Cryptography 101

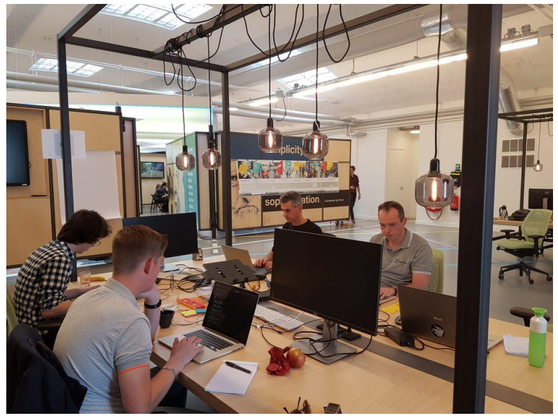
For  Developers

@MichelSchudel

CRAFTSMEN

SOFTWARE ENGINEERS

Introduction



CRAFTSMEN
SOFTWARE ENGINEERS

Belastingdienst

bol.com

www.craftsmen.nl
michel@craftsmen.nl

@MichelSchudel


BAHCO
5m
BAHCO
18 86
PROFESSIONAL

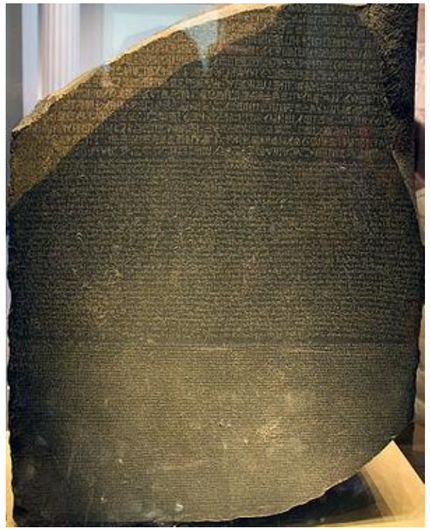

Zcash
Ripple
0.25%
Bitcoin
1.71%
Ethereum
14.91%

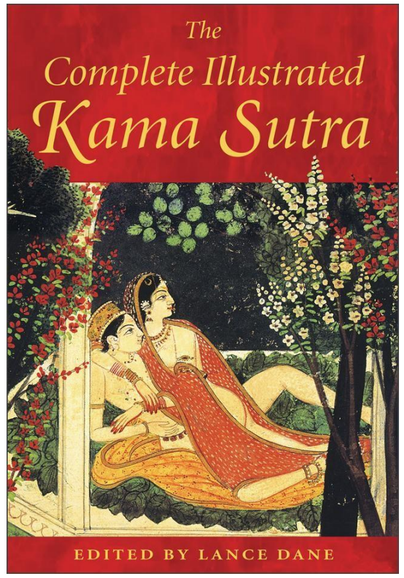
Basic understanding of cryptographic concepts and how they’re done in

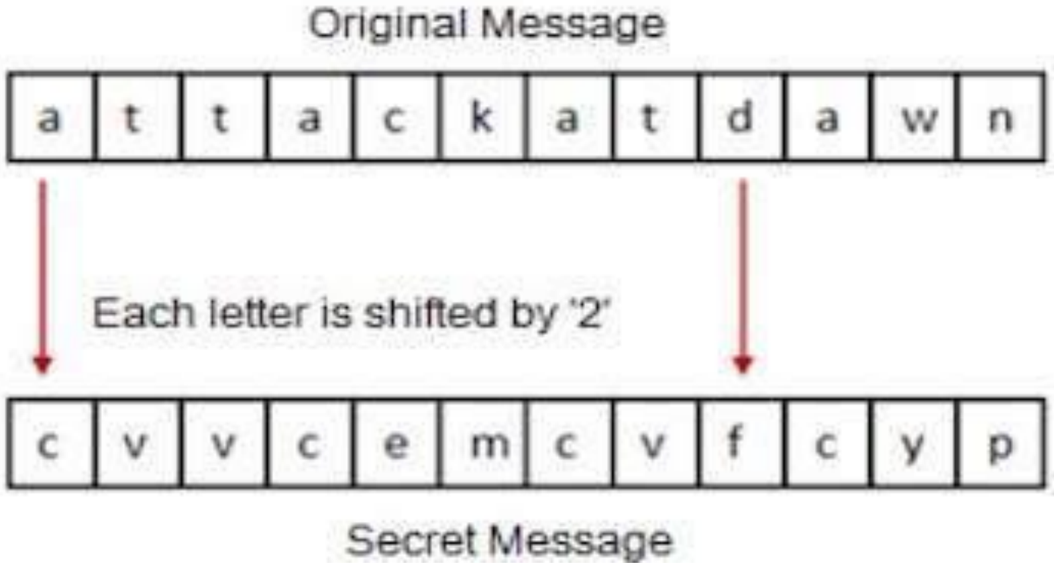
Java

Cryptography (or cryptology; from Greek κρυπτός, kryptos, "hidden, secret"; and γράφ, gráph, "writing", or -λογία, -logia, respectively) is the practice and study of hiding information.

https://en.wikipedia.org/wiki/Outline_of_cryptography



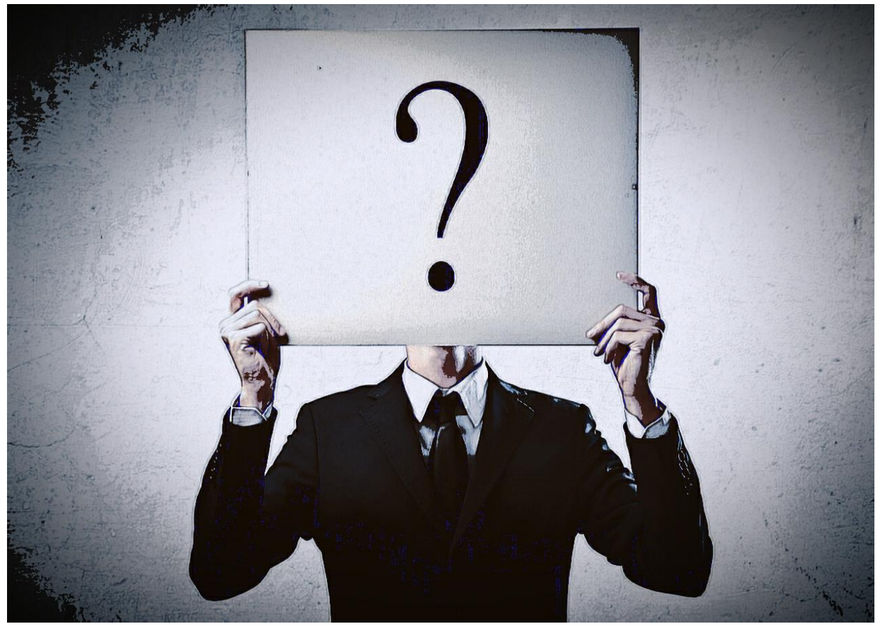

The
Complete Illustrated
Kama Sutra
EDITED BY LANCE DANE


Original Message
a t t a c k a t d a w n
Each letter is shifted by '2'
c v v c e m c v f c y p
Secret Message

What does cryptography solve?

