ENCRYPTION BOOT CAMP

Security is the Mission

@matthewmccull

Signed in as **matthew.mccullough**

delicious | Home | Bookmarks | People | Tags

Matthew McCullough's encryption Bookmarks

Bookmarks | Network | Tags | Subscriptions | Inbox

See all encryption bookmarks in Popular, Recent, or your Network.

matthew.mccullough | encryption ✖ | Type another tag | Bookmarks 29

Display options

09 MAR 10 **Crypto-Gram - Bruce Schneier newsletter** 7
EDIT | DELETE — newsletter, cryptography, encryption

Cryptography and Liberty 2
EDIT | DELETE — cryptography, law, legal, encryption

FlexiProvider for JCE 2
The FlexiProvider is a powerful toolkit for the Java Cryptography Architecture (JCA/JCE). It provides cryptographic modules that can be plugged into every application that is built on top of the JCA.
EDIT | DELETE — java, jce, plugin, opensource, cryptography, encryption

LightCrypto - Light API on top of BouncyCastle 26
EDIT | DELETE — cryptography, java, jce, encryption

What's New in the Crypto Law
EDIT | DELETE — cryptography, law, legal, encryption

01 MAR 10 **Using AES with Java Technology** 160
Sun Java tutorial on using AES as a standalone and JSSE algorithm
EDIT | DELETE — java, programming, howto, security, cryptography, tutorial, encryption

22 FEB 10 **NIST IT Security - Non-Encrypted Hall of Shame** 3
EDIT | DELETE — hacking, newsevent, encryption

Steganography 101 using Java 4

delicious.com/matthew.mccullough/encryption

ENCRYPTING?

STATISTICS SAY 76% ARE NOT

DATA BREACHED

AT 85% OF COMPANIES IN JUST THE LAST 12 MONTHS

ANCIENT HISTORY

Everything old is new again

Recipient
or
Storage

Sensitive
Data

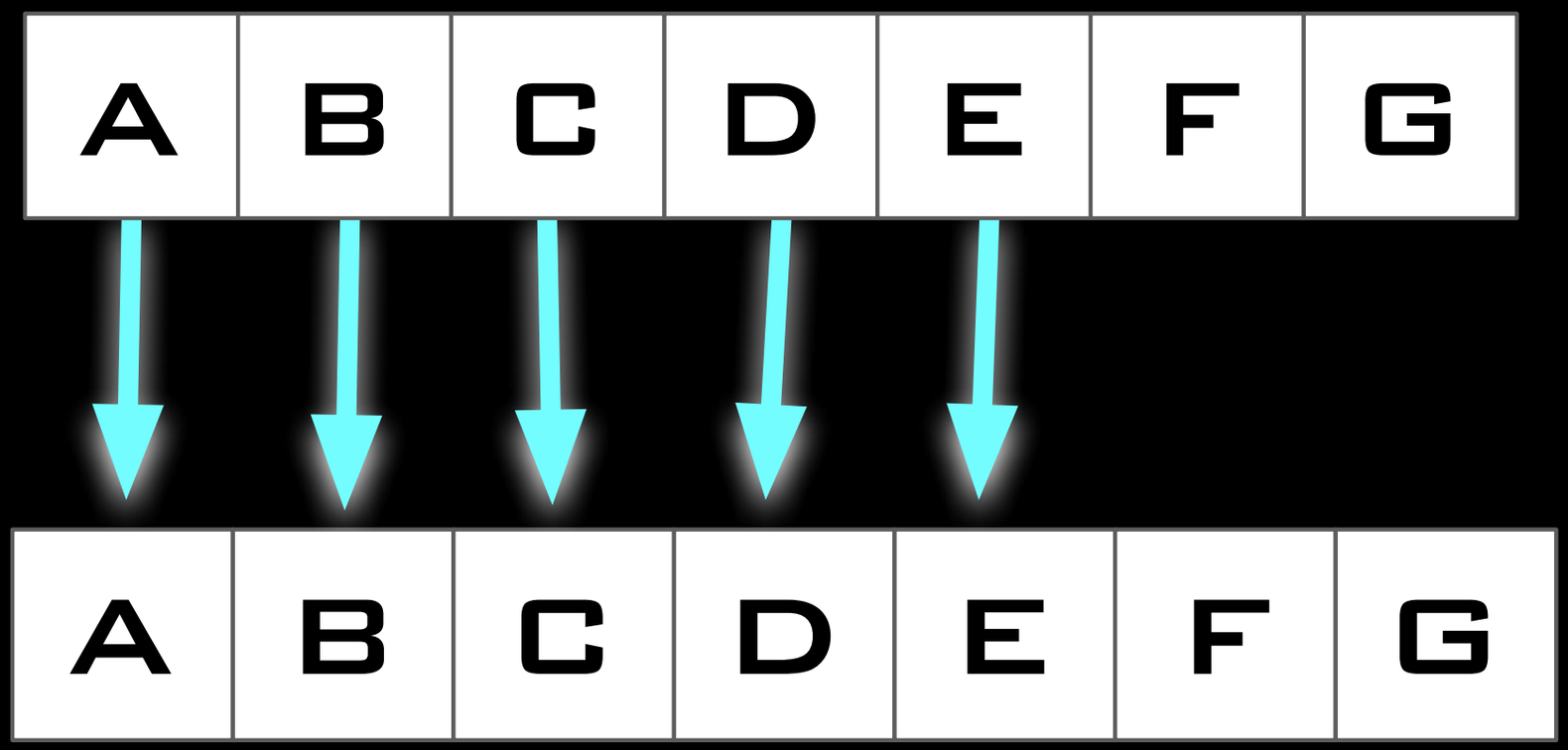
44 B.C.

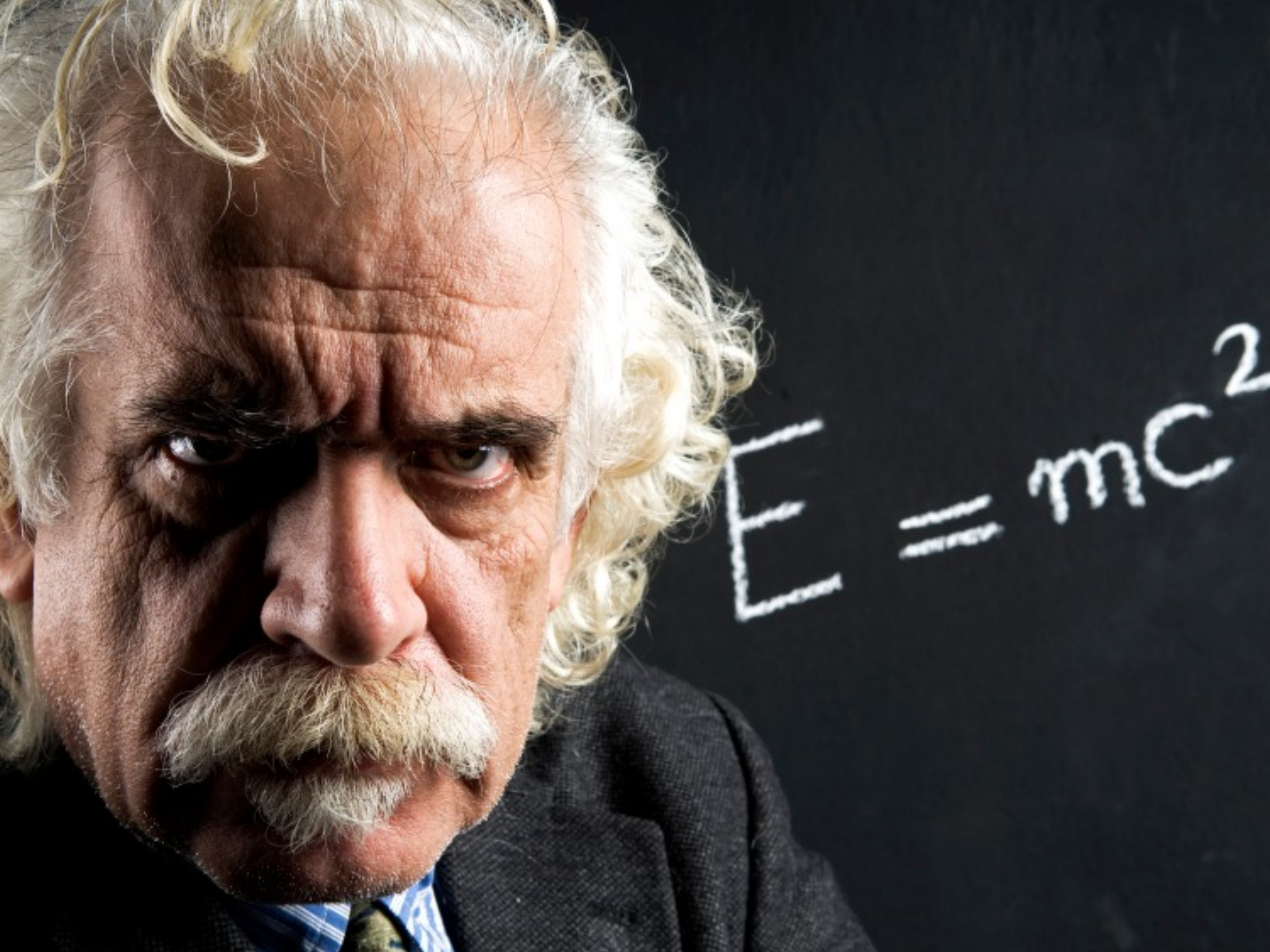
That’s 2,054 years ago...

