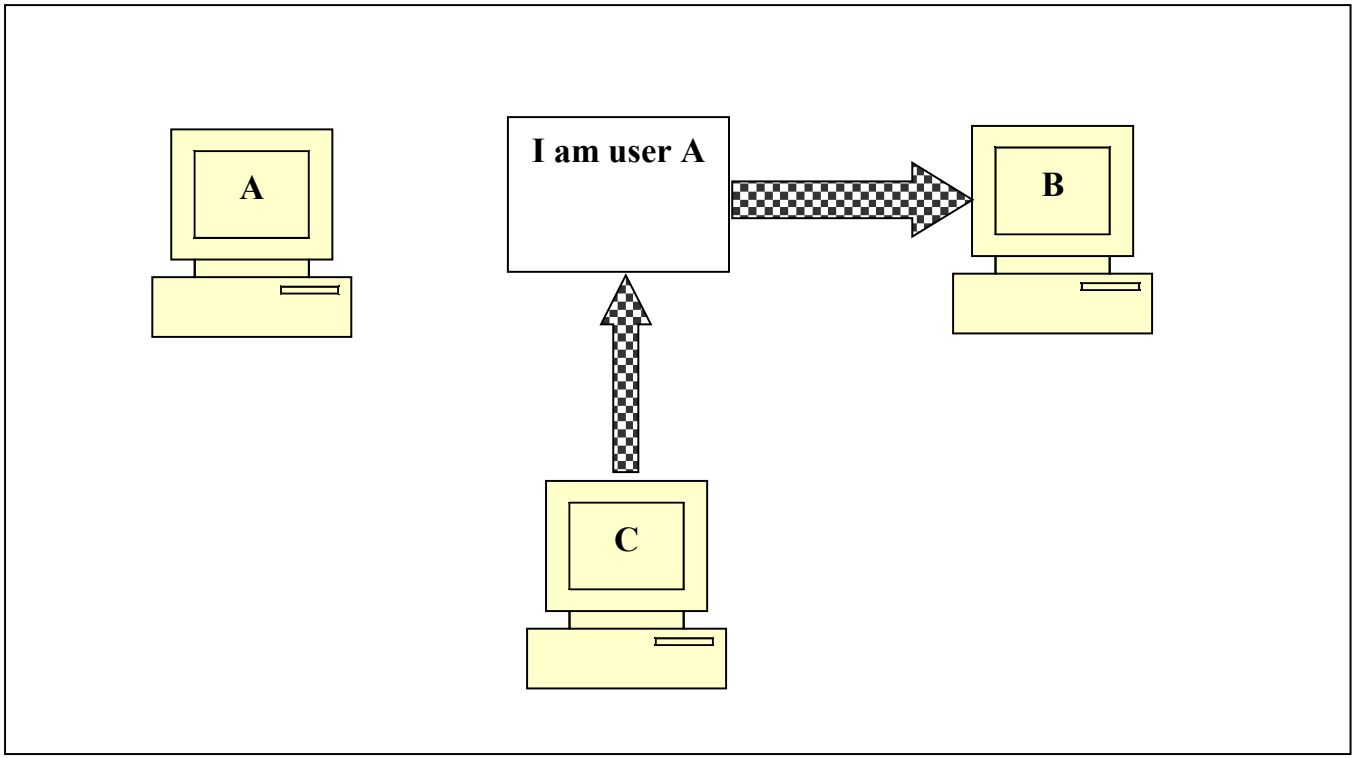
Transmission of Credit Card Details



Need for Security…

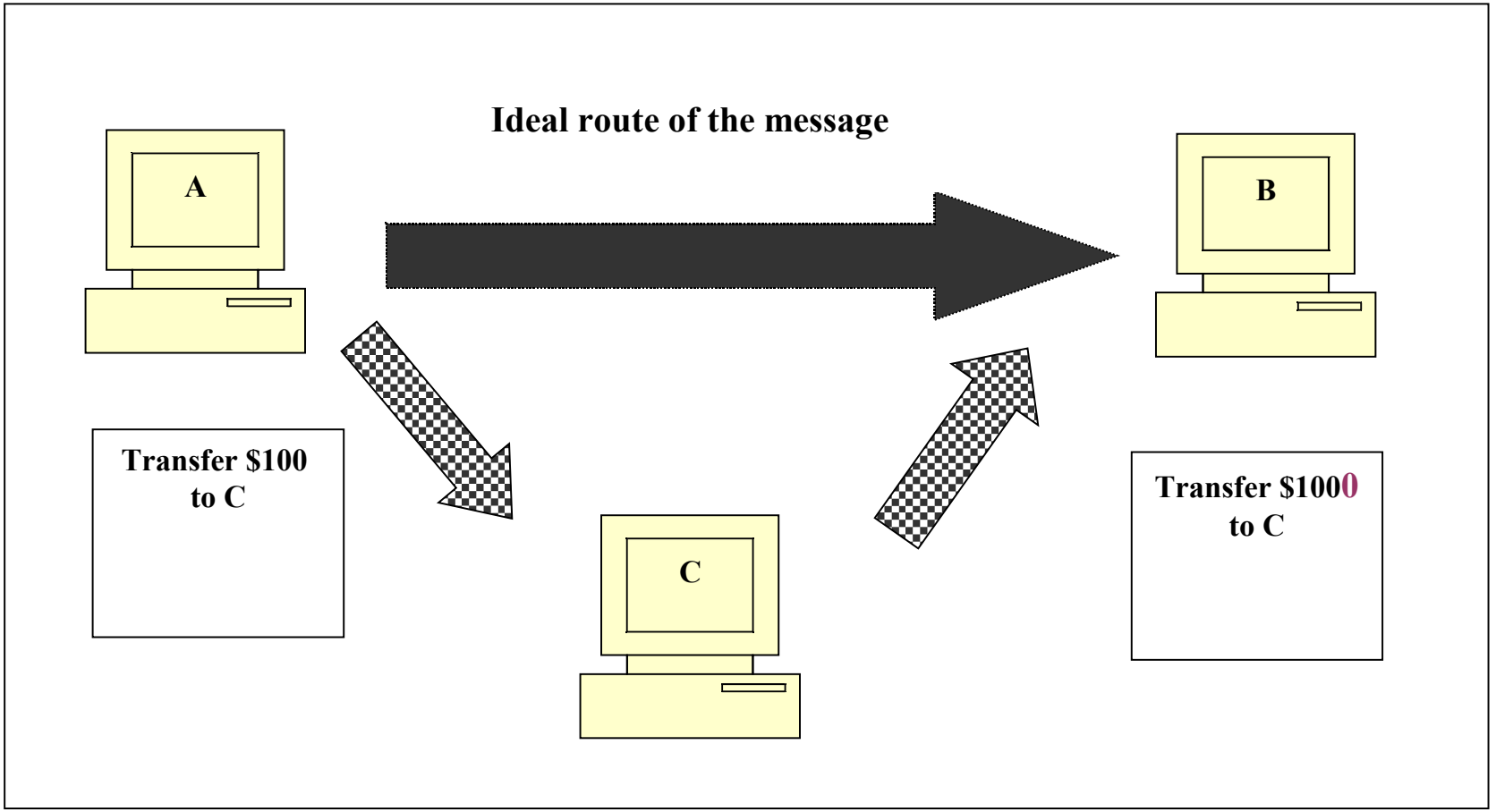
- Authentication
  - Do you trust the sender of a message?
- Integrity
  - Is the message changed during transit?
- Confidentiality
  - Is the message seen by someone else?
- Non-repudiation
  - Can the sender refute the message?