Confidentiality

Integrity

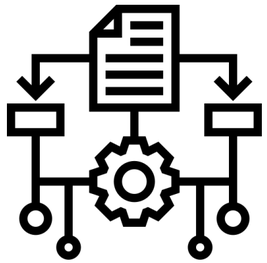


Authenticity

Key components of cryptography

Data

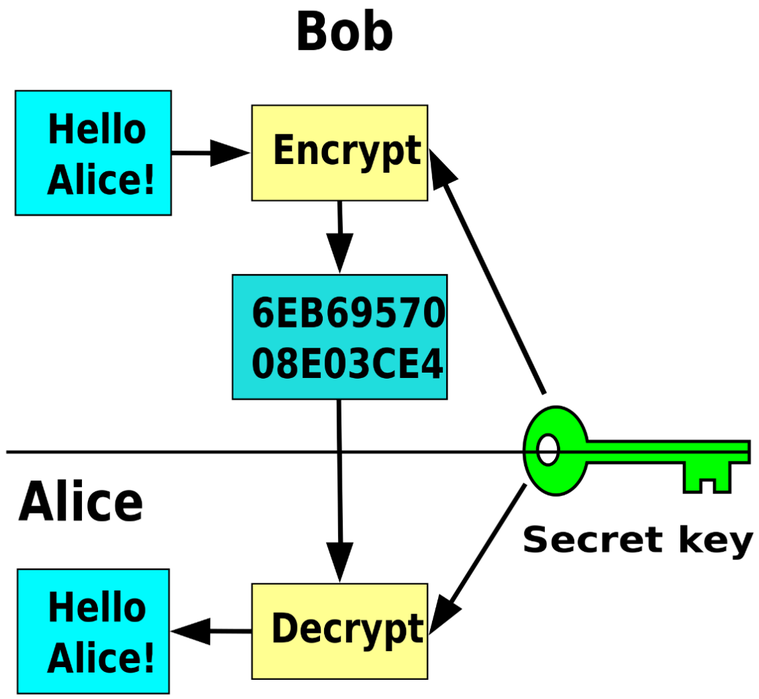
Algorithm



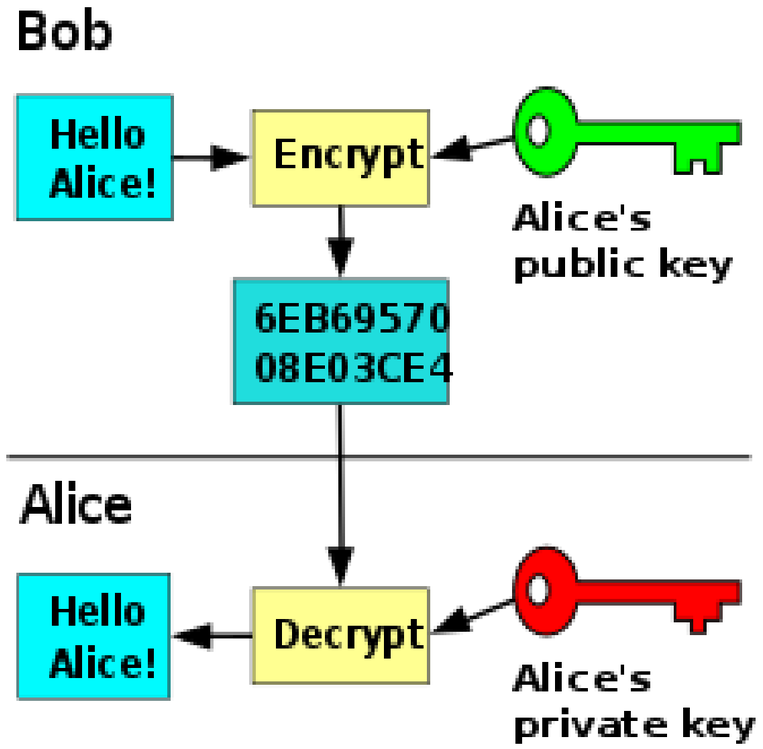
Key

Contemporary cryptography

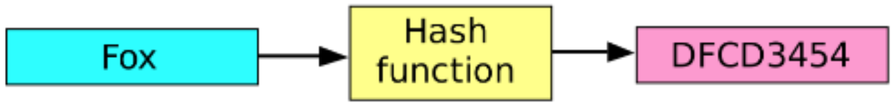
Symmetric encryption



Asymmetric encryption



Hashing

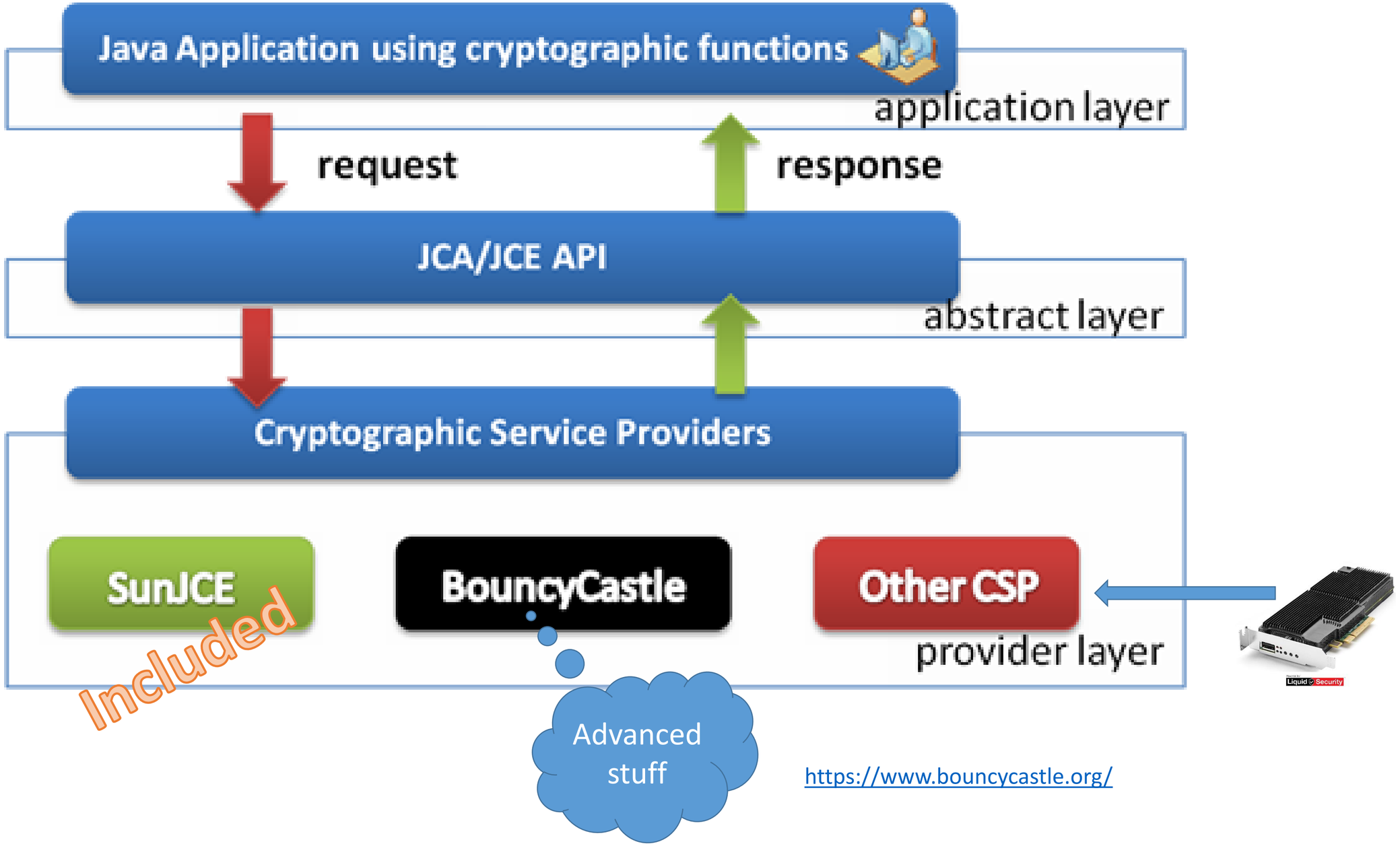


JCA

(Java Cryptography Architecture)

JCE

(Java Cryptography Extension)


Java Application using cryptographic functions
application layer
request
response
JCA/JCE API
abstract layer
Cryptographic Service Providers
SunJCE
Included
BouncyCastle
Other CSP
provider layer
Advanced stuff
https://www.bouncycastle.org/

How to plug in a security provider

```
Security.addProvider(new BouncyCastleProvider());
```

(extends `java.security.Provider`)