Julius Caesar

Caesar Cipher

A.K.A.
ROT(2)
Shift Cipher




E = mc

BROKEN

Perfectly safe data is a myth

Compromised

- ★ Every algorithm is vulnerable
- ★ Crack by **real-time brute force**
- ★ Crack by **pre-computed tables**
- ★ Function of
  time + money + hardware

JCE PRIMER

The world of Java crypto

Java Cryptography Extension

- Known as JCE
- Included in all JREs Since Java 1.2
- Pluggable provider architecture
- JCE extends Java Cryptography Architecture (JCA)

JCE Providers

Default Sun JRE Providers

- SUN
- SunJCE
- SunJSSE
- SunRsaSign

Registering a Provider

Static

- <java-home>/lib/security/java.security
- security.provider.n=masterClassName

Terminal — zsh — zsh

```
Zeus /System/Library/Frameworks/JavaVM.framework/Versions/1.6/Home/lib/security
16 security %                                                              13:37:01
```

Registering a Provider

Dynamic

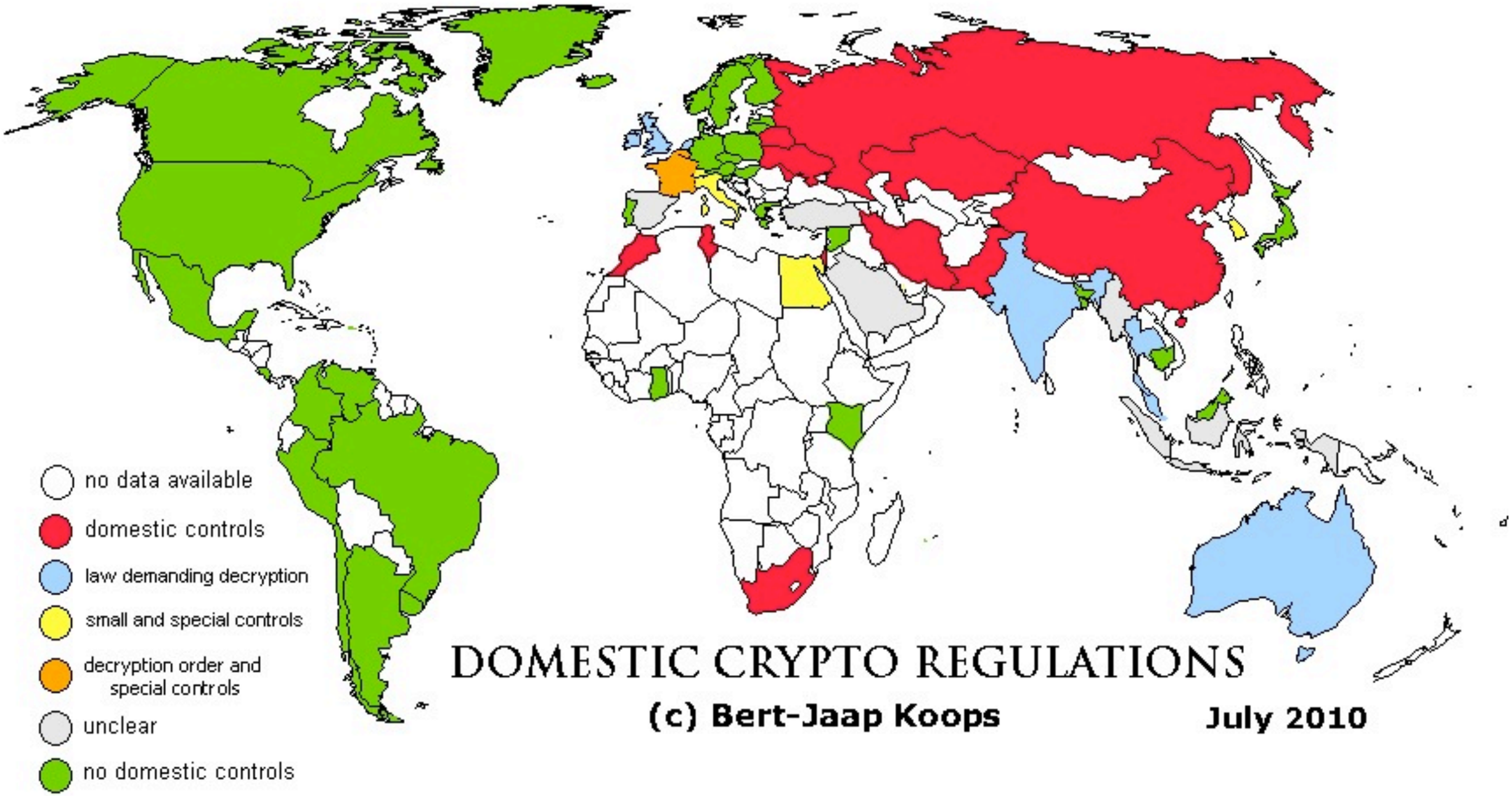
- java.security.Security class
  - addProvider()
  - insertProviderAt()
- Not persistent across VM instances