Authentication…



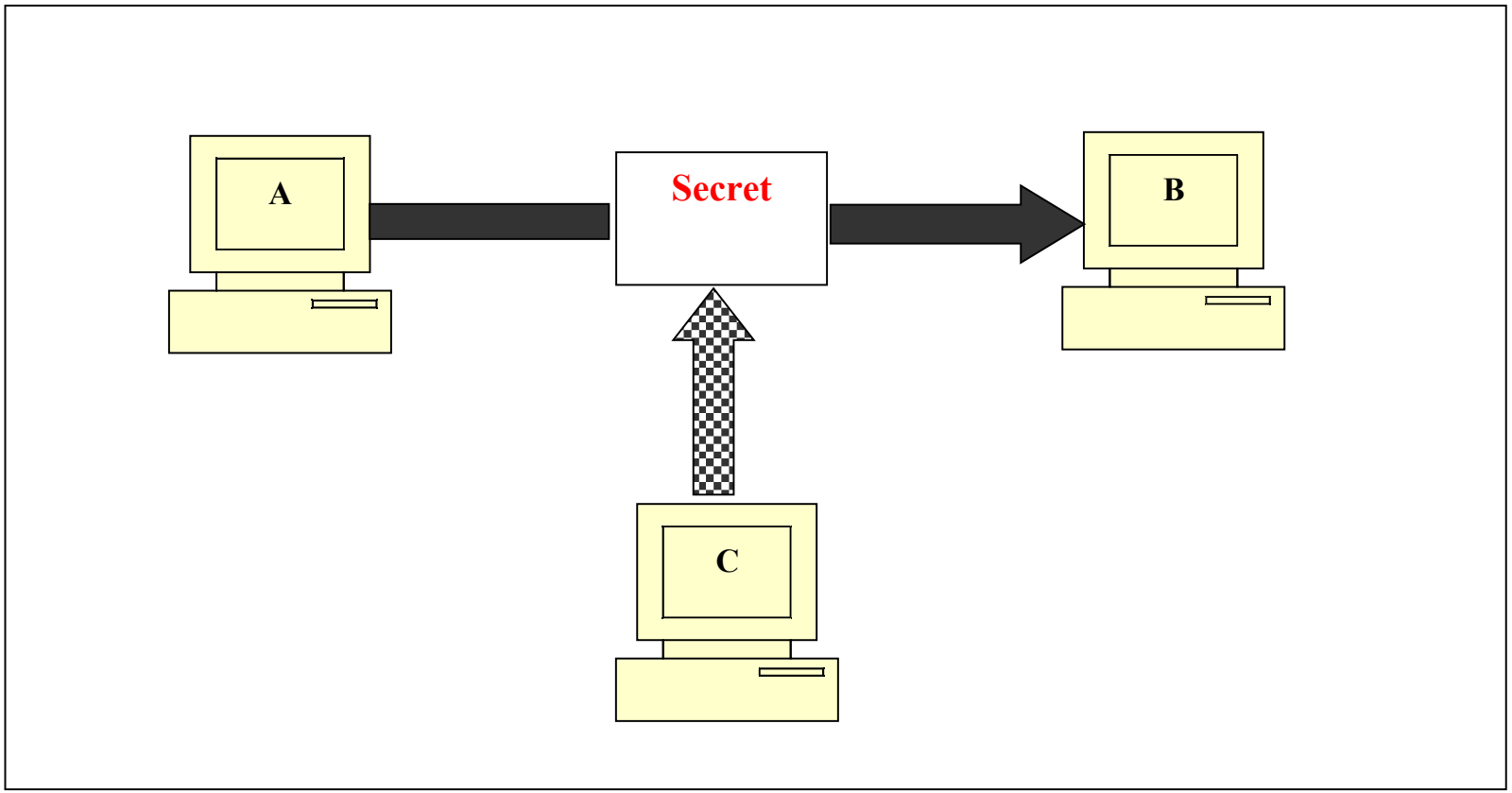
Who am I?

Integrity…



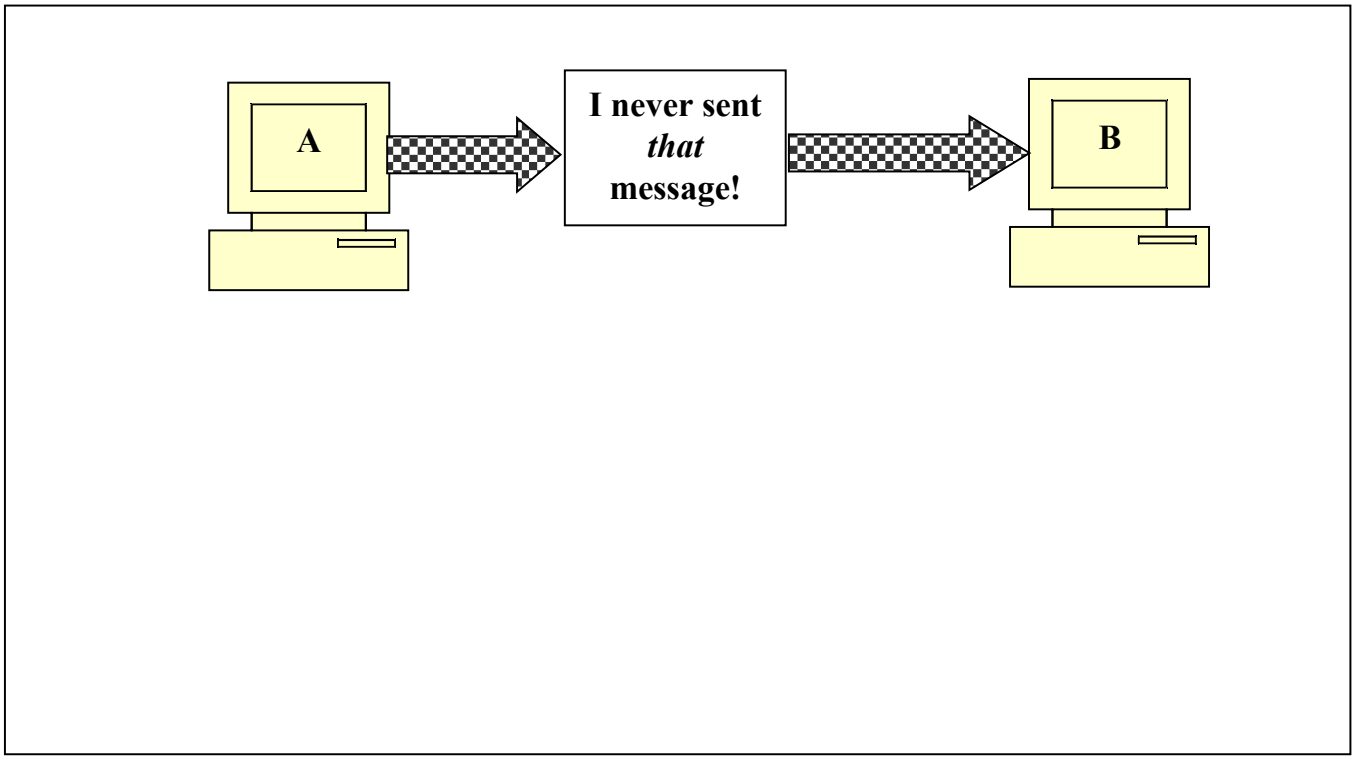
Has the Message Changed?

Confidentiality…



Has someone seen it?

Non-repudiation …



A sends a message and refutes it later

How to achieve Security?