Or register it in *`java.security`* file

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=org.bouncycastle.jce.provider.BouncyCastleProvider
```

- **JDK8 and previous:**
  - Update `jre/lib/security/java.security`
  - Place specific provider JAR in `lib/ext`
- **JDK9 and onwards:**
  - Update `conf/security/java.security`
  - Place specific provider JAR on *classpath*

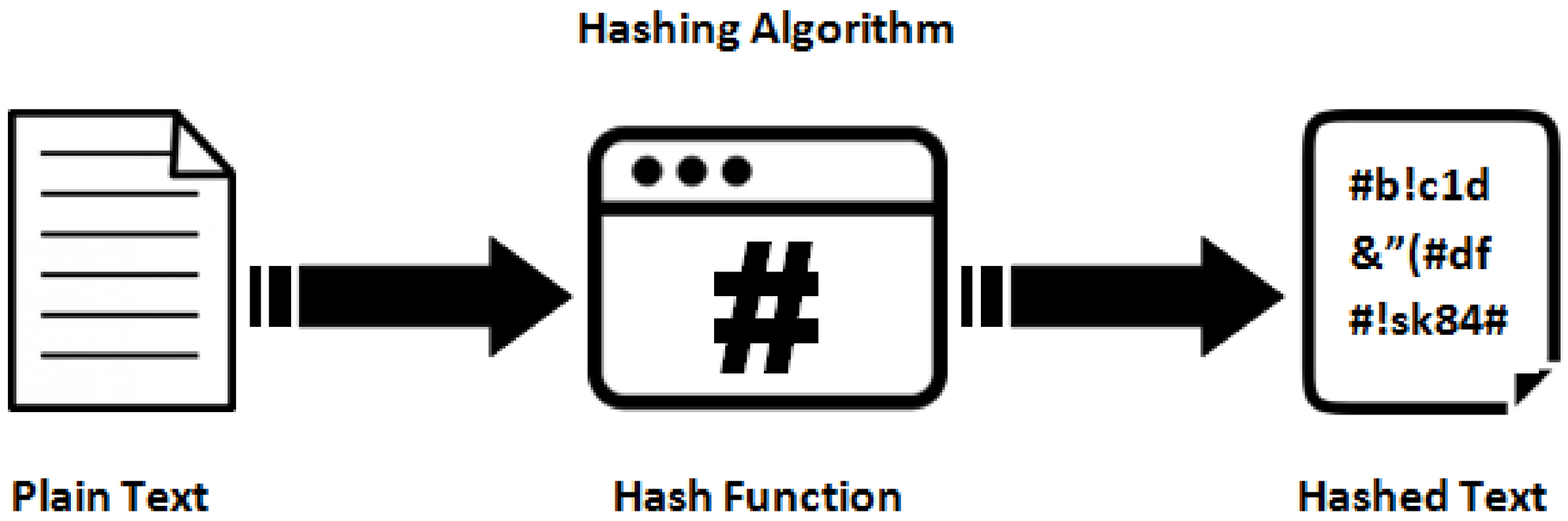
JCA /JCE API Summary

Interface / class	Function
`Security`	Provider configuration
`KeyGenerator`	Generates symmetric keys
`Key`	Represents key material
`Cipher`	Encryption algorithm
`SecretKeyFactory`	Converts symmetric key material to SecretKey abstraction
`KeyPairGenerator`	Generates public / private keys
`KeyPair`	Public / private keypair
`KeyFactory`	Converts public / private material to Key abstraction
`KeyStore`	Storing mechanism for keys and certificates
`MessageDigest`	Hashing
`HMAC`	Combines hashing and encryption

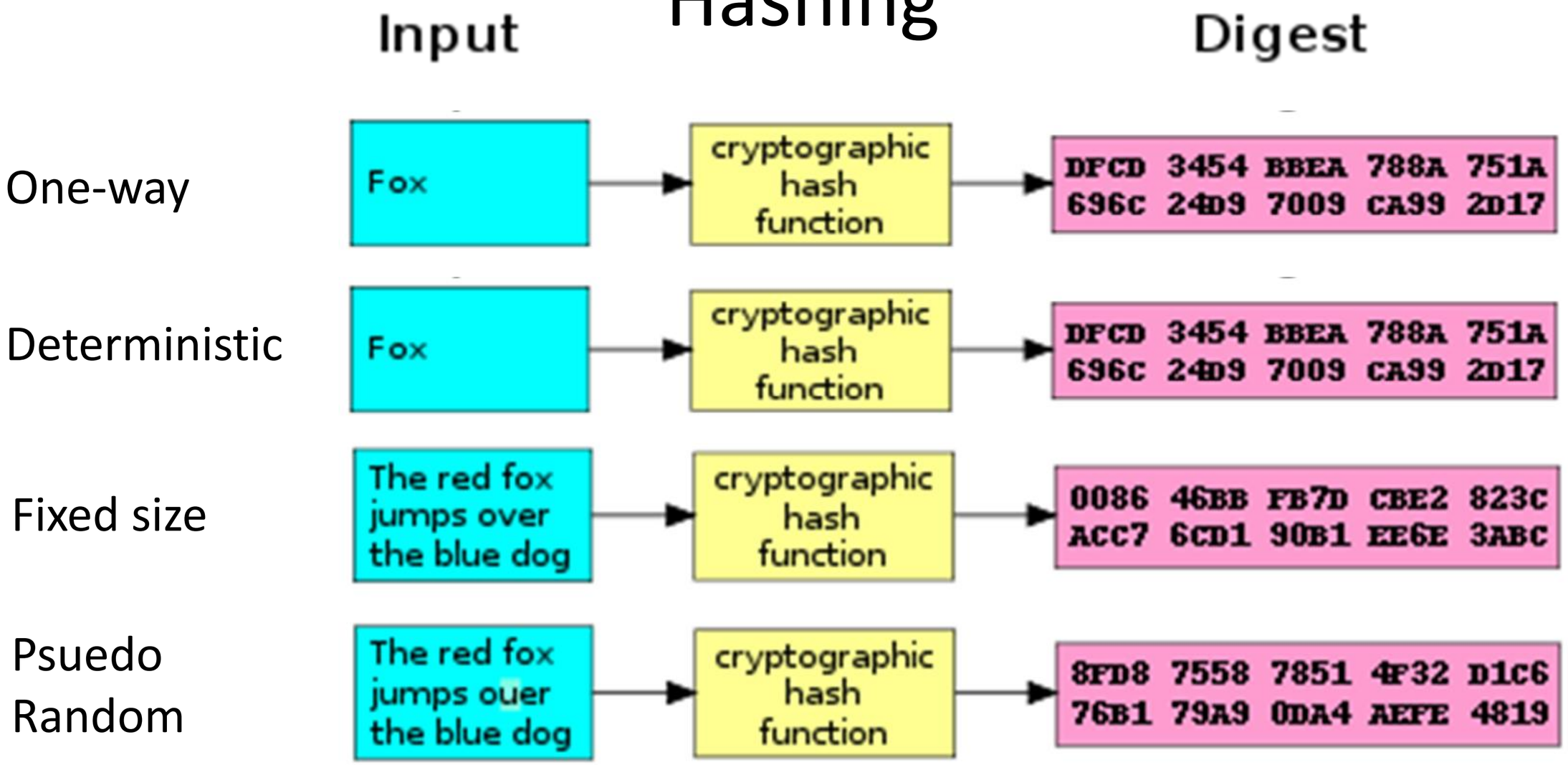
Today we’re going to cover:

1. Hashing
2. Symmetric encryption