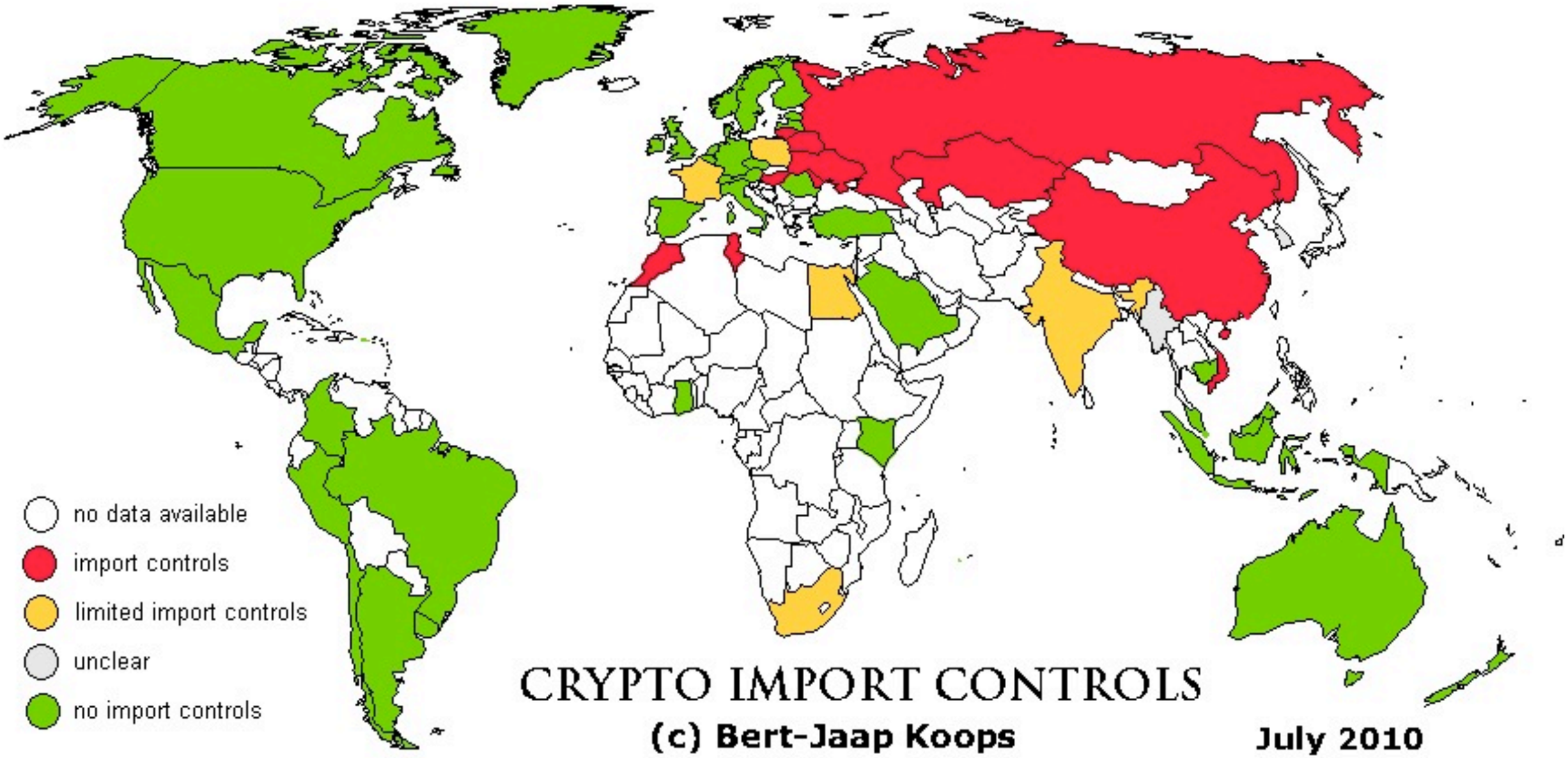
Encryption Law & the JCE

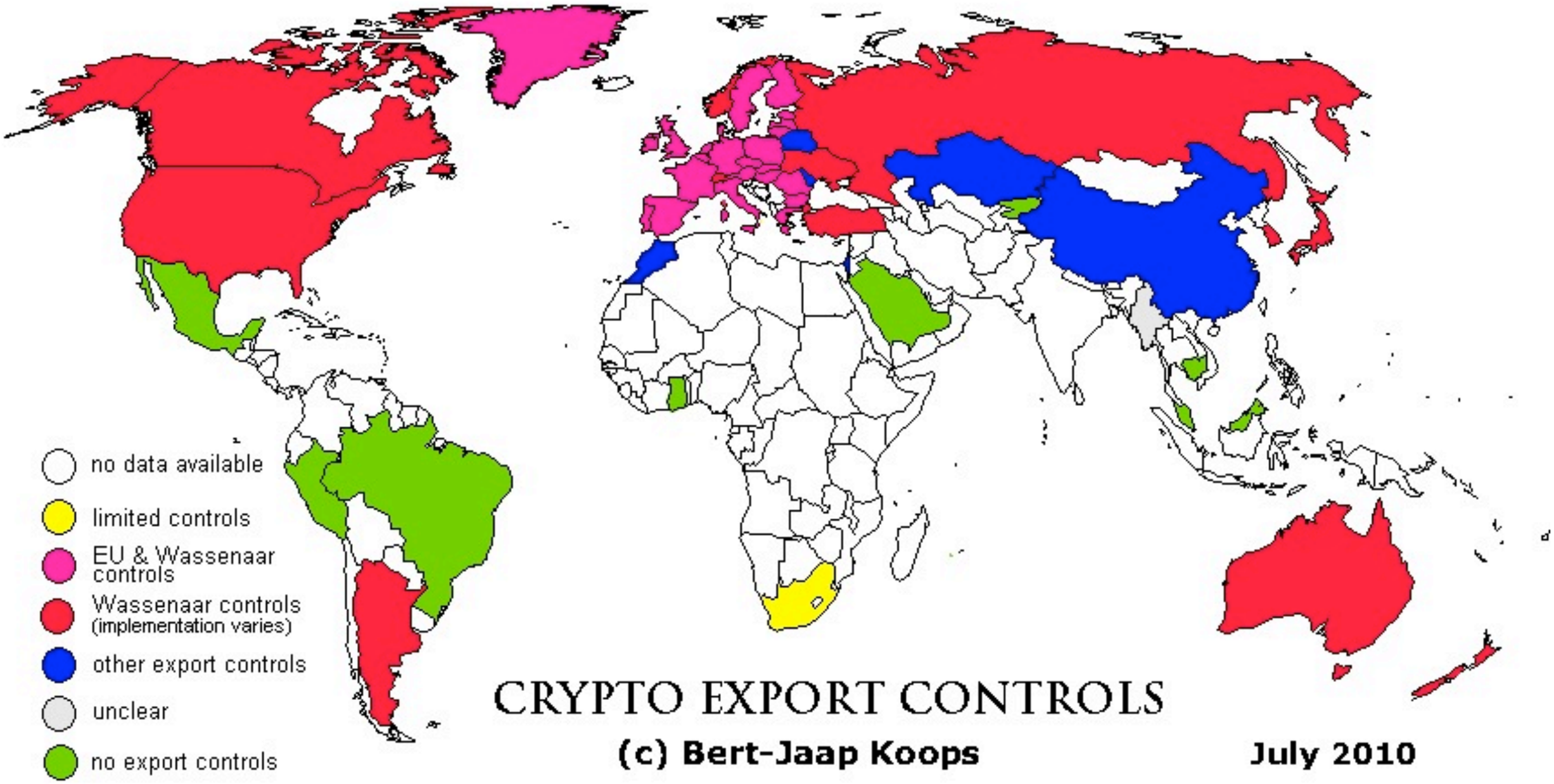
country borders stop bits

JCE Strength

- **Jurisdiction Policy Files**
- Two variants
- Algorithm strength differences


no data available
domestic controls
law demanding decryption
small and special controls
decryption order and special controls
unclear
no domestic controls
DOMESTIC CRYPTO REGULATIONS
(c) Bert-Jaap Koops
July 2010


no data available
import controls
limited import controls
unclear
no import controls
CRYPTO IMPORT CONTROLS
(c) Bert-Jaap Koops
July 2010


no data available
limited controls
EU & Wassenaar controls
Wassenaar controls (implementation varies)
other export controls
unclear
no export controls
CRYPTO EXPORT CONTROLS
(c) Bert-Jaap Koops
July 2010

Strong

Unlimited

JCE Strength

★ **Strong** strength included in all JREs

★ **Unlimited** strength is a separate download available based on US export rules

Java SE Downloads - Sun Developer Network (SDN)

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Google

JDK 6 Update 21 with NetBeans 6.9.1
This distribution of the JDK includes the NetBeans IDE, which is a powerful integrated development environment for developing applications on the Java platform. Learn more

Download

OTN Developer Day – Application Grids, Virtualisation, Clouds
Show Details Register Now
events.Oracle.com
Search

Additional Resources	
JDK DST Timezone Update Tool - 1.3.31 The tzupdater tool is provided to allow the updating of installed JDK/JRE images with more recent timezone data in order to accommodate the latest timezone changes. Learn more	Download ReadMe
Java SE 6 Documentation • Java SE 6 Documentation • Docs Installation Instructions • Oracle License	Download Zip
Java SE 6 JDK Source Code JDK 6 source code is available for those interested in exploring the details of the JDK. This includes schools, universities, companies, and individuals who want to examine the source code for personal interest or research & development. The licensing does not impose restrictions upon those who wish to work on independent open-source projects.	Download Sun Community Source License
Solaris SPARC Patches	Download
Solaris x86 Patches	Download
Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6	Download

What Java Do I Need?

You must have a copy of the Java Runtime Environment (JRE) on your system to **run** Java applications and applets. To **develop** Java applications and applets, you need the Java Development Kit, which includes the JRE.

Download Java(TM) Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6

Download Information and Files

Instructions: Click the file name to start the download.

Required Files

File Description and Name	Size
JCE Unlimited Strength Jurisdiction Policy Files 6 Release Candidate jce_policy-6.zip	8.89 KB

Notes:

- For download problems or questions, please see the Download FAQs.
- If you logged in first, you can complete this download any time in the next 30 days. Just visit your Download History.
- For Customer Service, contact Download Customer Service.

Getting Started?

» New to Java Center
» New to Solaris Center
» Sun Studio

Download Resources

» Download FAQs
» Download Customer Service

Related Resources

» Java.sun.com
» Solaris Developer Center
» JavaFX
» Web Developer Resource Center
» Developer Services
» JavaOne Online
» Sun Student Developer Program
» SunSolve
» Sun Microsystems Press
» Sun Partner Advantage Program for ISVs

Strong Policy

```
// File: default_local.policy
// Some countries have import limits on crypto strength.
// This policy file is worldwide importable.
grant {
    permission javax.crypto.CryptoPermission "DES", 64;
    permission javax.crypto.CryptoPermission "DESede", *;
    permission javax.crypto.CryptoPermission "RC2", 128,
                    "javax.crypto.spec.RC2ParameterSpec", 128;
    permission javax.crypto.CryptoPermission "RC4", 128;
    permission javax.crypto.CryptoPermission "RC5", 128,
          "javax.crypto.spec.RC5ParameterSpec", *, 12, *;
    permission javax.crypto.CryptoPermission "RSA", 2048;
    permission javax.crypto.CryptoPermission *, 128;
};
```

“Strong” Key Limits

Algorithm	Max Key Size
DES	64
DESede 3DES	168
RC2	128
RC4	128
RC5	128
RSA	2048
Others	128

Command Prompt

```
C:\Users\admin\Desktop\repos\encryption-jvm-bootcamp\symmetric-encrypt>
```

"Unlimited" Key Limits

Algorithm	Max Key Size
DES	∞
DESede 3DES	∞
RC2	∞
RC4	∞
RC5	∞
RSA	∞
Others	∞

Digests & Hashes

One way functions

What is a Hash?

- **Small** set of bytes representing a **large** message
- **Small** change in message = **large** change in digest
- **Digests** also known as **hashes**
  - Same algorithms, different purposes

What is a Digest?