- Use Cryptography
- Art of transforming messages
- Messages become unreadable

Cryptography Basics

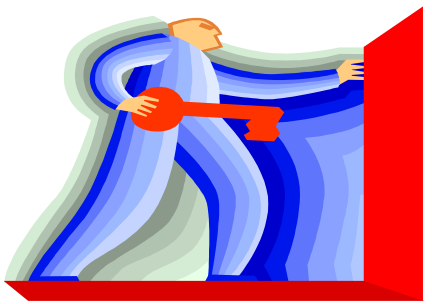
- Plain text
  - All understandable messages
  - Example: “My name is Atul”
- Cipher text
  - All non-understandable messages
  - Example: “G%er@17*0-1>-”

Cryptography Mechanisms

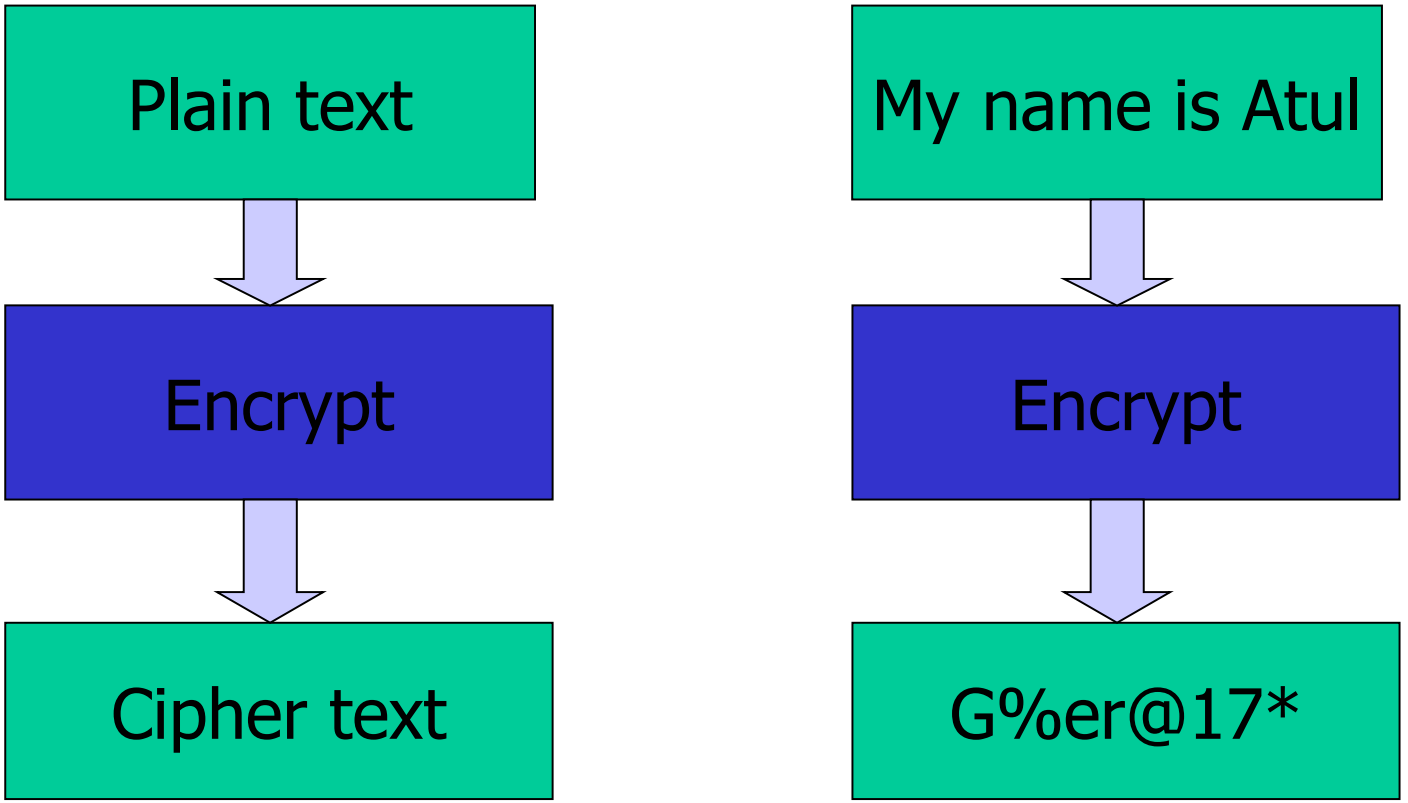
- Substitution Techniques
  - Replace one or more characters with other characters
  - Example: Replace each a with d, b with e, etc
- Transposition Techniques
  - Rearrange the text
  - Example: Replace 1st character with 4th, 2nd with 5th, etc
- Combinational Technique: Mostly used

Encryption and Decryption

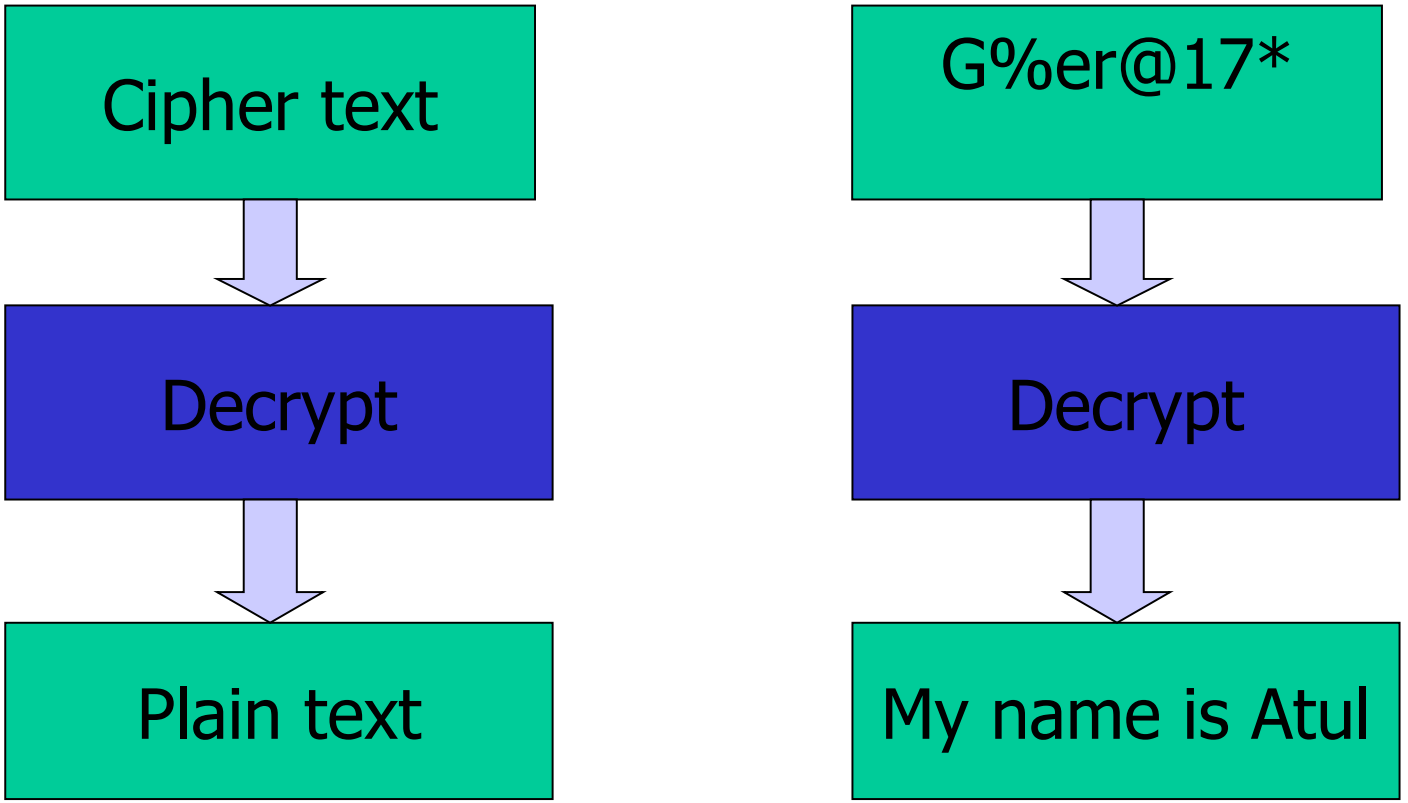
- Encryption
  - Conversion of *plain text* into *cipher text*
- Decryption
  - Conversion of *cipher text* into *plain text*
- Two Aspects
  - Algorithm
  - Key



Encryption


Plain text
Encrypt
Cipher text
My name is Atul
Encrypt
G%er@17*

Decryption


Cipher text
Decrypt
Plain text
G%er@17*
Decrypt
My name is Atul

Algorithm and Key

- Encryption/Decryption Algorithm
  - Specifies the operations to be performed
  - Examples: DES, IDEA, AES
  - Known to everybody

- Key
  - Must be kept secret
  - Usually at least 56 bits

Key Range

A 2-bit binary number has four possible states:
00
01
10
11

If we have one more bit to make it a 3-bit binary number, the number of possible states also doubles to eight, as follows:
000
001
010
011
100
101
110
111

In general, if an *n bit* binary number has *k* possible states, an *n+1 bit* binary number will have *2k* possible states.