3. Asymmetric encryption
4. Digital signatures
5. Certificates

Hashing


Hashing Algorithm
#b!c1d
&"(#df
#!sk84#
Plain Text
Hash Function
Hashed Text

Hashing


Input
Digest
One-way
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
Deterministic
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
Fixed size
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
Psuedo Random
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819

Hashing

Algorithms

MD-5 (hash size 128 bits)

SHA-1 (hash size 160 bits)

SHA-256 (hash size 256 bits)

https://www.computerworld.com/article/3173616/the-sha1-hash-function-is-now-completely-unsafe.html

Collisions

data1 <> data2 -> Hash(data1) == Hash(data2)

Current BitCoin hashrate: 300 quadrillion SHA-256 hashes per second

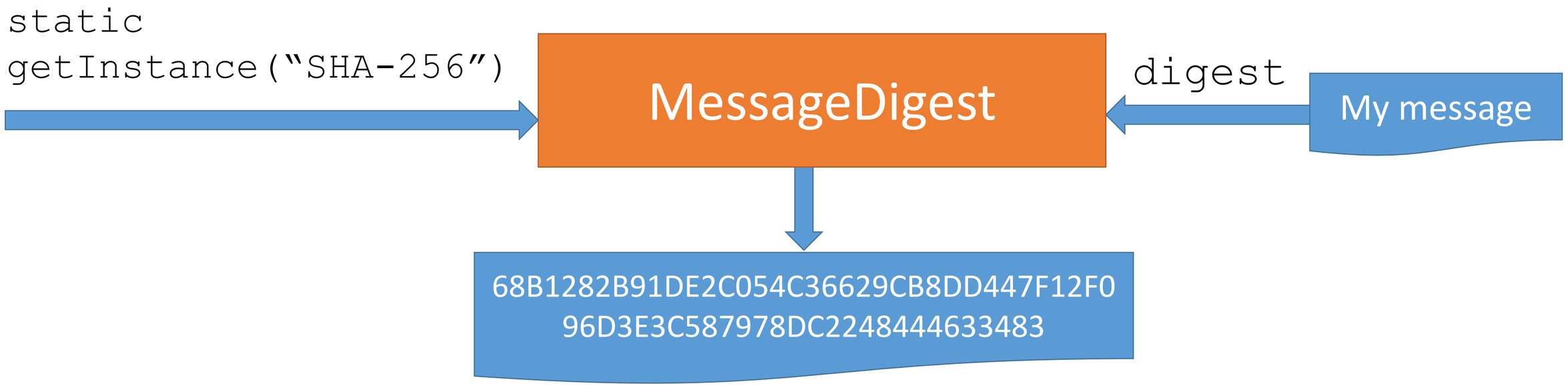
(300x10^15)

Collision attack: calculate 2^128 hashes

Time needed at Bitcoin’s current rate: 3.6x10^13 years

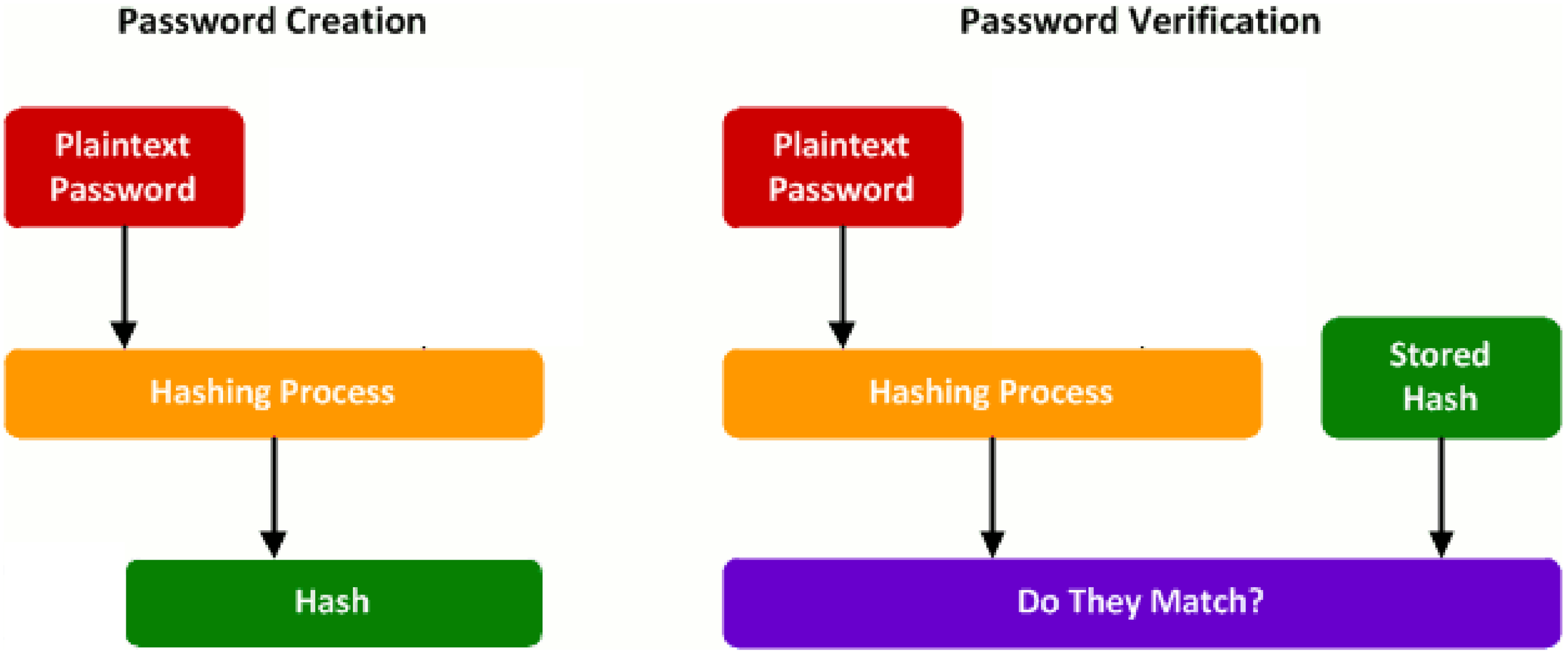
Age of the universe: 13.7x10^9 years

Hashing in Java

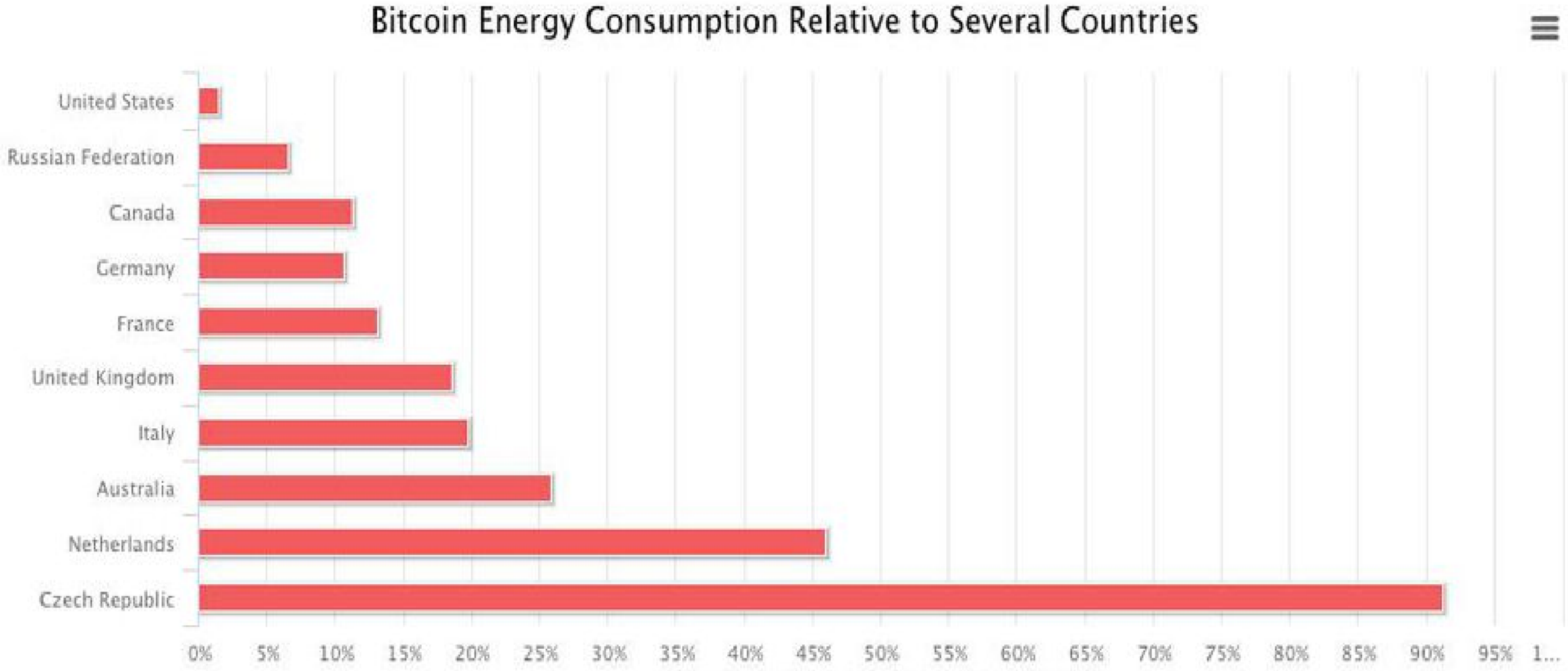