- **Integrity check** (MIC) for chunk of data

  or

- **Password storage** mechanism

MessageDigest

- ★ `java.security.MessageDigest`
- ★ Many algorithms available
- ★ MD2
- ★ MD5 (128 bit)
- ★ SHA-1 (160 bit)
- ★ SHA-256
- ★ SHA-384
- ★ SHA-512

MessageDigest

U. S. Department of Homeland Security said **MD5** is

"*considered cryptographically broken and unsuitable for further use*"

Java ™ Cryptography Architecture Standard Algorithm Name Documentation

for Java™ Platform Standard Edition 6

Table of Contents

download.oracle.com/javase/6/docs/technotes/guides/security/StandardNames.html

```
    + shaAndBase64Encode(message1));
  System.out.println("Message2 SHA1 digest: "
    + shaAndBase64Encode(message2));
}
```

```
/**
 * Helper function to both SHA-1 hash and
 * base64 encode the resulting bytes to a String
 */
public static String shaAndBase64Encode(String message)
    throws NoSuchAlgorithmException {
  MessageDigest sha = MessageDigest.getInstance("SHA-1");

  //Salt could be applied here
  //Integer salt = <some random number generator>
  //sha.update(salt.getBytes());

  byte[] digest = sha.digest(message.getBytes());
  return new sun.misc.BASE64Encoder().encode(digest);
  }
}
```

```java
/**
 * Demonstrate that very similar messages
 * have radically different hashes.
 */
public class MessageDigestSHA
{
  public static void main( String[] args )
    throws NoSuchAlgorithmException
  {
    //Set up the message to be encoded
    String message1 = "Four score and seven years ago";
    String message2 = "Four score and seven tears ago";

    System.out.println("Message1 SHA1 digest: "
      + shaAndBase64Encode(message1));
    System.out.println("Message2 SHA1 digest: "
      + shaAndBase64Encode(message2));
  }

  /**
   * Helper function to both SHA-1 hash and
   * base64 encode the resulting bytes to a String
   */
  public static String shaAndBase64Encode(String message)
      throws NoSuchAlgorithmException {
```

Input

```
String message1 = "Four score and seven years ago";
String message2 = "Four score and seven tears ago";
```

Result

```
Message1 SHA1 digest: DmCJIg4Bq/xpGIxVXxo3IB0vo38=
Message2 SHA1 digest: oaLHt8tr31ttngCDjyYuWowF5Mc=
```

SYMMETRIC

My key is your key

Why Symmetric?

- ★ Fast
- ★ Well suited for bulk data

Using Symmetric

- ★ **Secure network** for passing keys
  or
- ★ **Never decrypted** at remote end

Symmetric Problems

- Keys vulnerable to capture
- Eavesdropping on future communications after key compromise
- Key distribution challenges
  - Triangular number key growth