Key Types

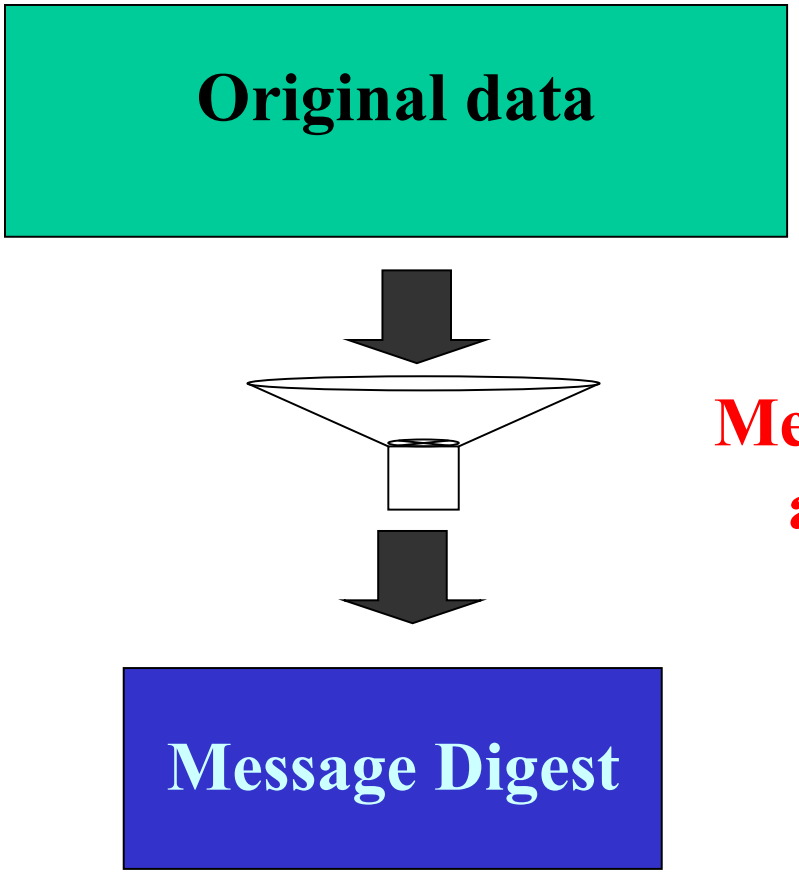
- Symmetric Key Encryption
  - The *same* key is used for encryption and decryption
- Asymmetric Key Encryption
  - One key used for encryption
  - Another, *different* key used for decryption

Message Digest: Concepts and Algorithms

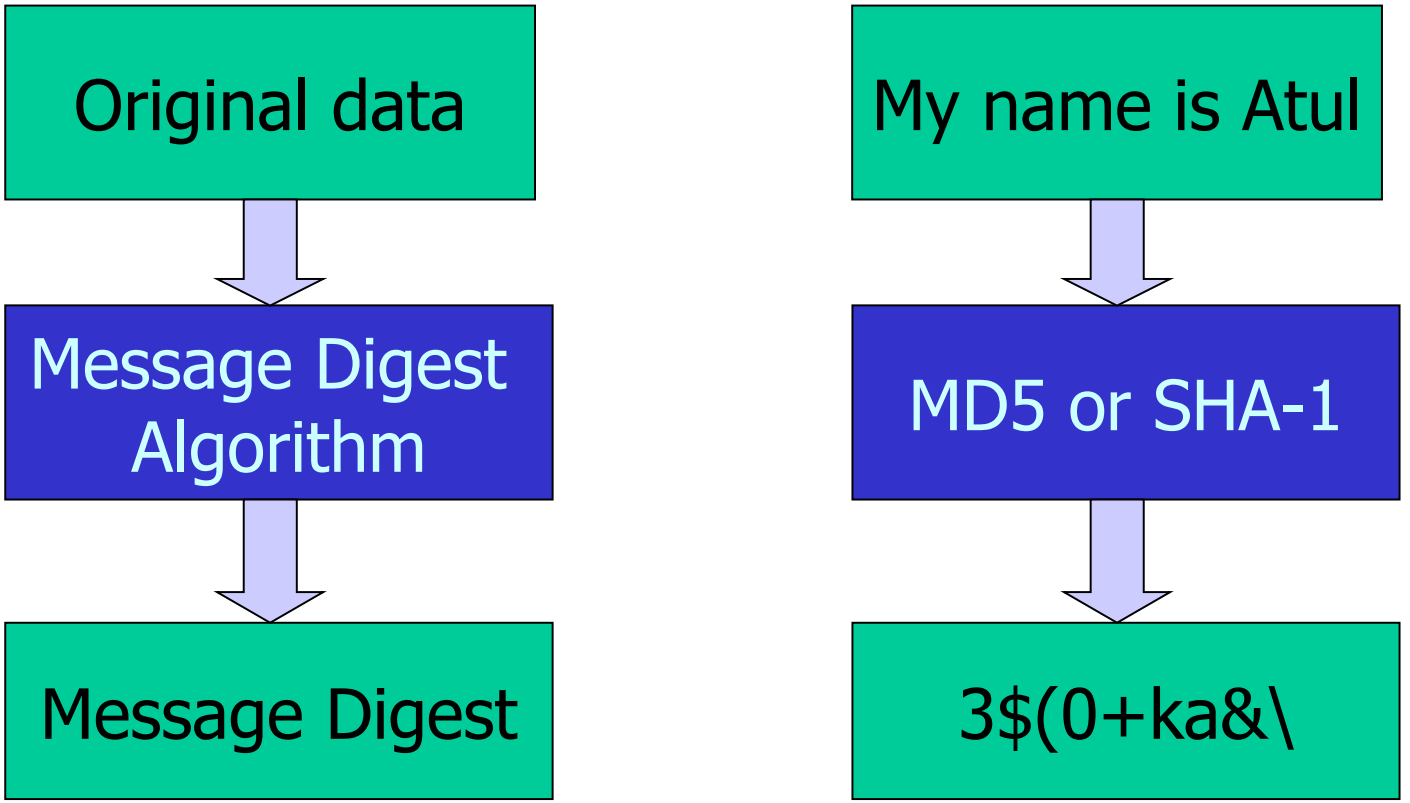
Message Digest

- Also called as *Hash*
- *Finger print* of the original data
- Similar to a Cyclic Redundancy Check (CRC)
- Used to detect if the data has changed
- Always one-to-one relationship with the original data