DEMO

Application: password hashing

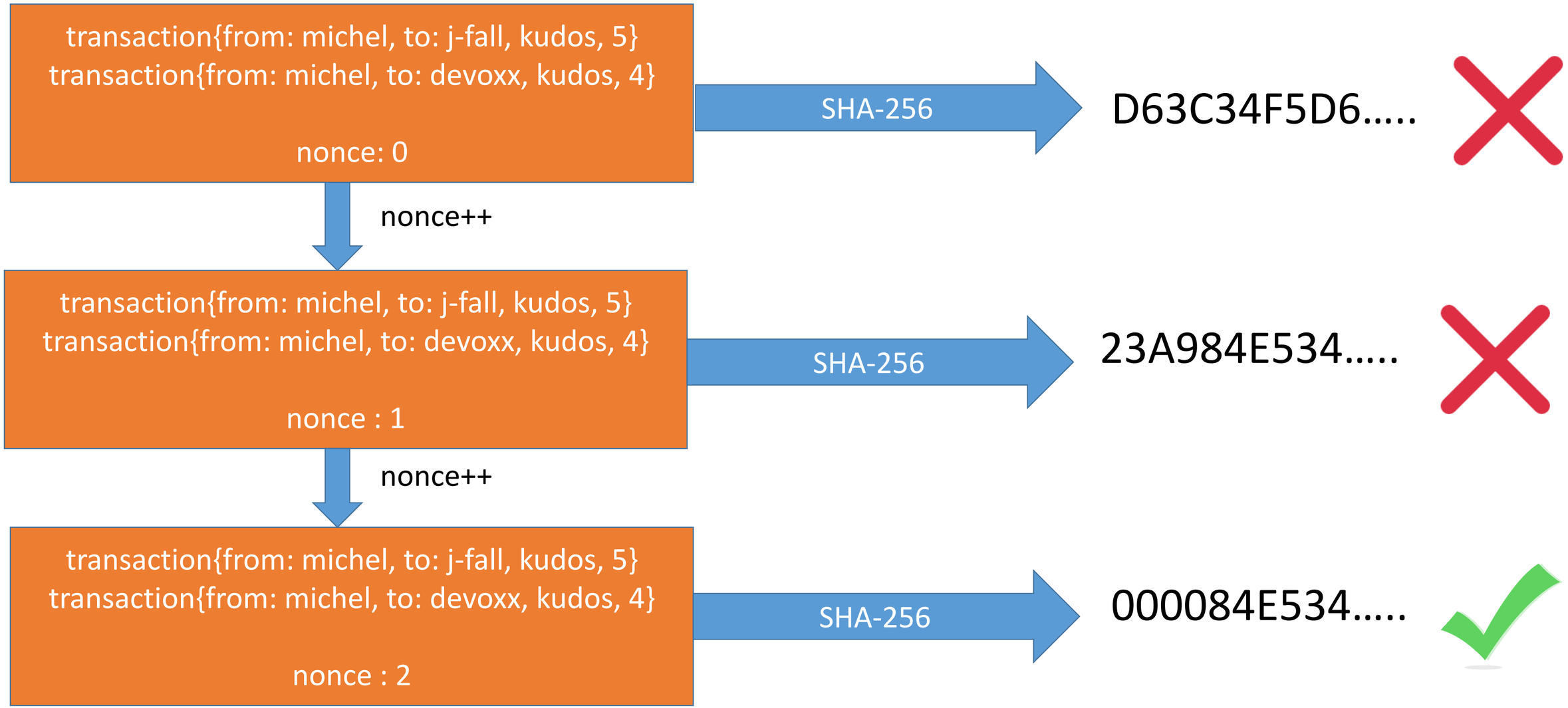


Application: bitcoin block mining



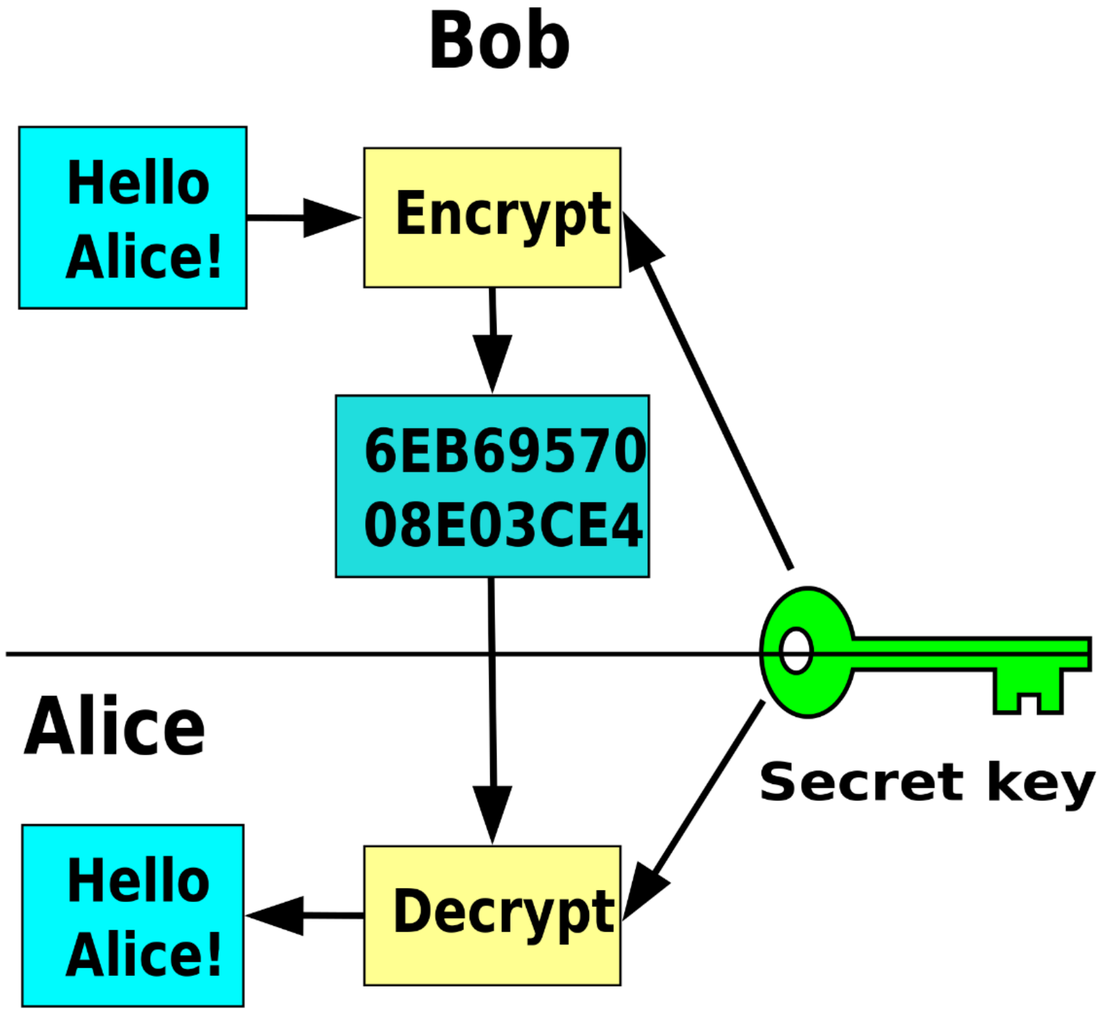
Block mining

Hash of a block should start with 0000….

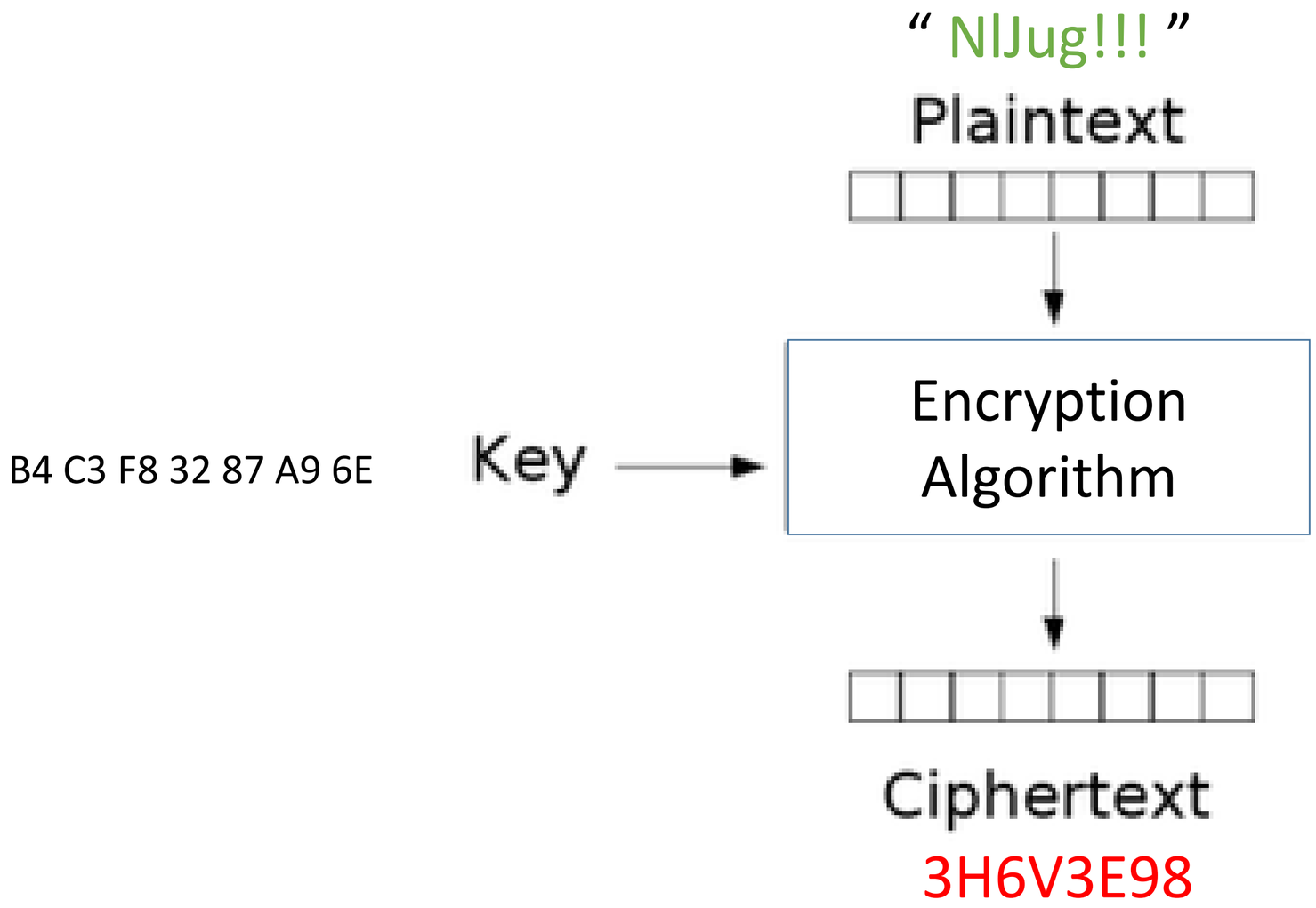


1. Hashing ✓
2. Symmetric encryption 🔍
3. Asymmetric encryption
4. Digital signatures
5. Certificates

Symmetric encryption



Symmetric encryption



Symmetric encryption

Algorithms

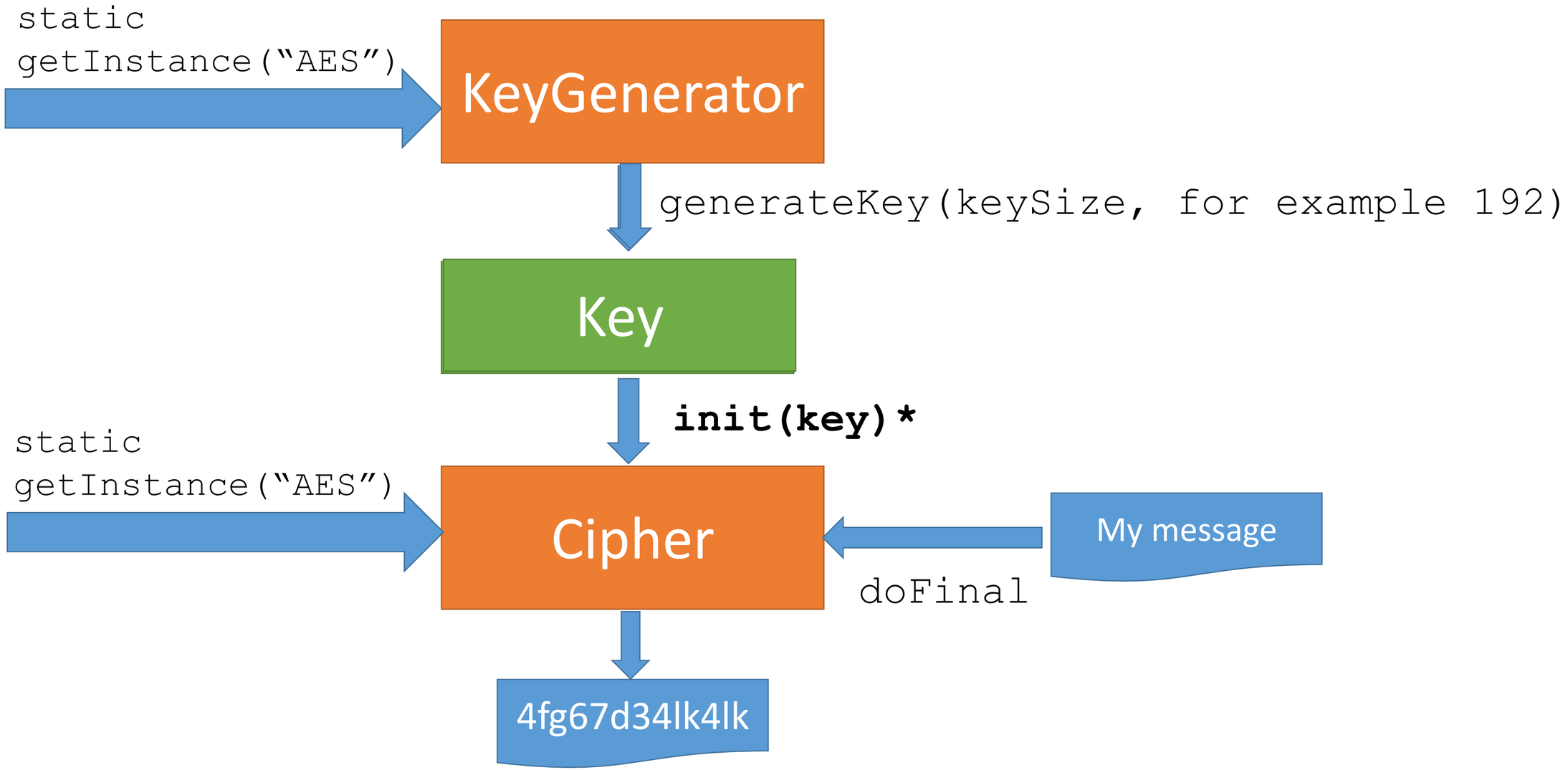
- DES (Data Encryption Standard)
  - block size 64 bits
  - key size 56 bits

HACKED

- AES (Advanced Encryption Standard)
  - block size 128 bits
  - key size 128/ 192 / 256 bits

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Symmetric encryption in Java



Key size restrictions

`"Illegal key size..."` (calling init with keysize 192)

Limit on key sizes outside USA

- Install "unlimited strength jurisdiction policy files"
- /jre/lib/security/local_policy.jar, export_policy.jar

Java 8u152 includes unlimited strength jars

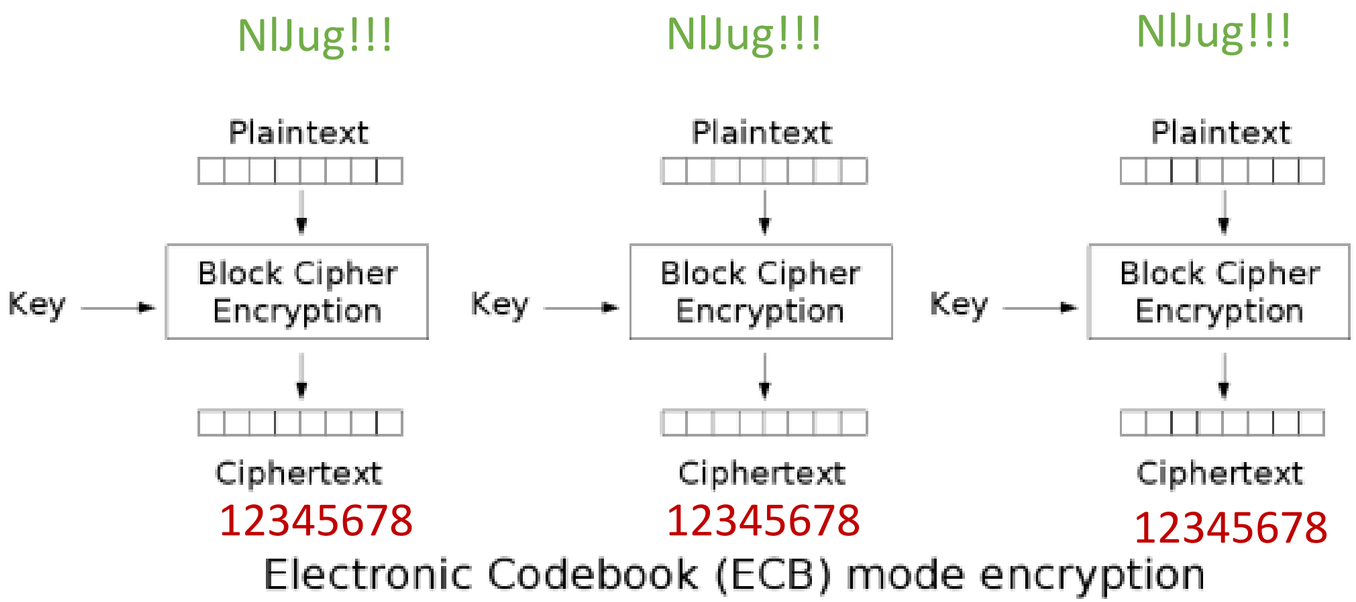