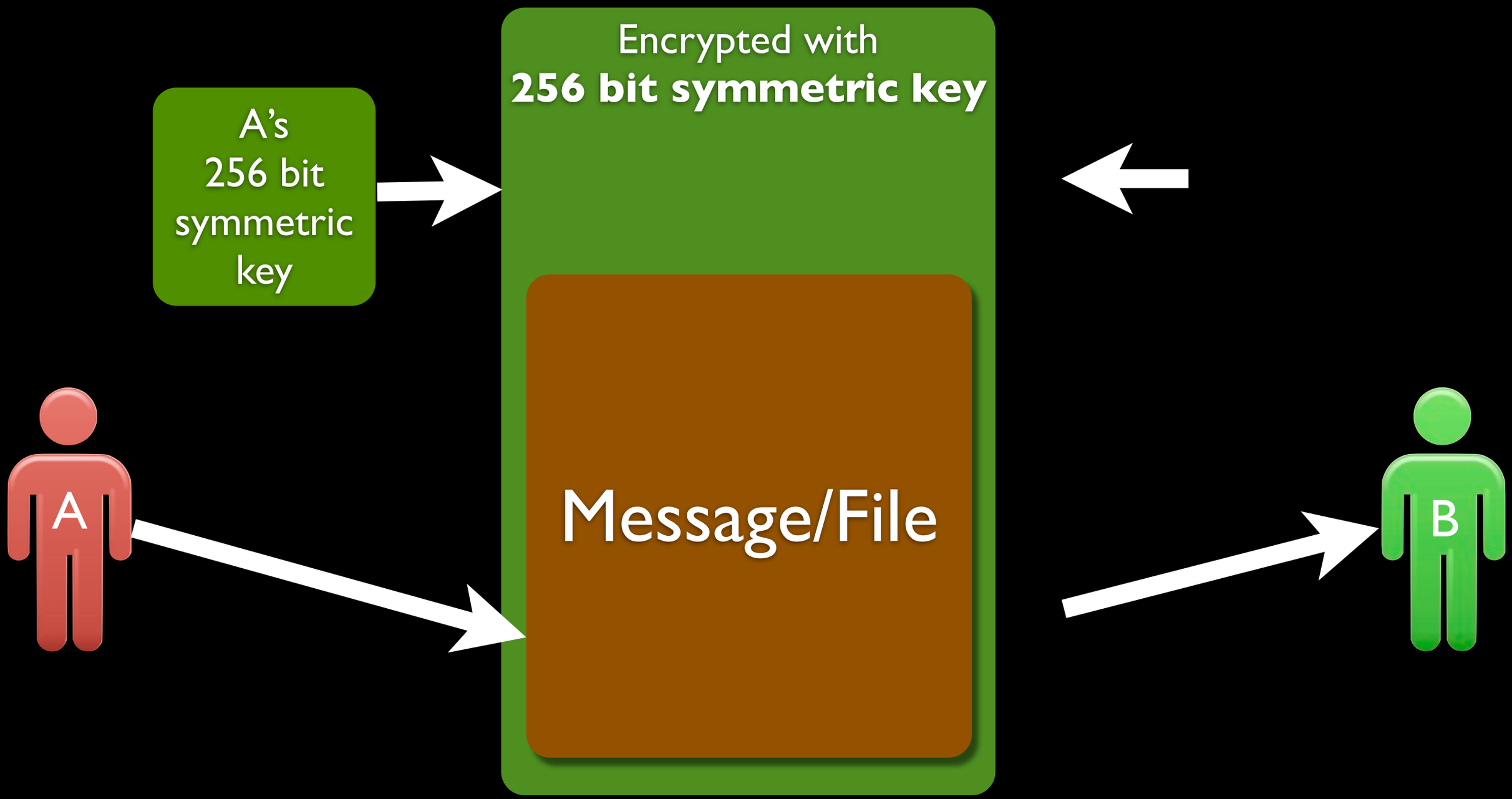
Symmetric Problems

- Triangular number key growth

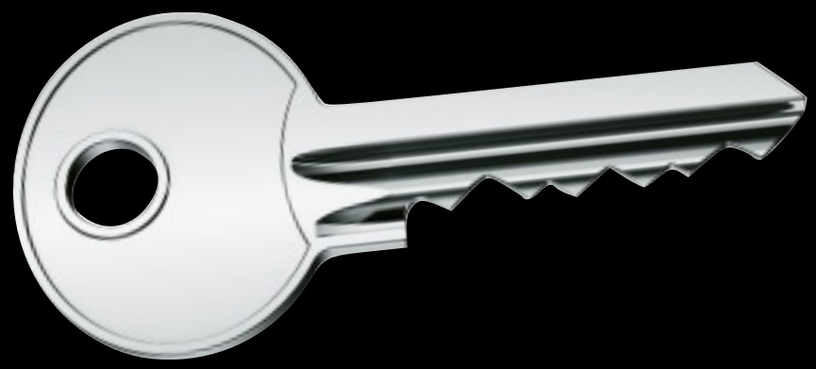
$$n$$

$$T_n \equiv \sum_{k=1} k$$

Symmetric


A's
256 bit
symmetric
key
Encrypted with
256 bit symmetric key
Message/File
A
B

How do we securely get the key from Alice to Bob?



```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

import sun.misc.BASE64Encoder;

/**
 * Use the SecureRandom java security class to generate
 * a more expensive, but cryptographically secure random number.
 */
public class SymmetricEncrypt
{
  public static void main( String[] args )
    throws NoSuchAlgorithmException, NoSuchProviderException,
    NoSuchPaddingException, InvalidKeyException,
    IllegalBlockSizeException, BadPaddingException
  {
    final String message1 = "Four score and seven years ago";
```

```java
import sun.misc.BASE64Encoder;

/**
 * Use the SecureRandom java security class to generate
 * a more expensive, but cryptographically secure random number.
 */
public class SymmetricEncrypt
{
  public static void main( String[] args )
    throws NoSuchAlgorithmException, NoSuchProviderException,
    NoSuchPaddingException, InvalidKeyException,
    IllegalBlockSizeException, BadPaddingException
  {
    final String message1 = "Four score and seven years ago";

    //Build a new encryption key
    final KeyGenerator keyGen = KeyGenerator.getInstance("DESede");
    keyGen.init(168);
    final SecretKey desKey = keyGen.generateKey();

    //Set up the cipher
    final Cipher desCipher = Cipher.getInstance("DESede/ECB/PKCS5Padding");

    ////////////////////////////////////
    //Put the cipher in encryption mode
    desCipher.init(Cipher.ENCRYPT_MODE, desKey);

    //Encrypt and output the base64 data
    byte[] clearText = message1.getBytes();
    byte[] encryptedBytes = desCipher.doFinal(clearText);
    BASE64Encoder b64e = new sun.misc.BASE64Encoder();
```

```
        final String message1 = "Four score and seven years ago";

        //Build a new encryption key
        final KeyGenerator keyGen = KeyGenerator.getInstance("DESede");
        keyGen.init(168);
        final SecretKey desKey = keyGen.generateKey();

        //Set up the cipher
        final Cipher desCipher = Cipher.getInstance("DESede/ECB/PKCS5Padding");

        ////////////////////////////////////
        //Put the cipher in encryption mode
        desCipher.init(Cipher.ENCRYPT_MODE, desKey);

        //Encrypt and output the base64 data
        byte[] clearText = message1.getBytes();
        byte[] encryptedBytes = desCipher.doFinal(clearText);
        BASE64Encoder b64e = new sun.misc.BASE64Encoder();
        String base64Encrypted = b64e.encode(encryptedBytes);
        System.out.println("Encrypted text: " + base64Encrypted);

        ////////////////////////////////////
        //Put the cipher in decryption mode
        desCipher.init(Cipher.DECRYPT_MODE, desKey);

        //Decrypt and output the original string
        byte[] decryptedBytes = desCipher.doFinal(encryptedBytes);
        String decryptedText = new String(decryptedBytes);
        System.out.println("Decrypted text: " + decryptedText);
    }
}
```

```
    //Set up the cipher
    final Cipher desCipher = Cipher.getInstance("DESede/ECB/PKCS5Padding");

    ////////////////////////////////////
    //Put the cipher in encryption mode
    desCipher.init(Cipher.ENCRYPT_MODE, desKey);

    //Encrypt and output the base64 data
    byte[] clearText = message1.getBytes();
    byte[] encryptedBytes = desCipher.doFinal(clearText);
    BASE64Encoder b64e = new sun.misc.BASE64Encoder();
    String base64Encrypted = b64e.encode(encryptedBytes);
    System.out.println("Encrypted text: " + base64Encrypted);


    ////////////////////////////////////
    //Put the cipher in decryption mode
    desCipher.init(Cipher.DECRYPT_MODE, desKey);

    //Decrypt and output the original string
    byte[] decryptedBytes = desCipher.doFinal(encryptedBytes);
    String decryptedText = new String(decryptedBytes);
    System.out.println("Decrypted text: " + decryptedText);
  }
}
```

Input

```
String message1 = "Four score and seven years ago";
```

Result

```
Encrypted text: P0FT6N3XXrohtsz7OLh3FGYY0wErkPIur1DP6Csbj4g=
Decrypted text: Four score and seven years ago
```

SYMMETRIC

Block versus Stream Algorithms

Stream vs. Block

- Specific algorithms for each

SYMMETRIC (BLOCK)

Block

Predefined content length

- ★ Well-known end to the content
- ★ Files on disk
- ★ Inefficient when padding

DES

Data **E**ncryption **S**tandard

- Block cipher
- Banking industry
- DES is known to be **broken**

3DES

Triple **D**ata **E**ncryption **S**tandard

- Block cipher
- a.k.a DESede
- Basically three passes of DES
- Reasonably strong

Blowfish

- Block cipher
- Unpatented *(intentionally)*
- Secure replacement for DES
  - Faster than DES
- 32 to 448 bit keys
- Overshadowed by AES

AES

Advanced **E**ncryption **S**tandard

- Block cipher
- Government standard
  - ***Rijndael*** algorithm
    (Joan Daemen, Vincent Rijmen)
  - 4 years of evaluation
  - Final in December 2000
- Very Secure

SYMMETRIC (STREAM)

Stream

Unknown content length

- ★ Streaming video
- ★ Streaming voice
- ★ But still can do **files**

RC4

Rivest's Code 4

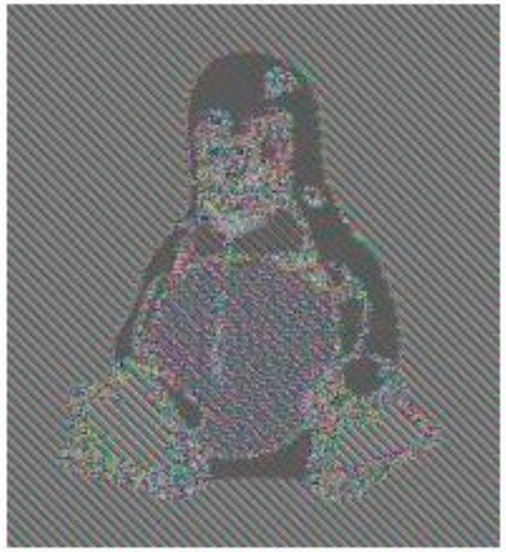
- Stream cipher
- Trademarked *(name, but not algorithm)*
- Used by
  - Browsers in SSL, TLS
  - WiFi in WEP, WPA
  - BitTorrent
  - ssh
  - Microsoft RDP
  - PDF

ENCRYPTED = SAFE, RIGHT?

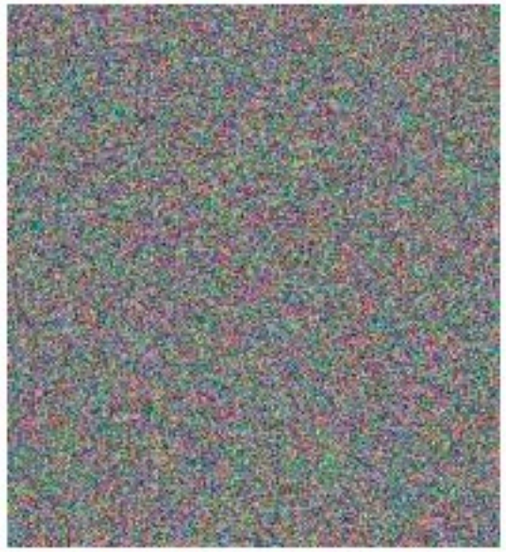
information leakage from encrypted data

Encrypted isn't enough?

Original



Encrypted using ECB mode

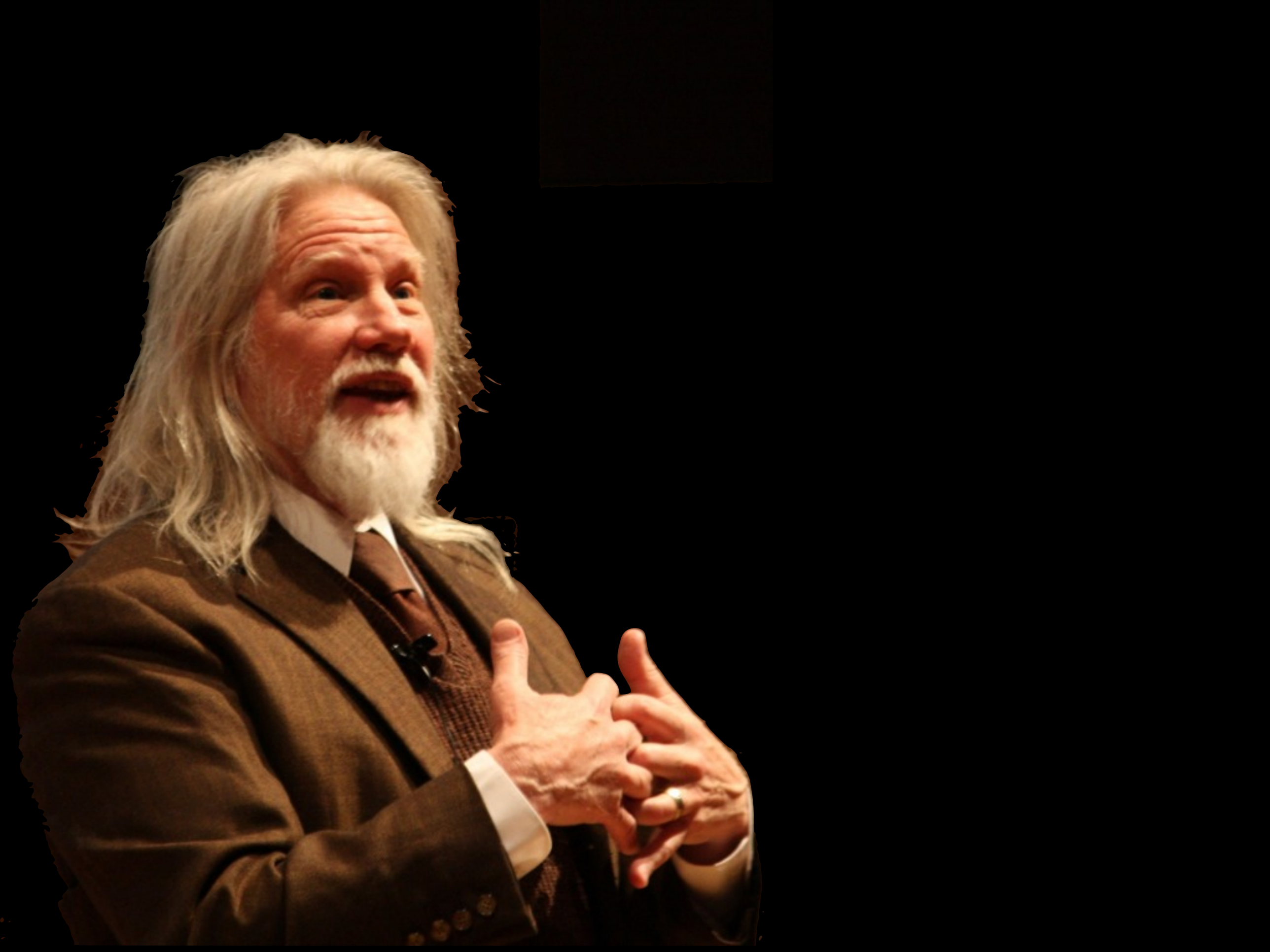


Other modes than ECB results in pseudo-randomness

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

SECURE KEY EXCHANGE

securely swapping symmetric keys



Diffie-Hellman

Key Agreement Protocol

- Alice & Bob independently generate the shared (session) key
- Published 1976, but invented earlier