Message Digest: Concept



Message Digest: Conceptual Example



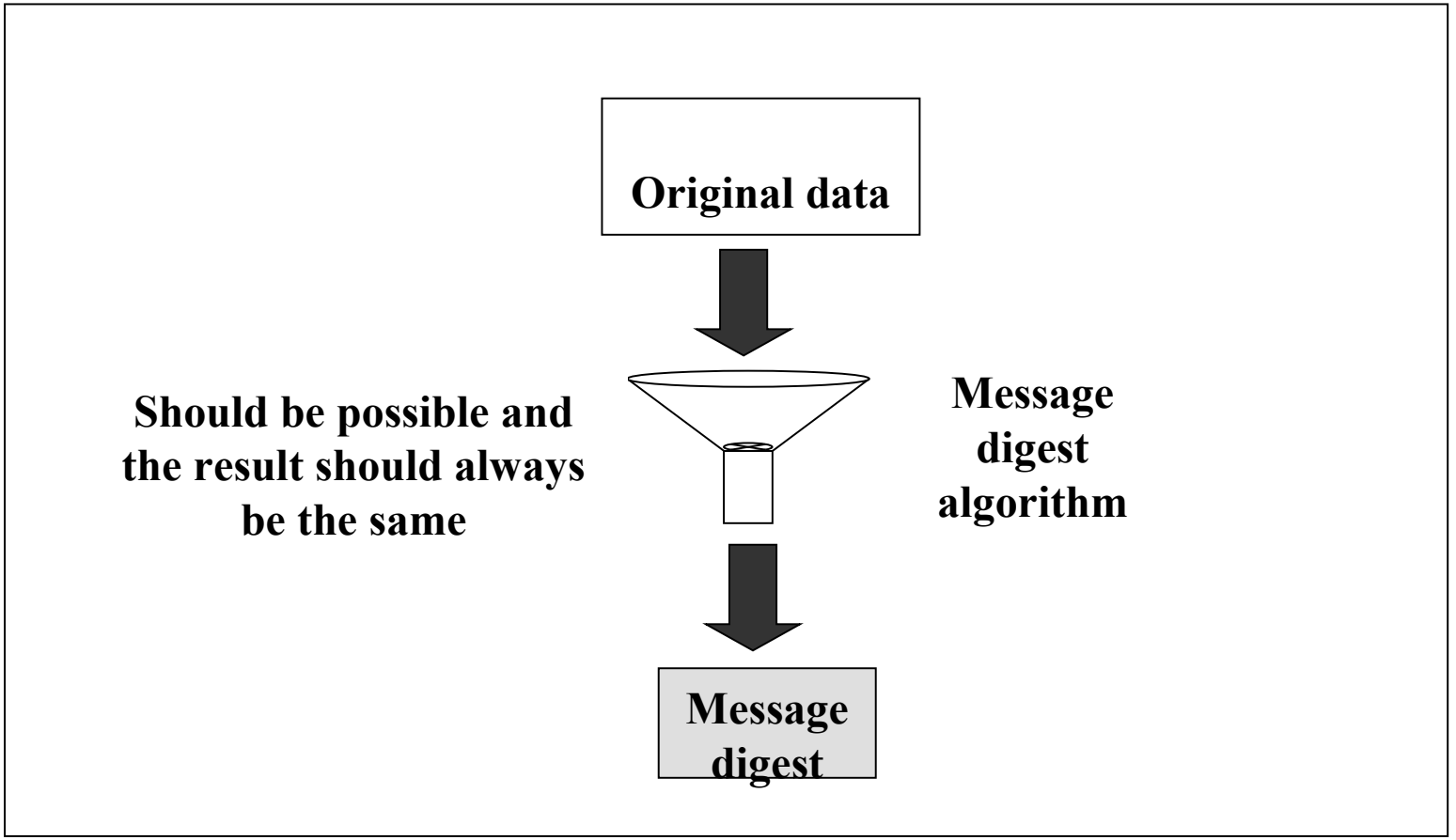
Message Digest: Actual Example

- **Original number is 7391743**

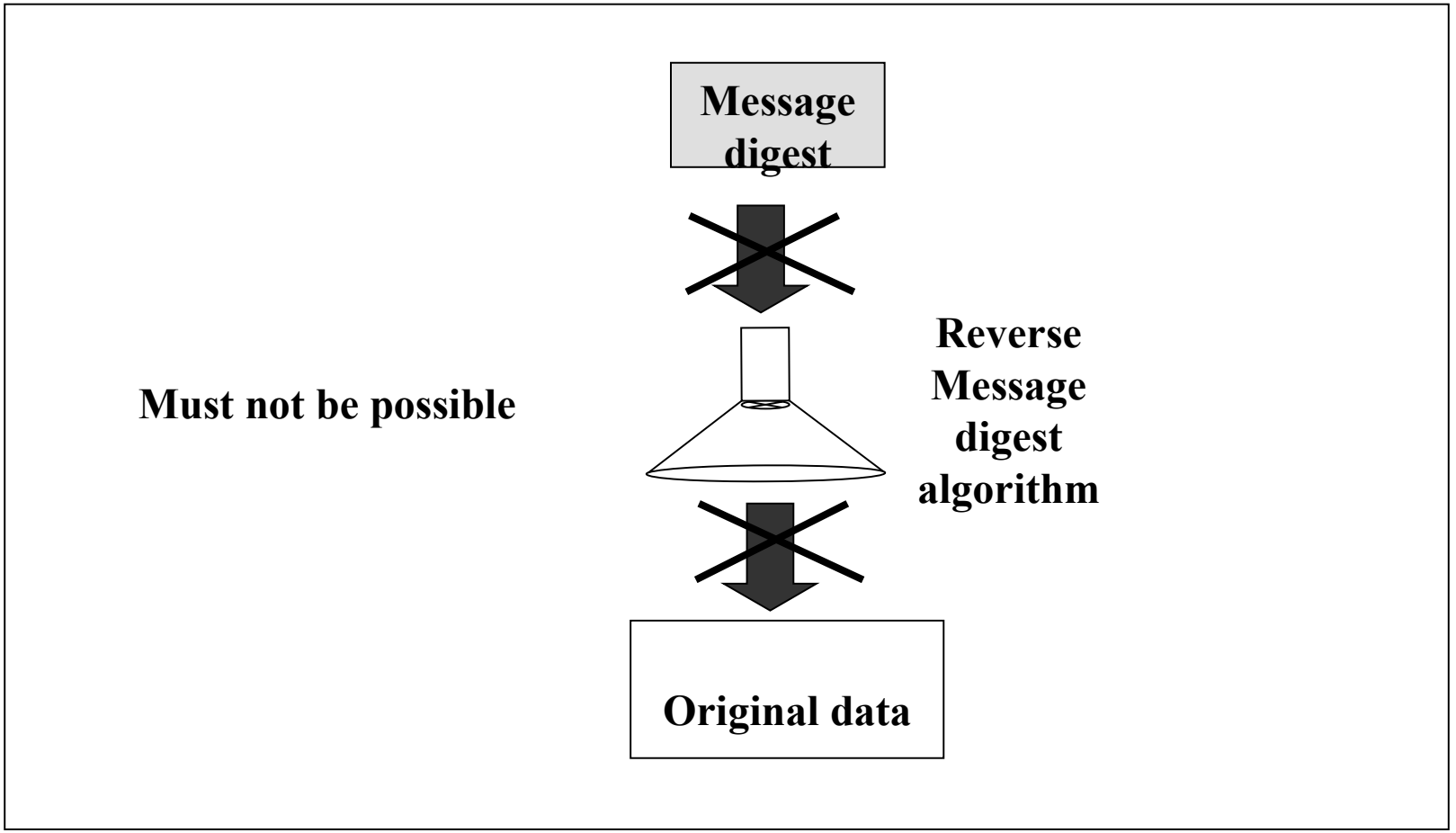
Operation	**Result**
Multiply 7 by 3	21
Discard first digit	1
Multiply 1 by 9	9
Multiply 9 by 1	9
Multiply 9 by 7	63
Discard first digit	3
Multiply 3 by 4	12
Discard first digit	2
Multiply 2 by 3	6