Java 8u162 and up have unlimited strength by default

https://golb.hplar.ch/2017/10/JCE-policy-changes-in-Java-SE-8u151-and-8u152.html

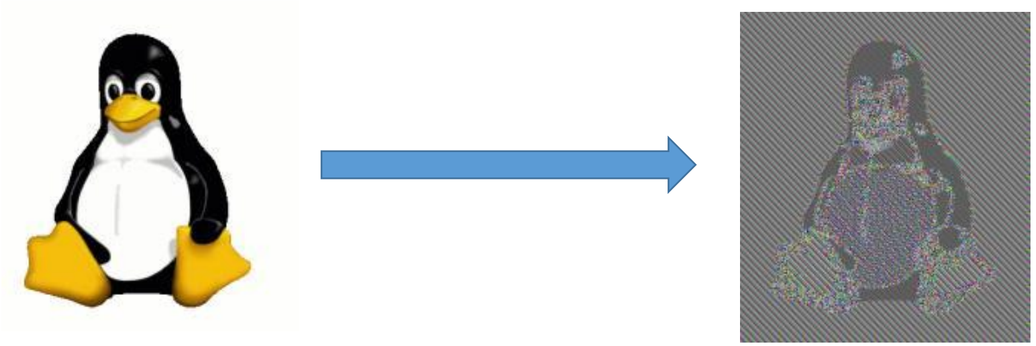
DEMO

AES Encryption (with ECB)

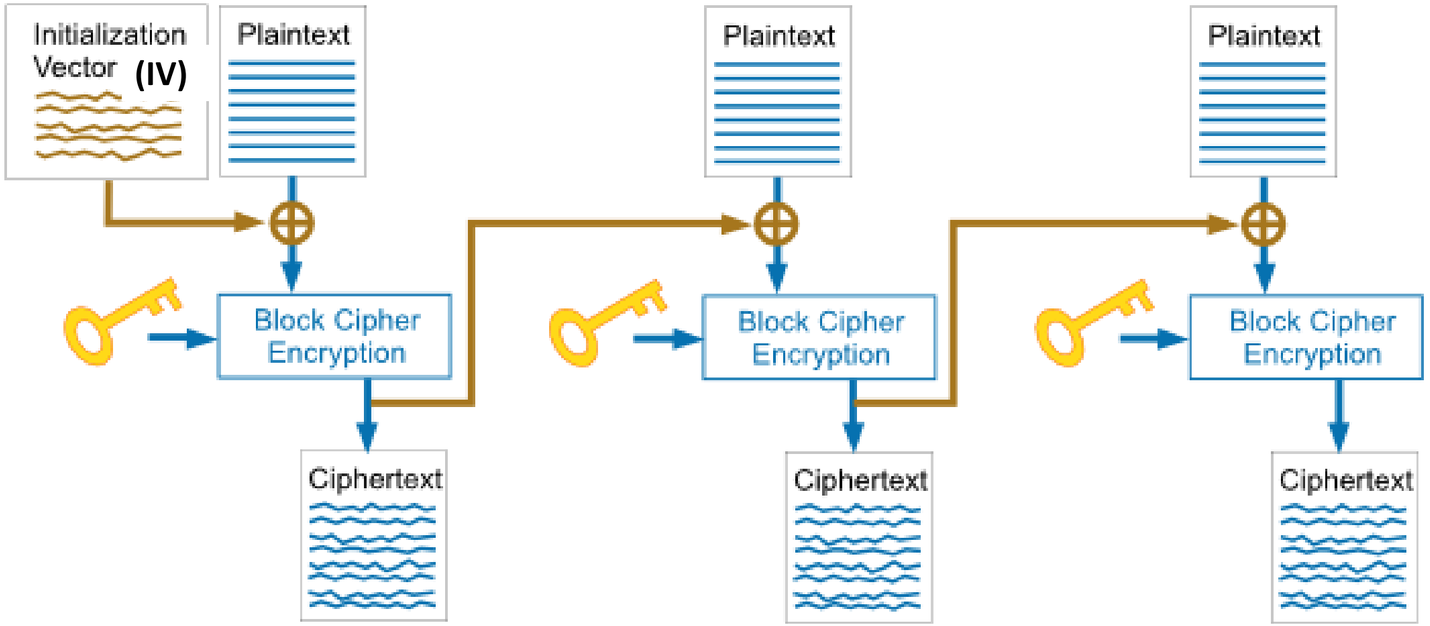
Electronic CopyBook Encryption (ECB)

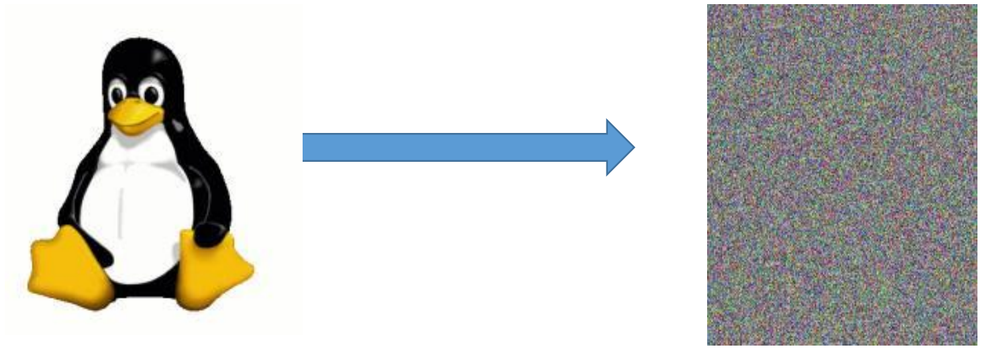


Electronic Codebook (ECB) mode encryption

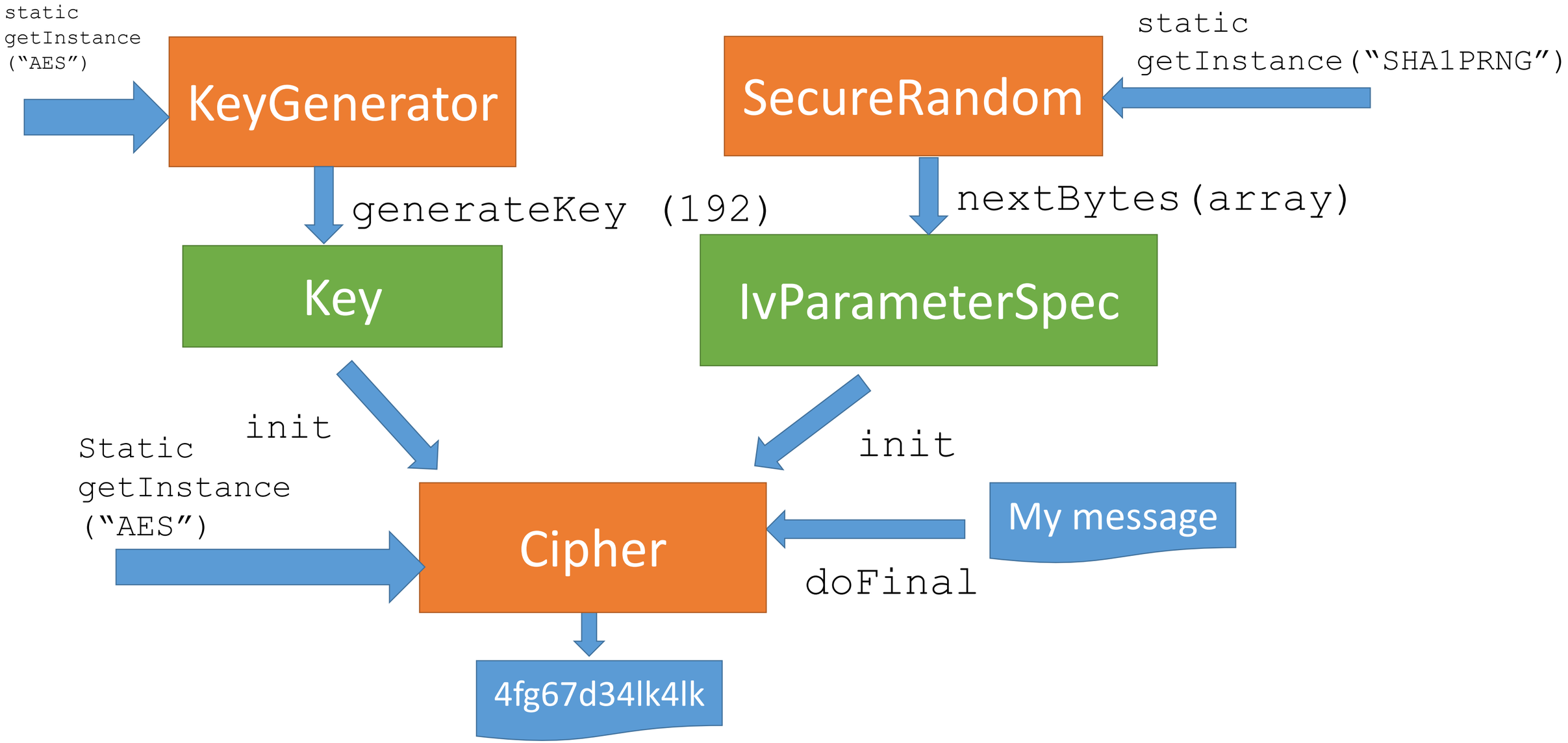


Cipher Block Chaining





Symmetric encryption with IV in Java



DEMO

AES Encryption with CBC

Application: Encrypt a Wallet (or zip file)

“myPassword”

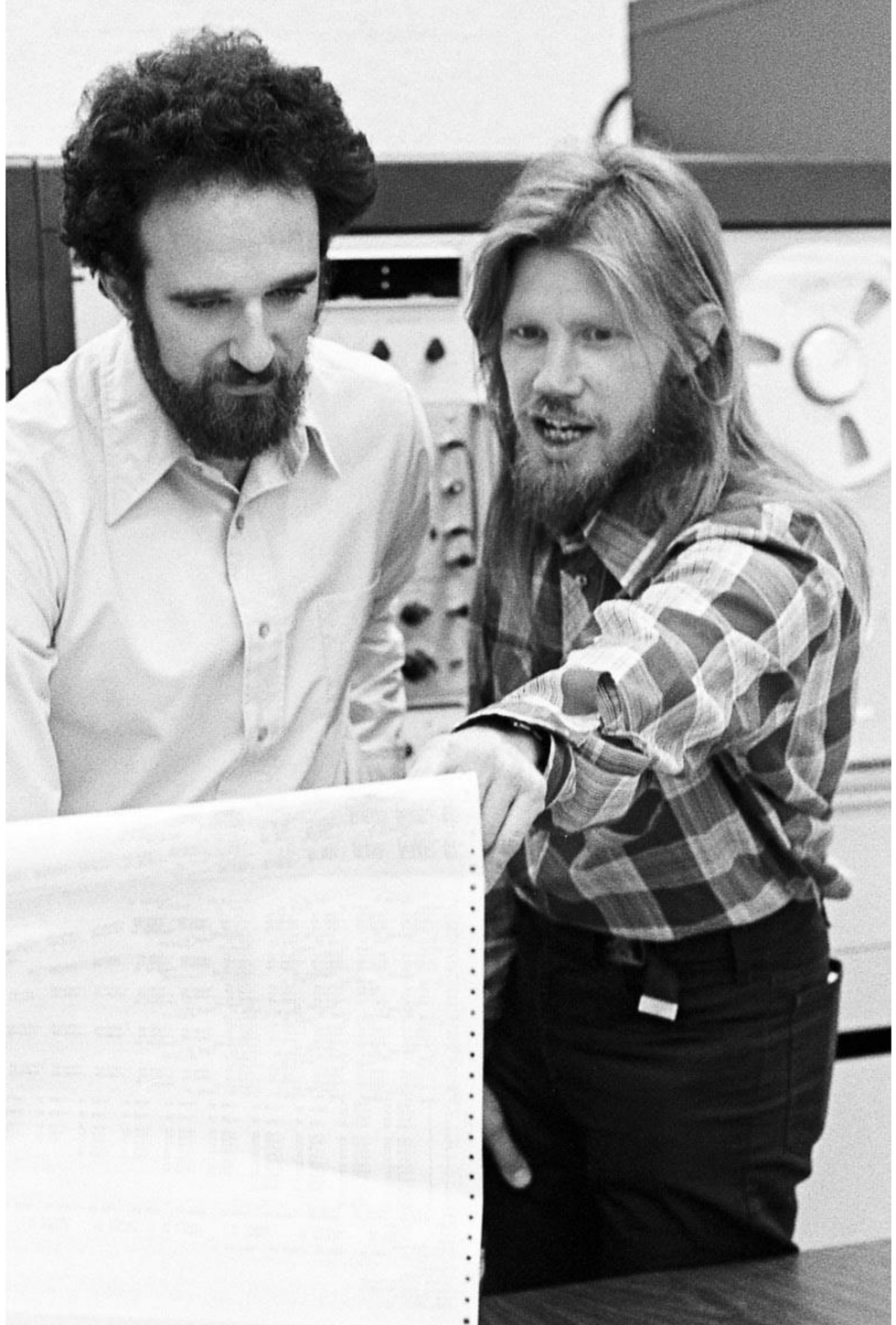
PBKDF2 SHA256

AES

Password-Based Key Derivation Function

Key exchange

Martin Hellman



Whitfield Diffie

Agree upon two numbers:

P Prime Number 13

G Generator of P 6

Private = 5

Randomly generate a Private Key

Private = 4

(6^{5}) MOD 13

(7776) MOD 13

Public = 2

Calculate Public Key:

$(G^{Private})$ MOD P

(6^{4}) MOD 13

(1296) MOD 13

Public = 9

Exchange Public Keys

(9^{5}) MOD 13

(59049) MOD 13

Shared Secret = 3