DH Diagrammed

predetermined and openly **shared**

g = *random* **p** = *prime*
g = **11** **p** = **23**

picks **a** = 6

$A = g^a \bmod p$

$\mathbf{9} = 11^6 \bmod 23$

A=9

picks **b** = 4

$B = g^b \bmod p$

$\mathbf{13} = 11^4 \bmod 23$

B=13

$K = B^a \bmod p$

$\mathbf{6} = 13^6 \bmod 23$

$K = A^b \bmod p$

$\mathbf{6} = 9^4 \bmod 23$

Encryption can begin

What's wrong with this?

RANDOM NUMBERS

Seed the machine

SecureRandom

- ★ java.security.SecureRandom
- ★ Cryptographically strong pseudo-random number generator (PRNG)
- ★ “Unable to distinguish from a true random source”
- ★ Used in combination with many ciphers

```
package com.ambientideas;

import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

/**
 * Use the SecureRandom java security class to generate
 * a more expensive, but cryptographically secure random number.
 */
public class SecureRandomNumber
{
  public static void main( String[] args ) throws
                                        NoSuchAlgorithmException
  {
    //Do the expensive one time setup of the
    // random number generator instance
```

```java
import java.security.SecureRandom;

/**
 * Use the SecureRandom java security class to generate
 * a more expensive, but cryptographically secure random number.
 */
public class SecureRandomNumber
{
  public static void main( String[] args ) throws
                                        NoSuchAlgorithmException
  {
    //Do the expensive one time setup of the
    // random number generator instance
    SecureRandom prng = SecureRandom.getInstance("SHA1PRNG");

    //Get the next random number
    String randomNum = new Integer( prng.nextInt() ).toString();

    System.out.println("Random number: " + randomNum);
  }
}
```

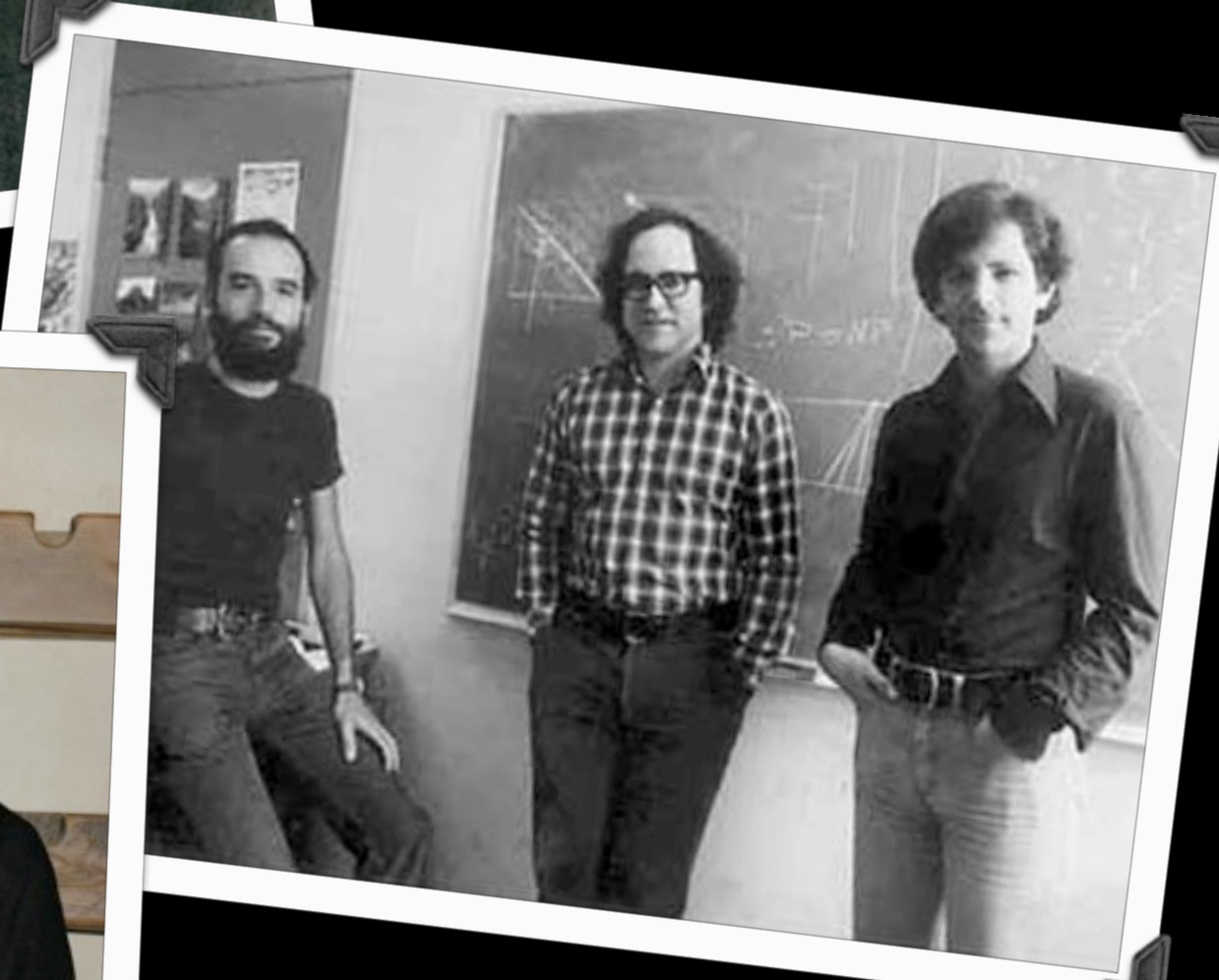
```
 * a more expensive, but cryptographically secure random number.
 */
public class SecureRandomNumber
{
  public static void main( String[] args ) throws
                                        NoSuchAlgorithmException
  {
    //Do the expensive one time setup of the
    // random number generator instance
    SecureRandom prng = SecureRandom.getInstance("SHA1PRNG");

    //Get the next random number
    String randomNum = new Integer( prng.nextInt() ).toString();

    System.out.println("Random number: " + randomNum);
  }
}
```

Result

Random number: 1633471380

ASYMMETRIC

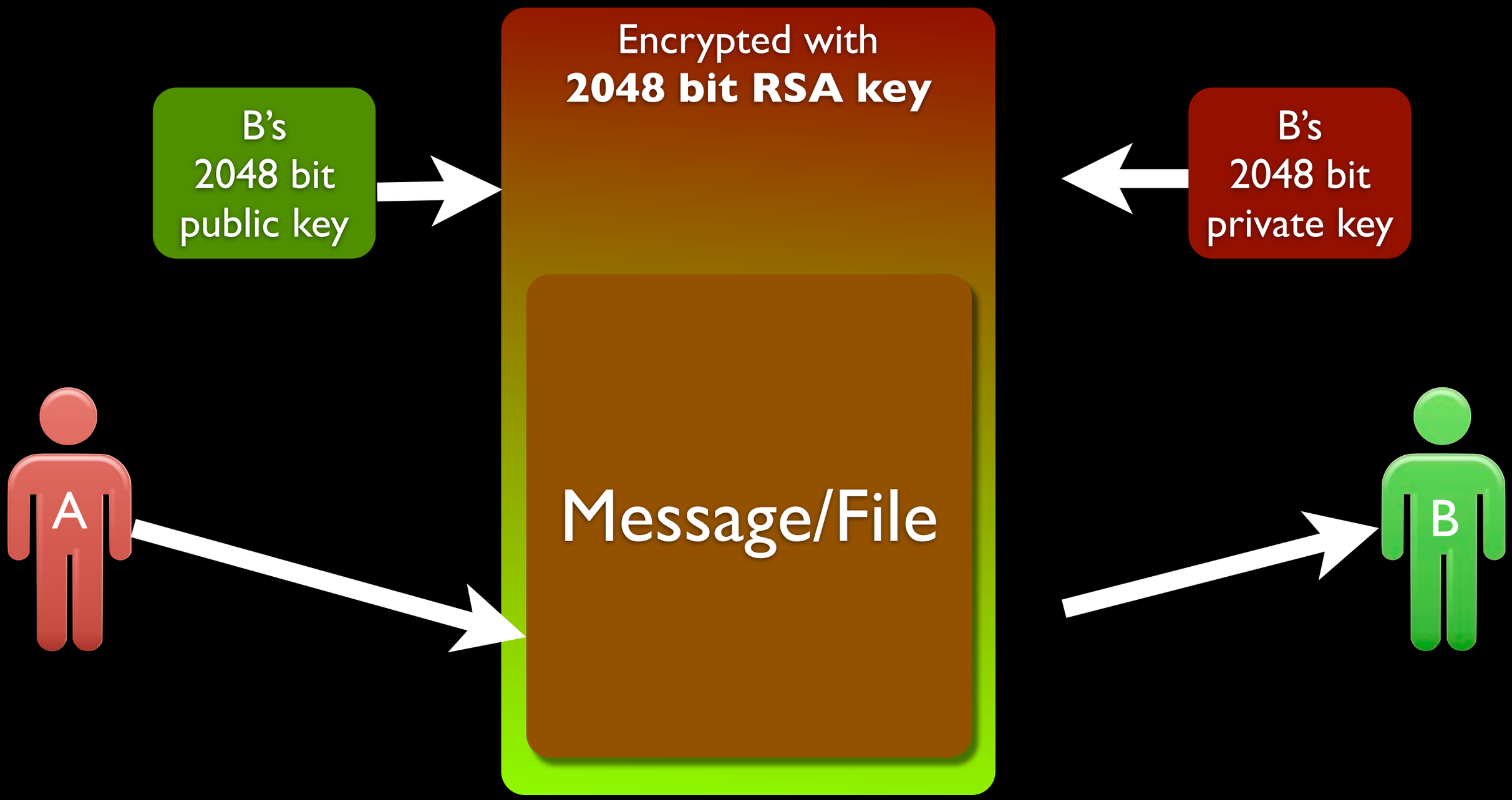
Throwing away keys
faster than an intern locksmith