- **Message digest is 6**

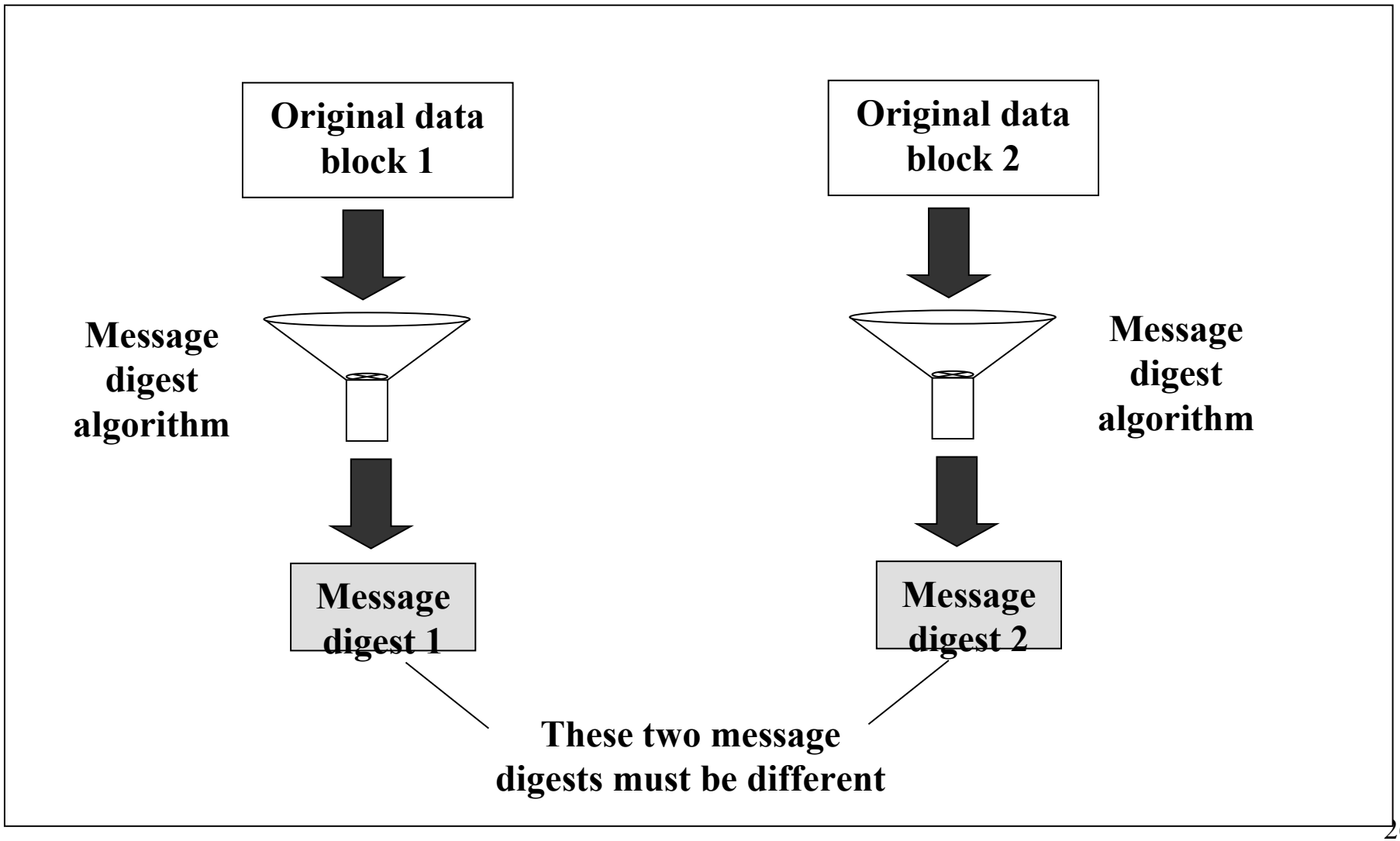
Message Digest Demands - 1



Message Digest Demands - 2



Message Digest Demands - 3



Message Digest Example

Message	**Please pay the newspaper bill today**
Message digest	**306706092A864886F70D010705A05A3058020100300906052B0E03021A050 0303206092A864886F70D010701A0250423506C6561736520706179207468652 06E6577737061706572062696C6C20746F646179041479630AC8041BA A1C40747F2FC29D881AEF92299B**
Message	**Please pay the newspaper bill tomorrow**
Message digest	**306A06092A864886F70D010705A05D305B020100300906052B0E03021A0 500303506092A864886F70D010701A0280426506C6561736520706179207 46865206E6577737061706572062696C6C20746F6D6F72726F7704146EE C2E0DB9570A5AF6CEB631CE057AE830A87C5B**

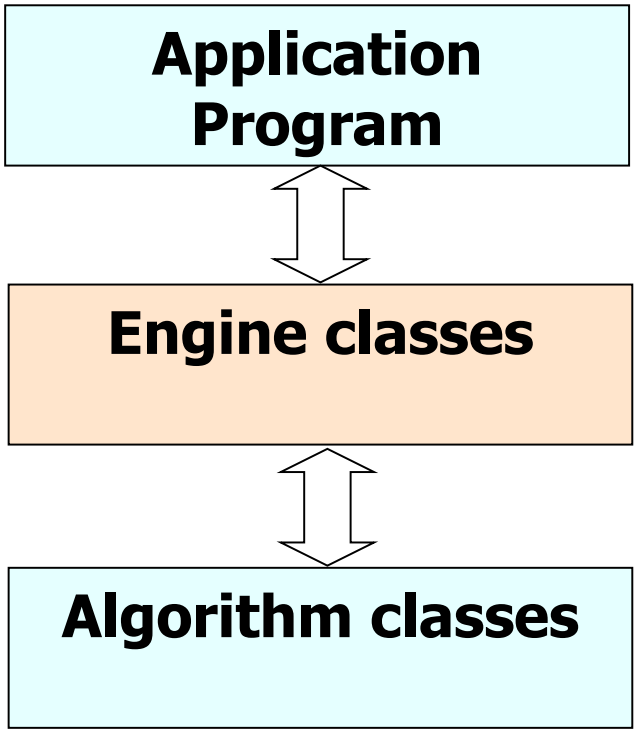
Java Cryptography

Java Cryptography

- Java Cryptography Architecture (JCA)
  - Takes care of certificates, message digests, and signatures
- Java Cryptography Extension (JCE)
  - Provides encryption services

JCA Concept

- Concept of interface and implementation is used
- Engine classes provide the interface (part of JDK), algorithm classes provide the implementation (Sun or other companies can supply these)



Java Cryptography Architecture (JCA)

- Consists of a number of classes in the java.security package and its sub-packages
- Provides generic APIs for message digests and digital signatures
- Important classes:
  - MessageDigest
  - Signature
  - KeyPairGenerator
  - KeyFactory
  - CertificateFactory
  - KeyStore
  - AlgorithmParameters
  - AlgorithmParameterGenerator
  - SecureRandom

Basics of JCA – Message Digest Example

- Call the *getInstance* method of the *MessageDigest* class

```
MessageDigest md = MessageDigest.getInstance ("MD5");
```

- Provide the data to be digested by calling the *update* method (it is assumed to be in an array called myData)

```
md.update (myData);
```

- Calculate the digest by using the *digest*

```
byte [ ] myDigest = md.digest ();
```

Java Cryptography Extension (JCE)

- Consists of a number of classes in the javax.crypto package and its sub-packages
- Provides generic APIs for encryption
- Important classes:
  - Cipher
  - KeyAgreement
  - KeyGenerator
  - Mac
  - SecretKey
  - SecretKeyFactory

Basics of JCE – Encryption Example

1. Generate the key

```
KeyGenerator kg = KeyGenerator.getInstance (“Blowfish”);
Key key = kg.generateKey ();
```

5. Create a cipher and initialize with the key just created

```
Cipher cipher = Cipher.getInstance (“Blowfish/ECB/PKCS5Padding”);
Cipher.init (Cipher.ENCRYPT_MODE, key);
```

- Calculate the digest by using the *encrypt*

```
byte [ ] myEncryptedData = cipher.doFinal (myData);
```

Example: Use of Blowfish for Encryption

```
// This program attempts to generate a Blowfish key and then performs an encryption operation

package securitydemo;

import javax.crypto.*;

public class SecurityDemo {
    public static final String str = "This is the text to be encrypted";

    public static void main(String[] args) throws Exception{

        // Generate a Blowfish key
        System.out.println ("Attempting to generate a Blowfish key ...");
        KeyGenerator kg = KeyGenerator.getInstance ("Blowfish");
        kg.init (128);
        SecretKey sk = kg.generateKey ();
        System.out.println ("Result: Success");

        // Attempt to perform encryption
```