Calculate the Shared Secret

$(Shared\ Public^{Private})$ MOD P

(2^{4}) MOD 13

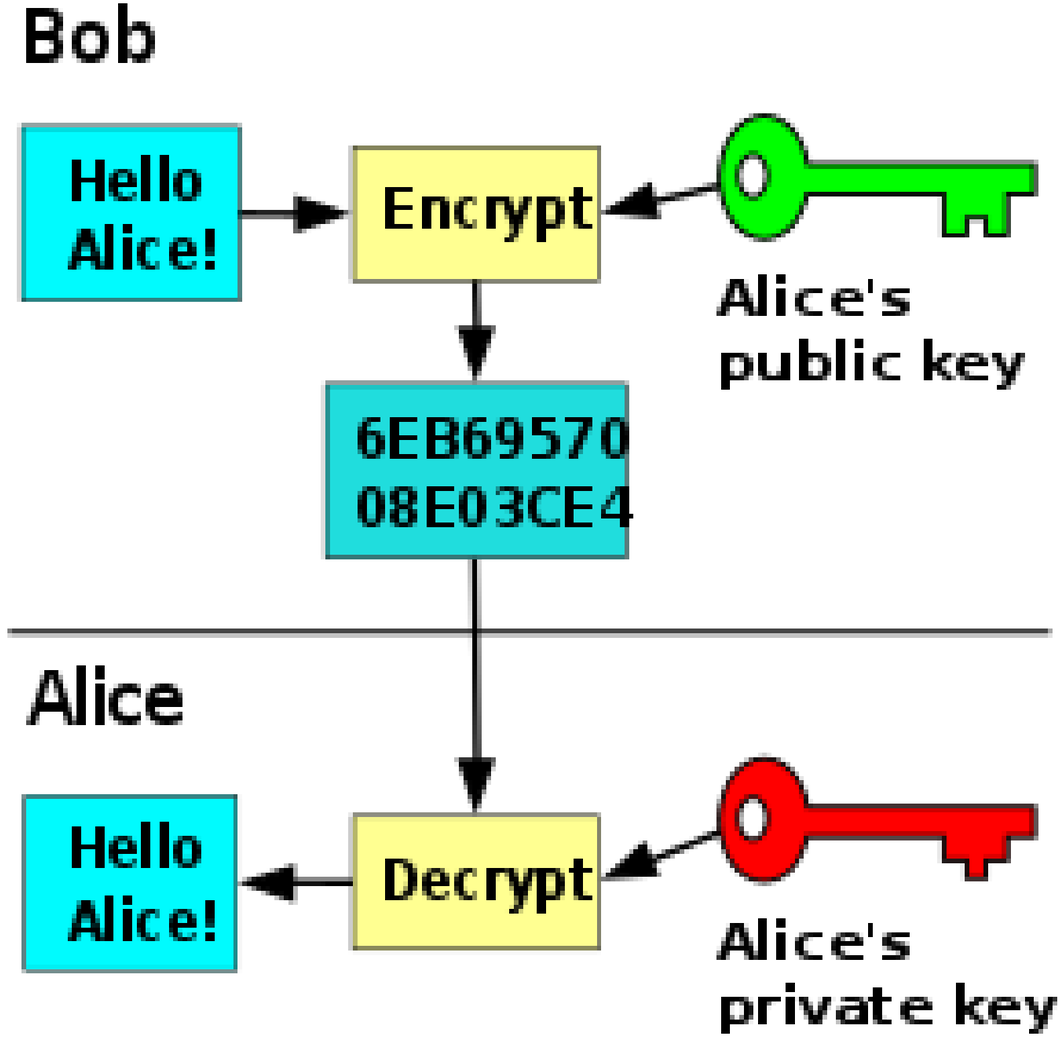
(16) MOD 13

Shared Secret = 3

PRACTICAL NETWORKING .NET

1. Hashing ✓
2. Symmetric encryption ✓
3. Asymmetric encryption 🔍
4. Digital signatures
5. Certificates

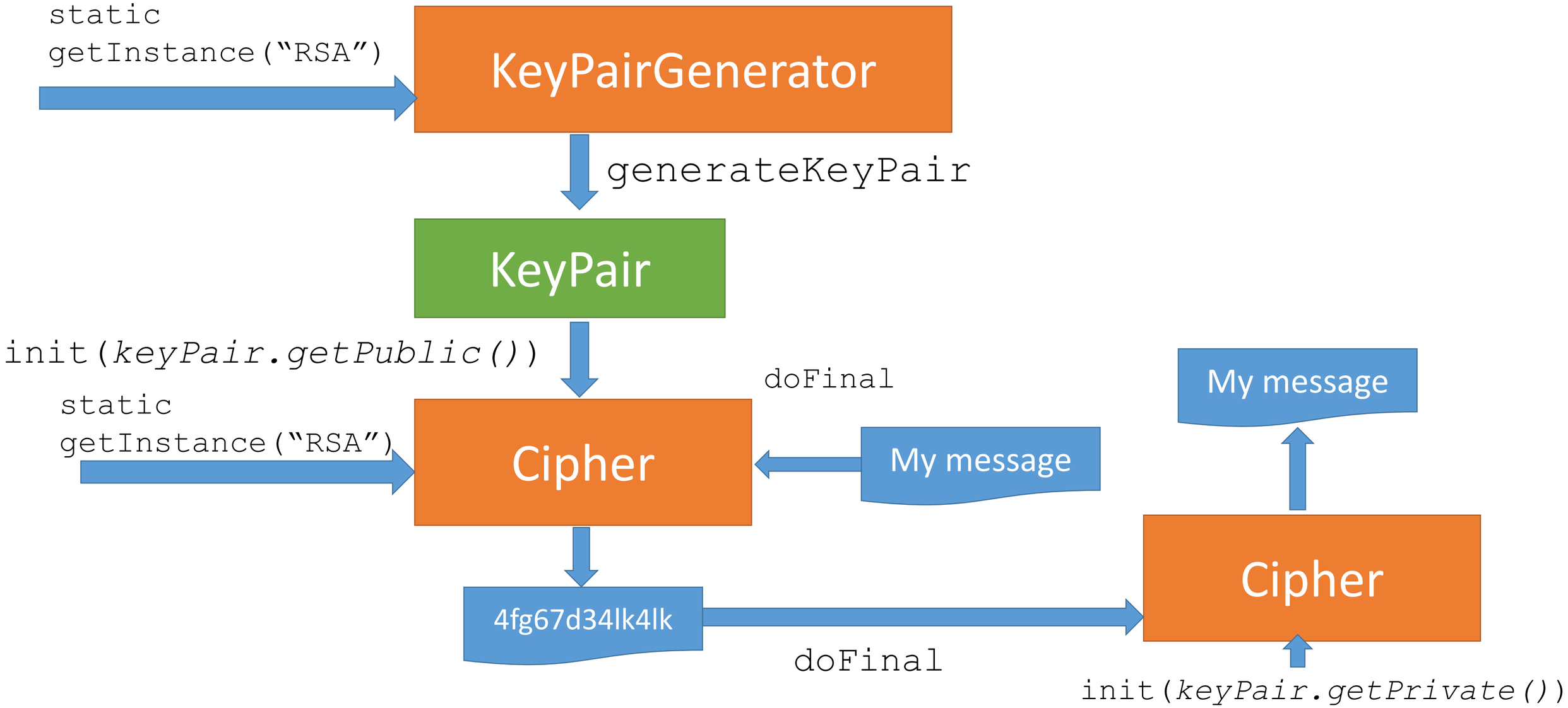
Asymmetric keys
(public key cryptography)



Public key algorithms

- *Diffie-Hellman key agreement protocol*
- RSA (Rivest-Shamir-Adleman) – key size 1024,2048, 3072
- ECC (Elliptic Curve Cryptography) – keysize 128

Asymmetric cryptography in Java



DEMO

RSA Encryption

Don’t use Asymmetric encryption to encrypt large blocks of data

RSA in particular is slow

…but you CAN use it for…

- Agreeing on symmetric keys (Diffie-Hellman)
- Encryting / Decrypting symmetric keys
- Encrypting hashes, or *message digests* (Digital Signature)

1. Hashing ✓
2. Symmetric encryption ✓
3. Asymmetric encryption ✓
4. Digital signatures 🔍
5. Certificates

Digital signatures

Confidentiality

Integrity

Authenticity

Digital signatures