RSA

- Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman
- Published in 1978
- M.I.T. Patented in 1983
- Patent Expired in 2000

RSA


B's
2048 bit
public key
Encrypted with
2048 bit RSA key
B's
2048 bit
private key
Message/File
A
B

```
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

import sun.misc.BASE64Encoder;
```

```
/**
 * Use the SecureRandom java security class to generate
 * a more expensive, but cryptographically secure random
number.
```

```
  public static void main( String[] args ) throws
NoSuchAlgorithmException, NoSuchProviderException,
IOException, NoSuchPaddingException, InvalidKeyException,
IllegalBlockSizeException, BadPaddingException
  {
    final String message1 = "Four score and seven years ago";

    // Generate the Key Pair
    final KeyPairGenerator keyGen =
       KeyPairGenerator.getInstance("RSA");

    final SecureRandom random =
        SecureRandom.getInstance("SHA1PRNG", "SUN");
    keyGen.initialize(1024, random);

    KeyPair pair = keyGen.generateKeyPair();

    final PrivateKey privKey = pair.getPrivate();
    final PublicKey pubKey = pair.getPublic();


    //Encrypt using the private key
    Cipher rsa = Cipher.getInstance("RSA/OFB/PKCS1Padding");
    rsa.init(Cipher.ENCRYPT_MODE, pubKey);
    byte[] encryptedBytes = rsa.doFinal(message1.getBytes());
    BASE64Encoder b64e = new sun.misc.BASE64Encoder();
```

```
KeyPair pair = keyGen.generateKeyPair();

final PrivateKey privKey = pair.getPrivate();
final PublicKey pubKey = pair.getPublic();

//Encrypt using the private key
Cipher rsa = Cipher.getInstance("RSA/OFB/PKCS1Padding");
rsa.init(Cipher.ENCRYPT_MODE, pubKey);
byte[] encryptedBytes = rsa.doFinal(message1.getBytes());
BASE64Encoder b64e = new sun.misc.BASE64Encoder();
String base64Encrypted = b64e.encode(encryptedBytes);
System.out.println("Encrypted text: " + base64Encrypted);

//Decrypt using the private key
rsa.init(Cipher.DECRYPT_MODE, privKey);
byte[] decryptedBytes = rsa.doFinal(encryptedBytes);
String decryptedText = new String(decryptedBytes);
System.out.println("Decrypted text: " + decryptedText);
  }
}
```

```
final PublicKey pubKey = pair.getPublic();


//Encrypt using the private key
Cipher rsa = Cipher.getInstance("RSA/OFB/PKCS1Padding");
rsa.init(Cipher.ENCRYPT_MODE, pubKey);
byte[] encryptedBytes = rsa.doFinal(message1.getBytes());
BASE64Encoder b64e = new sun.misc.BASE64Encoder();
String base64Encrypted = b64e.encode(encryptedBytes);
System.out.println("Encrypted text: " + base64Encrypted);

//Decrypt using the private key
rsa.init(Cipher.DECRYPT_MODE, privKey);
byte[] decryptedBytes = rsa.doFinal(encryptedBytes);
String decryptedText = new String(decryptedBytes);
System.out.println("Decrypted text: " + decryptedText);
    }
}
```

Input

```
String message1 = "Four score and seven years ago";
```

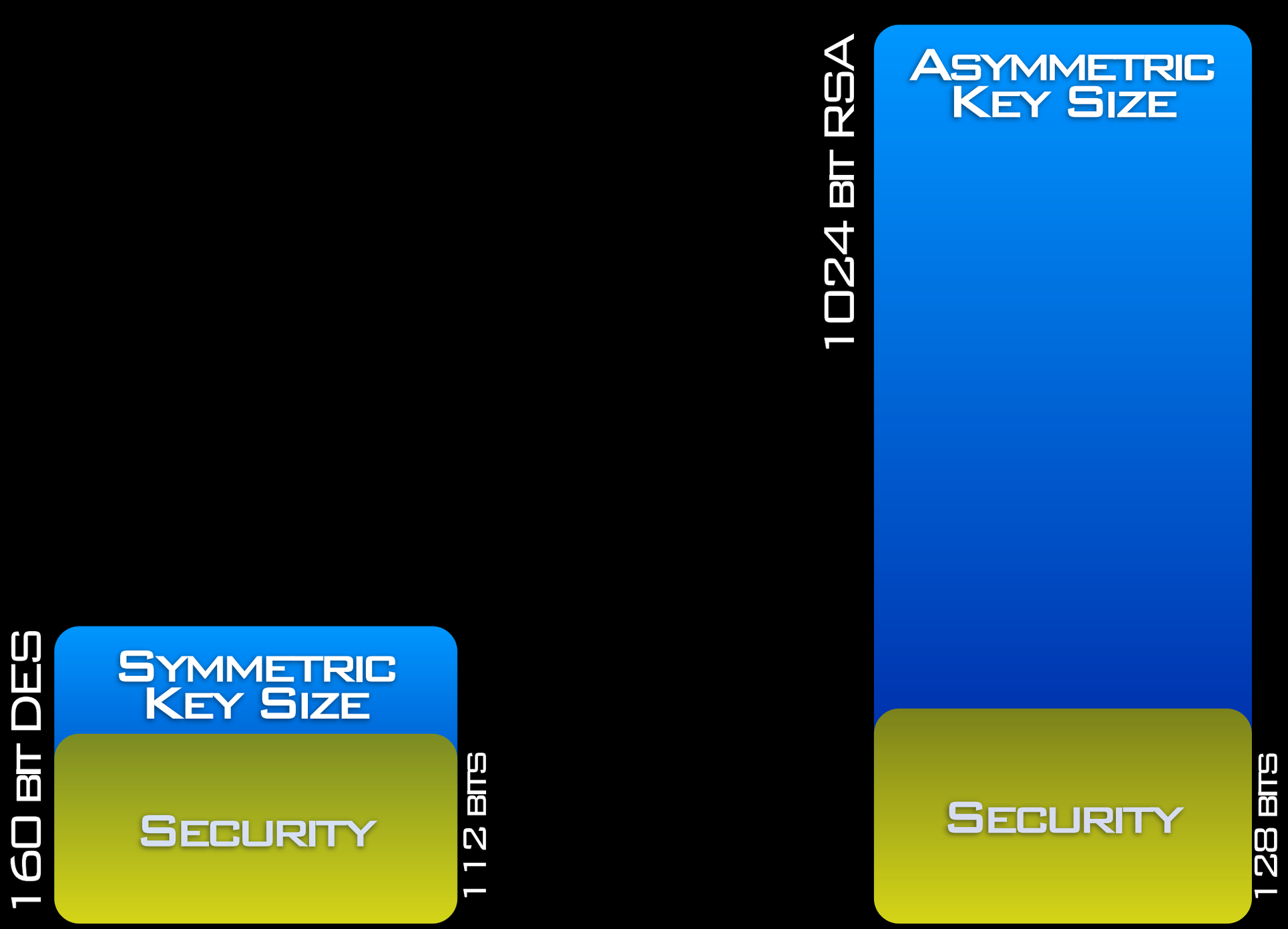
Result

```
Encrypted text: A8Is+4r7sDn28fD6IQvZiR5JxPs/vh7UnXrF38acJt6R/
ARisj/zLtC7Xn6iJgNQPhc16wkVZhCF
em7oNoim+ooTUDDZQ+E3qP6y/
DZJGkLBoZuZVLeLAW1LUtHSzduRUOg1uMynJz14wxzwfV8wfRwf
atpySkOhGqWS63bPNRs=
Decrypted text: Four score and seven years ago
```