Using DES-3 for Encryption and Decryption

```
// This program attempts to generate a Blowfish key and then performs an encryption operation

package securitydemo;

import javax.crypto.*;

public class EncryptDecrypt {
    public static final String str = "This is the text to be encrypted";

    public static void main(String[] args) throws Exception{

        // Generate a Blowfish key
        System.out.println ("Attempting to generate a Triple DES key ...");
        KeyGenerator kg = KeyGenerator.getInstance ("DESede");
        kg.init (168);
        SecretKey sk = kg.generateKey ();
        System.out.println ("Result: Success");

```

Message Digest Example

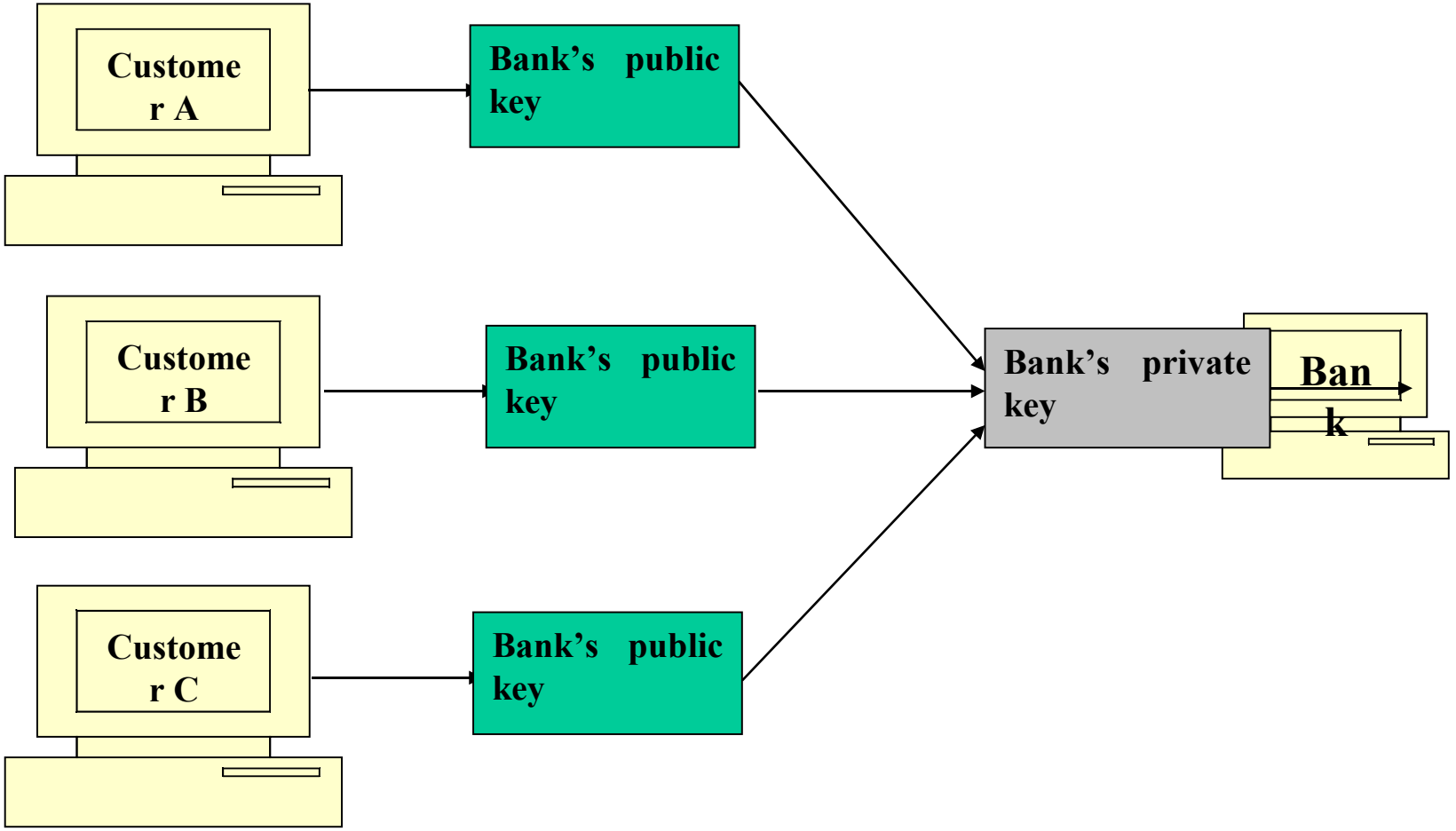
```
// This program attempts to calculate the message digest of a message

package securitydemo;

import java.security.MessageDigest;

public class MessageDigestExample {
    public static final String str = "This is the text to be digested. It is quite interesting!";

    public static void main(String[] args) throws Exception{

        // Create a message digest
        System.out.println ("Attempting to calculate message digest ...");
        MessageDigest md = MessageDigest.getInstance ("MD5");
```

Asymmetric Key Encryption

Asymmetric Key Encryption

- Also called as Public Key Encryption
- Each party has two keys
  - Public Key (say *K1*)
  - Private Key (say *K2*)
- Public Key is known to everybody
- Private Key must be kept secret
- Encrypt with *K1*, Decrypt with *K2*

Asymmetric Key Encryption: Concept



Digital Certificates

- Digital version of a Passport
- Binds a Person to a Public Key
- Issued by a Certification Authority (CA)
- Essential for Secure Web Transactions

Digital Certificate: Example

Digital Certificate

Subject Name: ***Atul Kahate***
Public Key: ***<Atul's key>***
Serial Number: ***1029101***
Other data: ***Email -***
akahate@indiatimes.com
Valid From: ***1 Jan 2001***
Valid To: ***31 Dec 2004***
Issuer Name: ***VeriSign***

Digital Certificate Contents

Version
Certificate Serial Number
Signature Algorithm Identifier
Issuer Name
Validity (Not Before / Not After)
Subject Name
Subject Public Key Information
Issuer Unique Identifier
Subject Unique Identifier
Extensions
Certification Authority's Digital Signature

Digital Signature

- Digital representation of a paper signature
- Used to authorize a transaction
- Useful in *non-repudiation*
- Signer cannot refuse having signed it
- Receiver cannot deny having received it (if used)

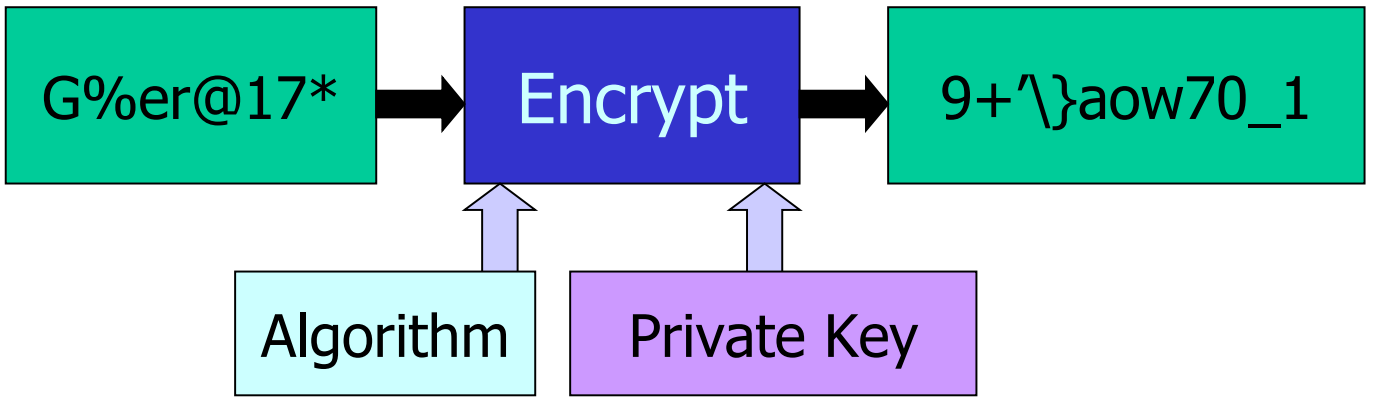
Digital Signature: Process

- Step 1: Calculate the message digest of the original data
- Step 2: Encrypt the message digest with the sender's private key
- Step 3: Output is the digital signature

Digital Signature: Example

Step 1: Calculate message digest

Step 2: Encrypt message digest to form the digital signature



XML Digital Signature

- Facility to sign specific portions of an XML document
- Different users can sign different portions
- Example: Purchase Transaction
  - Office Manager can sign quantity portion
  - Finance Manager can sign amount portion
  - General Manager can sign overall document

XML Digital Signature Process

1. Determine the resource to be signed (i.e. the URI)
   - Example: www.xyz.com/name.htm

2. Calculate the digest of the resource

```
<Reference URI="http://www.xyz.com/name.htm"> <DigestMethod
    Algorithm="sha1" />
    <DigestValue>jtMup4NbeVu#4q328nk=</DigestValue>
</Reference>
```

XML Digital Signature Process - Continued

1. Encrypt the digest to form a digital signature of the data

```
<SignatureValue>MC0E LE=</SignatureValue>
```

2. Add key data

```
<KeyInfo>
    <X509SubjectName> CN=Ed Simon,O=XMLSec
    Inc.,ST=OTTAWA,C=CA </X509SubjectName>
    <X509Certificate>MIID5jCCA0+gA…IVN </X509Certificate>
    </X509Data>
</KeyInfo>
```

Java and Digital Signatures, Digital Certificates

Digital Signature Example

```
// This program attempts to calculate the digital signature on a message

package securitydemo;

import java.security.*;

public class DigitalSignatureExample {
    public static final String str = "This is the text to be signed. It is quite interesting!";

    public static void main(String[] args) throws Exception{

        // Generate a RSA key pair
        System.out.println ("Attempting to generate a key pair ...");
        KeyPairGenerator kpg = KeyPairGenerator.getInstance ("RSA");
        kpg.initialize (1024);
        KeyPair kp = kpg.genKeyPair ();
        System.out.println ("Key pair generated successfully ...");

        // Sign data
        byte [] ba = str.getBytes("UTF8");
```

Java Keystore and Keytool

- Keystore
  - Collection of keys and certificates
  - Java.security.KeyStore is an in-memory representation of it
  - Contains two types of entries
    - Trusted certificates
    - Keys (private or symmetric)
- Keytool
  - Application shipped with the JDK
  - Manages keystores and can create certificates
  - Can be used at the command prompt by typing *keytool*

Using the keytool

- Creating a keystore
  - keytool –genkey –alias test OR
  - keytool -genkey -keystore MET
- Verifying the keystore
  - Keytool –v –list
- Exporting the keystore to a digital certificate
  - Keytool –export –alias test –file test.cer

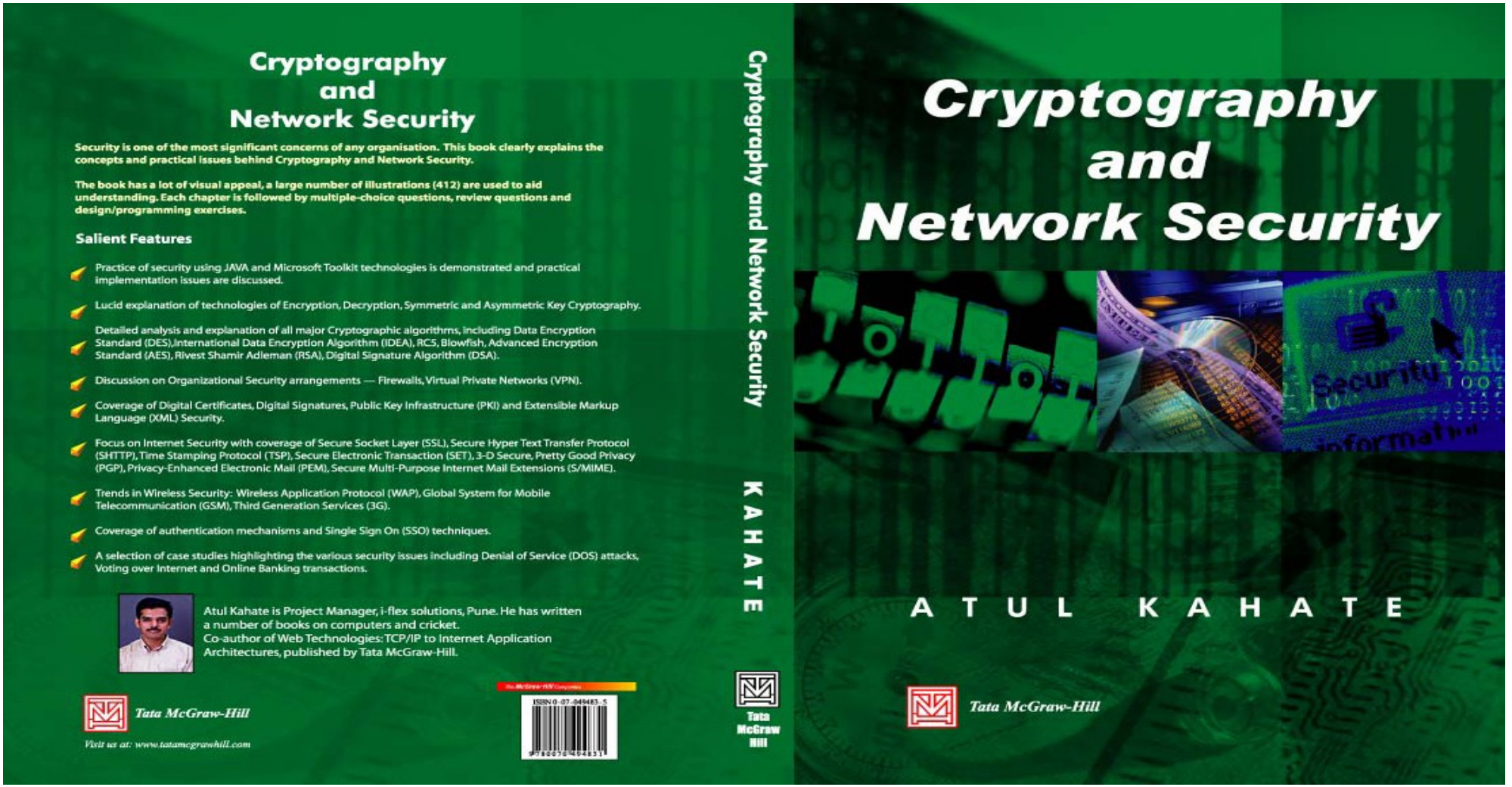
Using Digital Certificates in Java

```
package securitydemo;

import java.io.*;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;

public class PrintCertInfo {

    public static void main (String [] args) throws Exception {

        CertificateFactory cf = CertificateFactory.getInstance ("X.509");
        FileInputStream fis = new FileInputStream ("d:\\atul\\symbio~1\\webtec~1\\wt-2\\test.cer");
        Certificate cert = cf.generateCertificate (fis);
        fis.close ();

        System.out.println ("Here are the certificate contents ...");
        System.out.println (cert);
```

Using Keystore in Java

```
package securitydemo;

import java.io.*;
import java.security.cert.*;
import java.security.KeyStore;

public class PrintCertificateFromKeyStore {
    public static void main (String [] args) throws Exception {

        // default keystore is in the user's home diretcory
        String userHome = System.getProperty ("user.home");
        String keystorefilename = userHome + "\\" + ".keystore";

        // get the password and certificate alias to open the keystore
        char [] password = "password".toCharArray ();
        String alias = "test";

        // Open the keystore file and use a KeyStore instance to handle data in that file
        FileInputStream fis = new FileInputStream (keystorefilename);
        KeyStore keystore = KeyStore.getInstance ("JKS");
```

For lot more …



Thank You!!!

Questions, Comments Welcome

akahate@gmail.com