BLENDED

symmetric with a twist of asymmetric

Key Size & Security

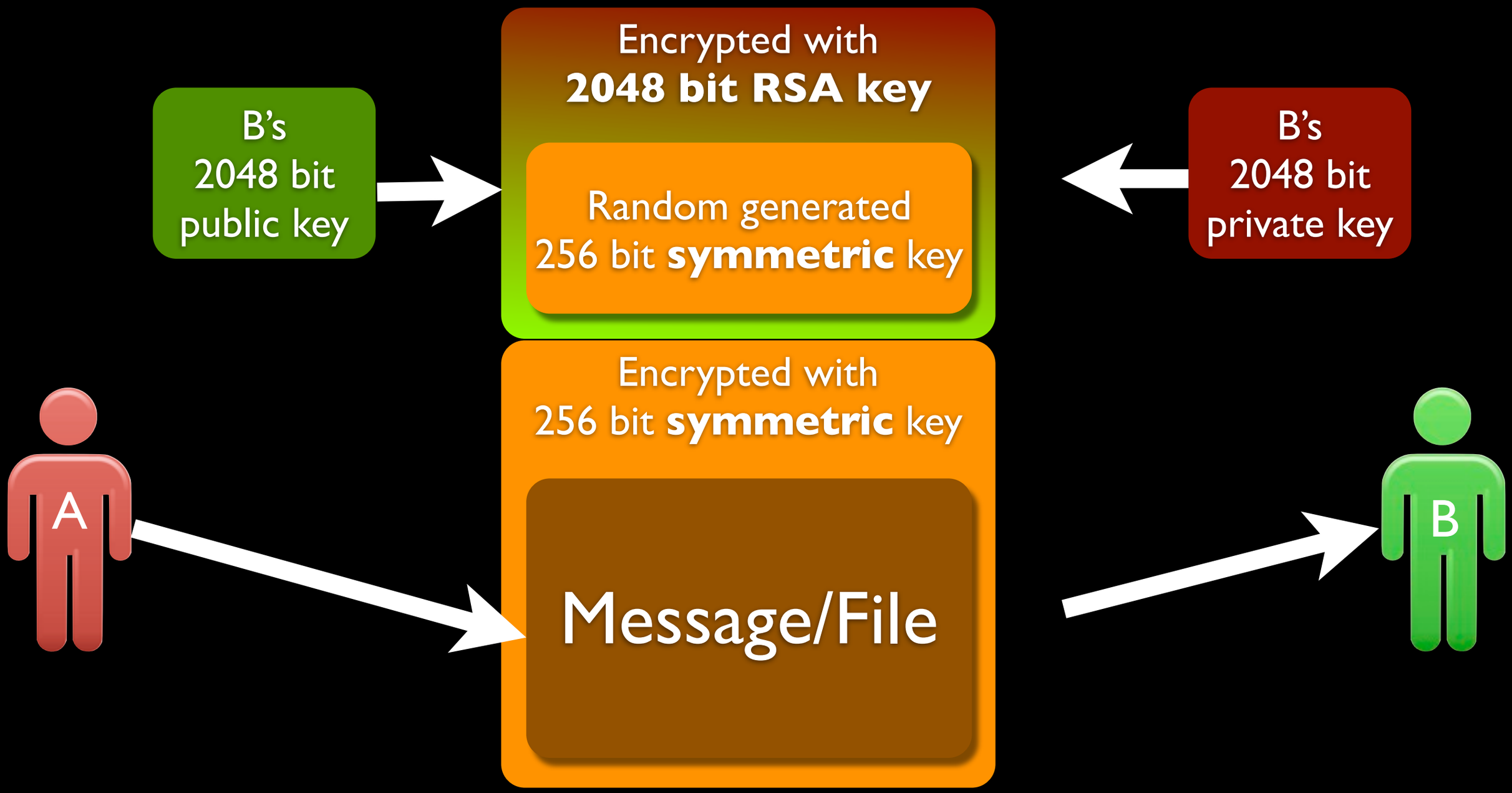


Encryption Speed

asymmetric can be **1000x slower** than **symmetric**

What do we do about that?

PGP


Encrypted with
2048 bit RSA key
B's
2048 bit
public key
Random generated
256 bit **symmetric** key
B's
2048 bit
private key
Encrypted with
256 bit **symmetric** key
A
Message/File
B

OTHER FRAMEWORKS

and alternative JCE providers

Bouncy Castle

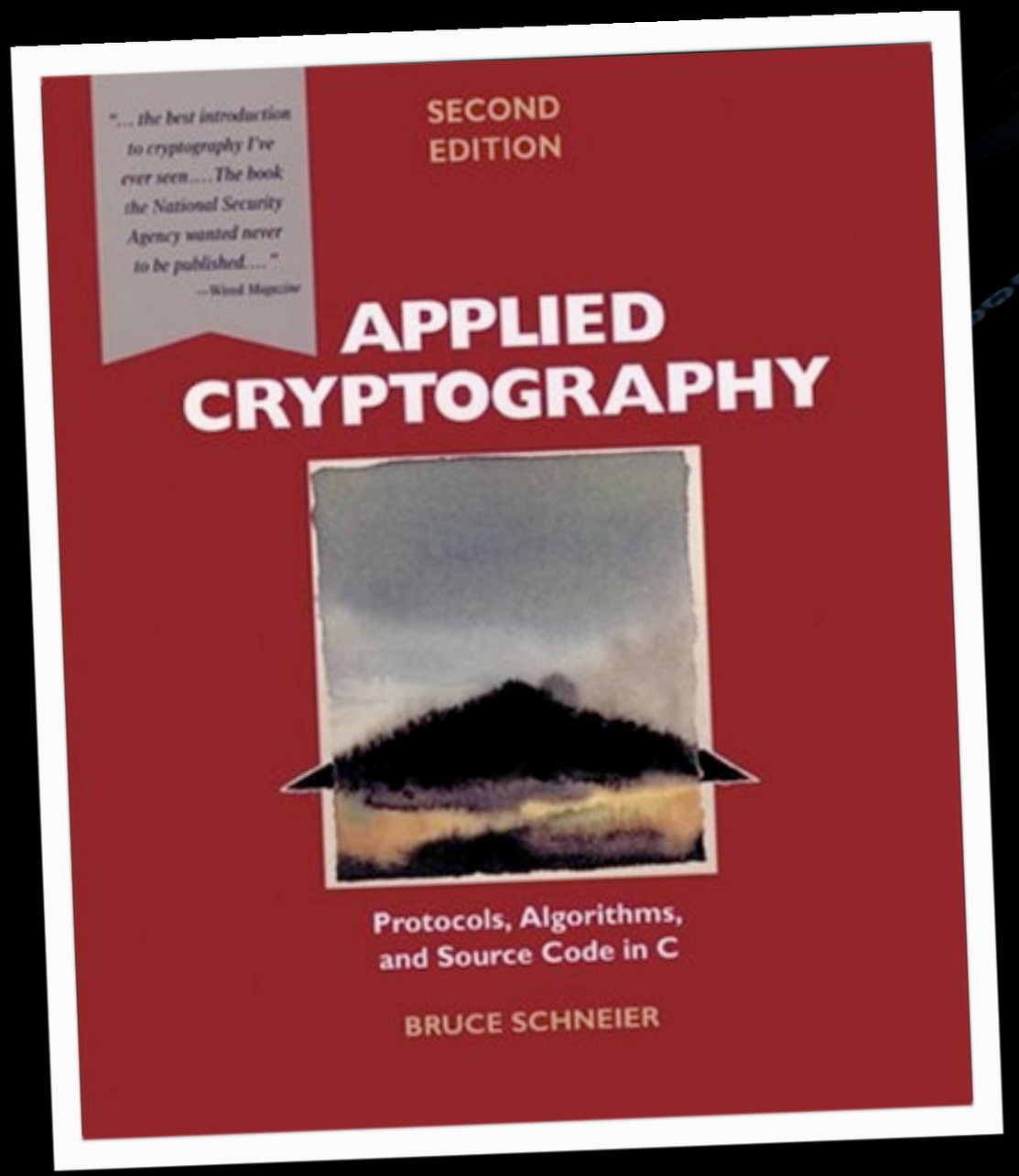
White box implementation of JCE & Emerging Algorithms

GNU

Open source library

JASYPT

Frictionless Java encryption


"... the best introduction to cryptography I've ever seen.... The book the National Security Agency wanted never to be published...."
—Wired Magazine
SECOND EDITION
APPLIED CRYPTOGRAPHY
Protocols, Algorithms, and Source Code in C
BRUCE SCHNEIER

DON'T FORGET THE HUMANS

http://xkcd.com/538/

ENCRYPTION BOOT CAMP

Security is the Mission

Matthew McCullough

Email matthewm@ambientideas.com
Twitter @matthewmccull
Blog http://ambientideas.com/blog

Samples http://github.com/matthewmccullough/encryption-jvm-bootcamp