message

“Hi, Alice!”

HASH

message

“I need money, Alice!”

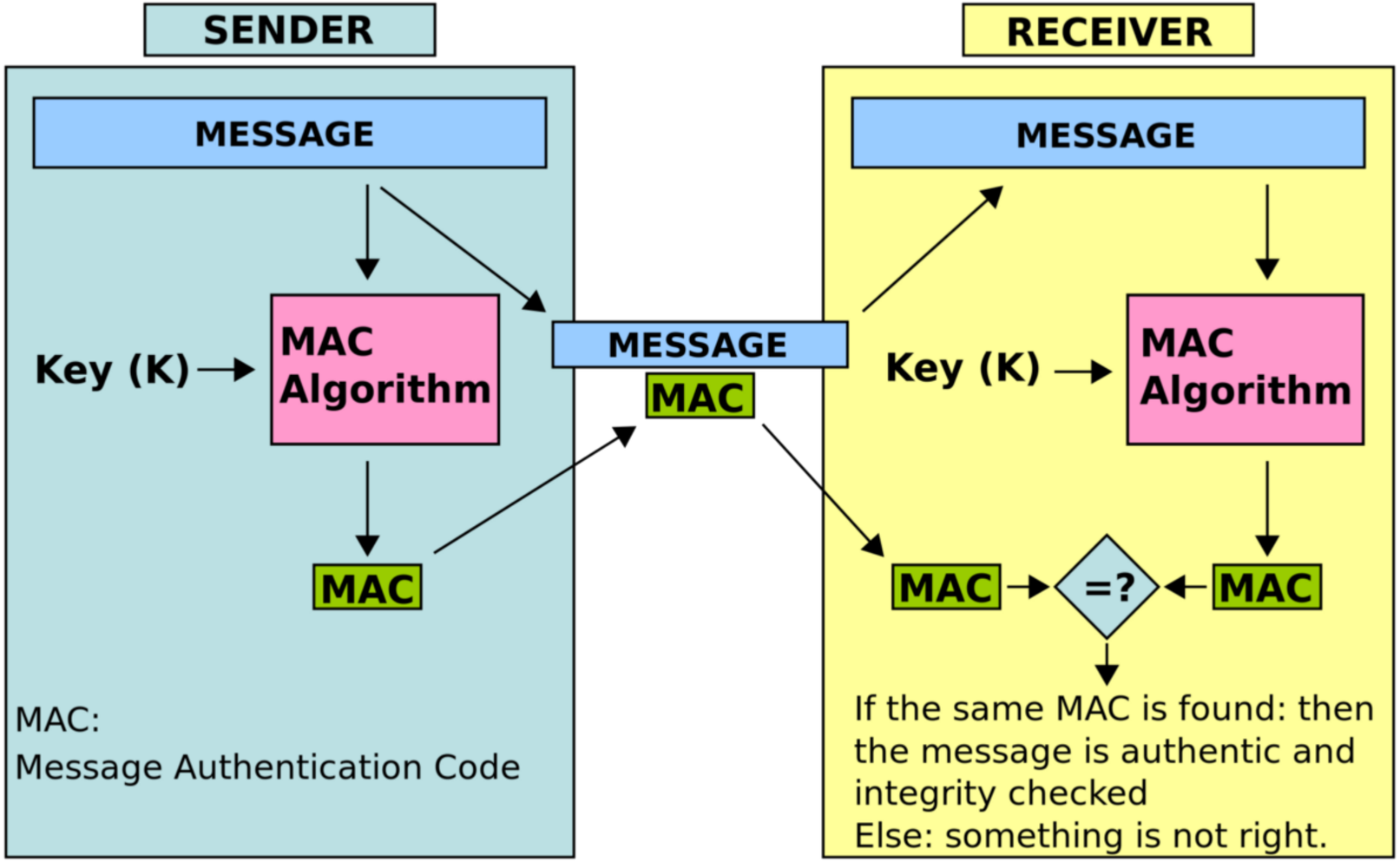
“Hi, Alice!”

HASH

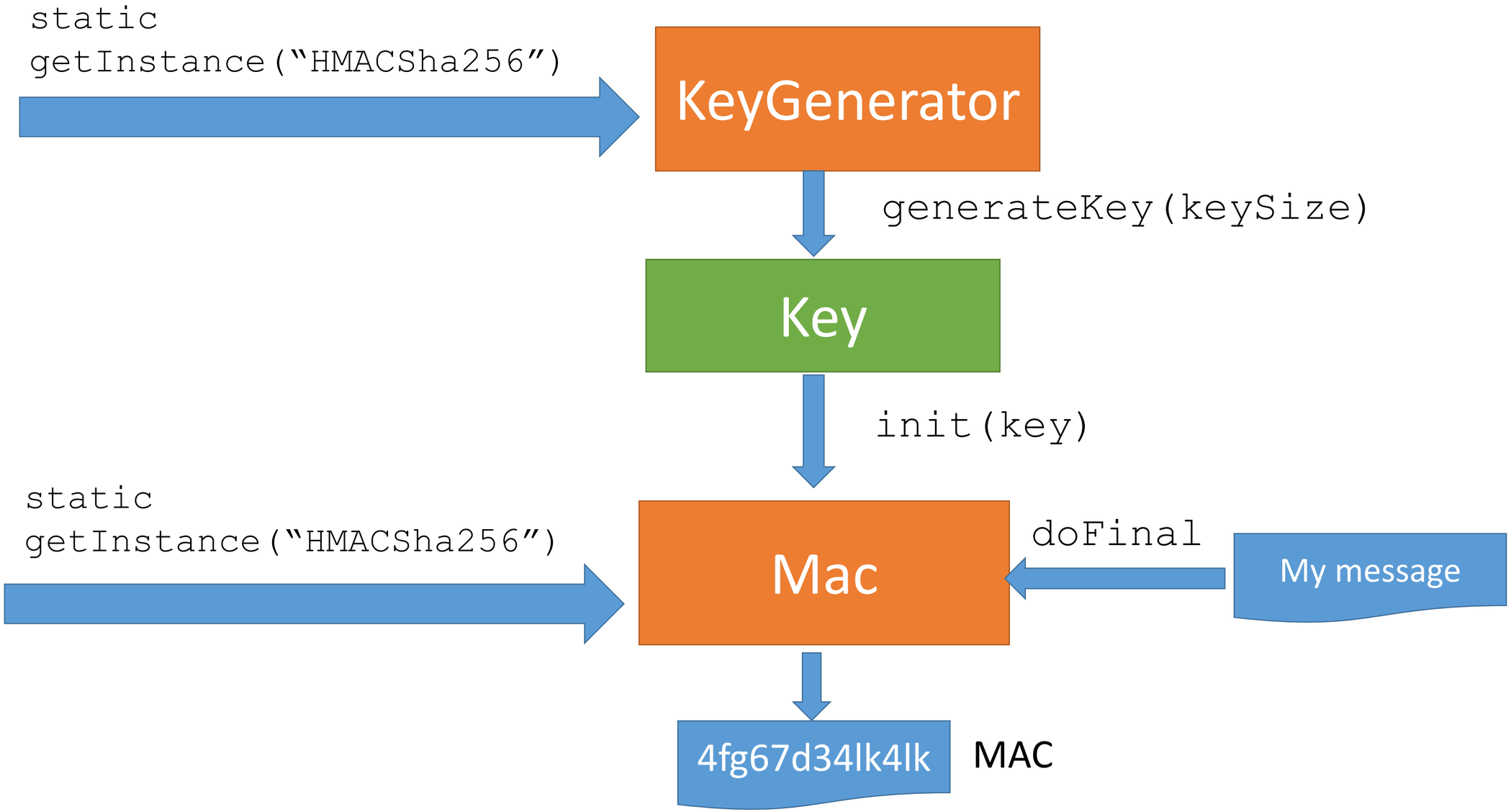
Calculate

HASH

HMAC - keyed-hash message authentication code



HMAC in Java


static
getInstance(“HMACSha256”)
KeyGenerator
generateKey(keySize)
Key
init(key)
static
getInstance(“HMACSha256”)
Mac
doFinal
My message
4fg67d34lk4lk
MAC

Digital signature using Asymmetric encryption

DATA

HASH

ENCRYPT HASH USING SIGNER'S PRIVATE KEY

PUBLIC KEY (CERTIFICATE)

DIGITALLY SIGNED

VALIDATE PUBLIC KEY

SIGNATURE

DECRYPT USING SIGNER'S PUBLIC KEY

HASH

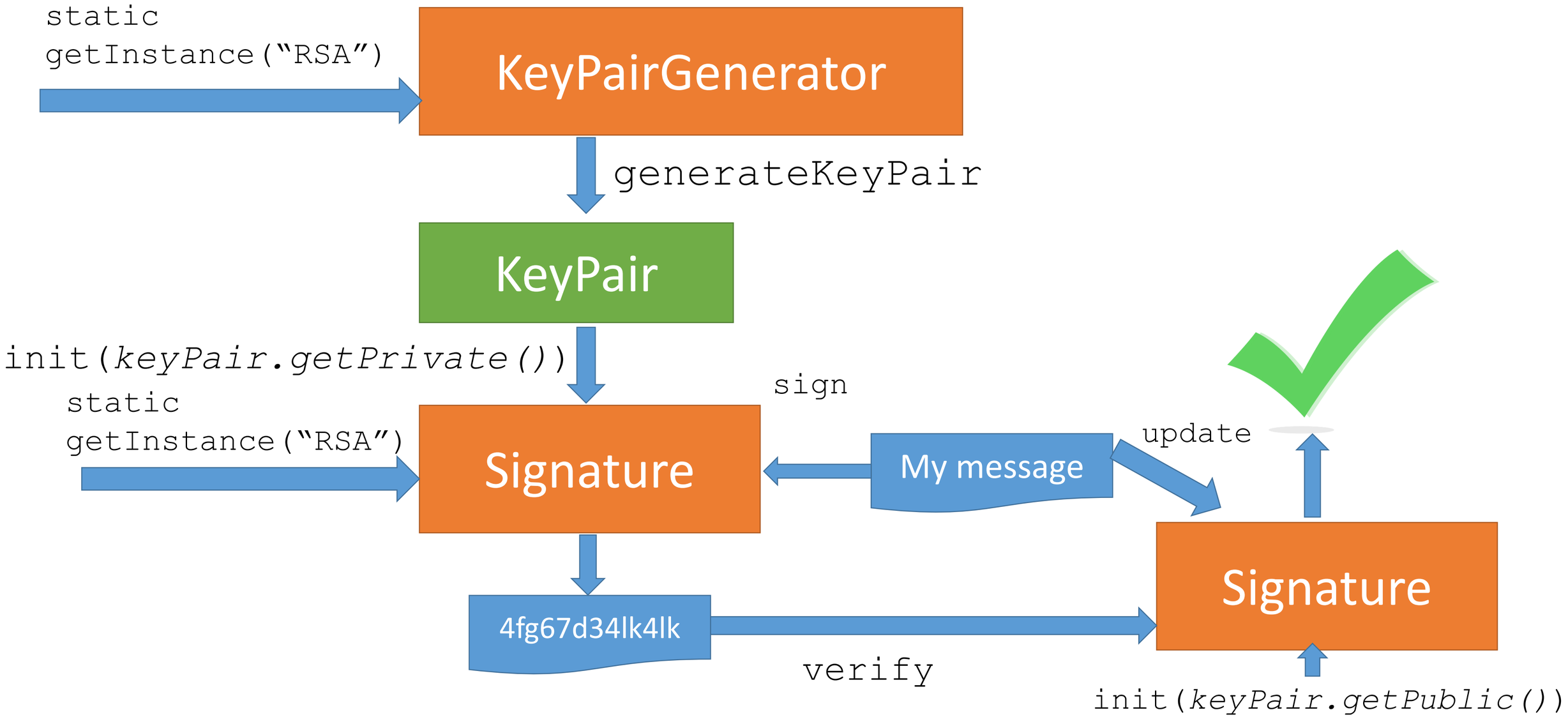
DATA

HASH

HASH

IF THE HASHES ARE EQUAL, THE SIGNATURE IS VALID

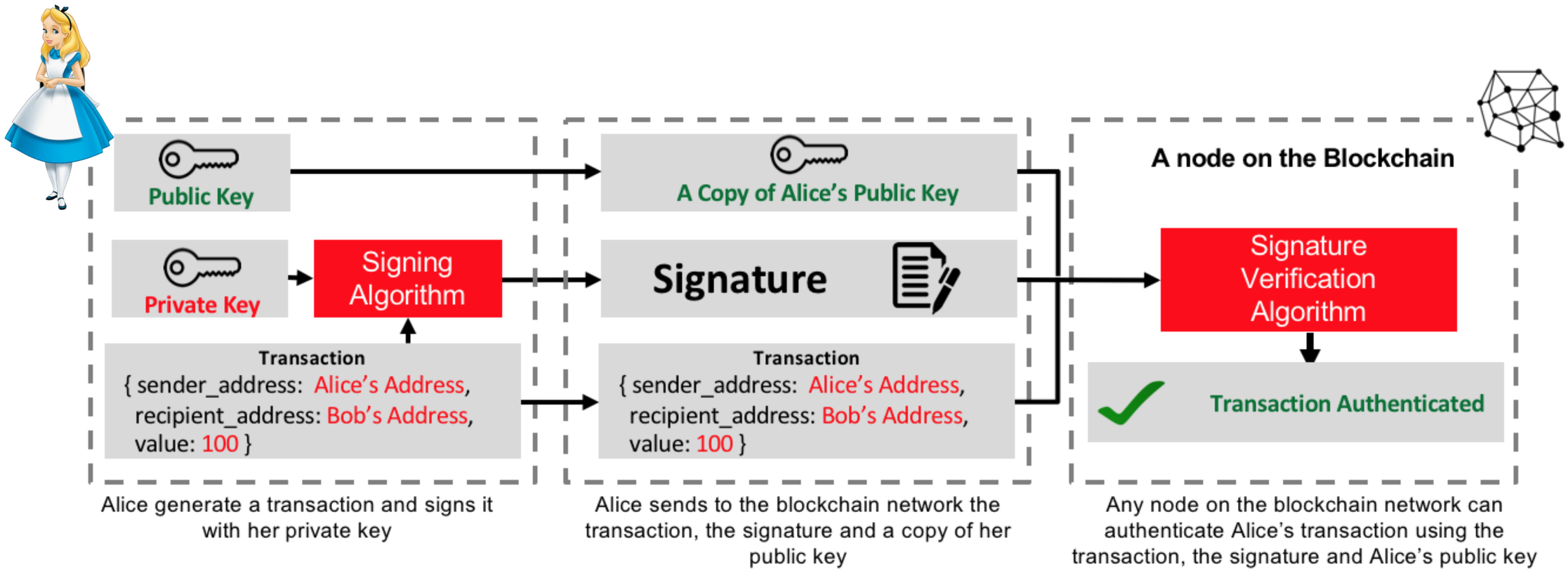
Digital signatures in Java



DEMO

Digital signature

Application: Signing crypto transactions



1. Hashing ✓
2. Symmetric encryption ✓
3. Asymmetric encryption ✓
4. Digital signatures ✓
5. Certificates 🔍

Confidentiality

Integrity

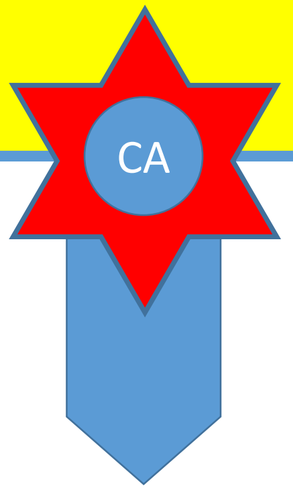
Authenticity

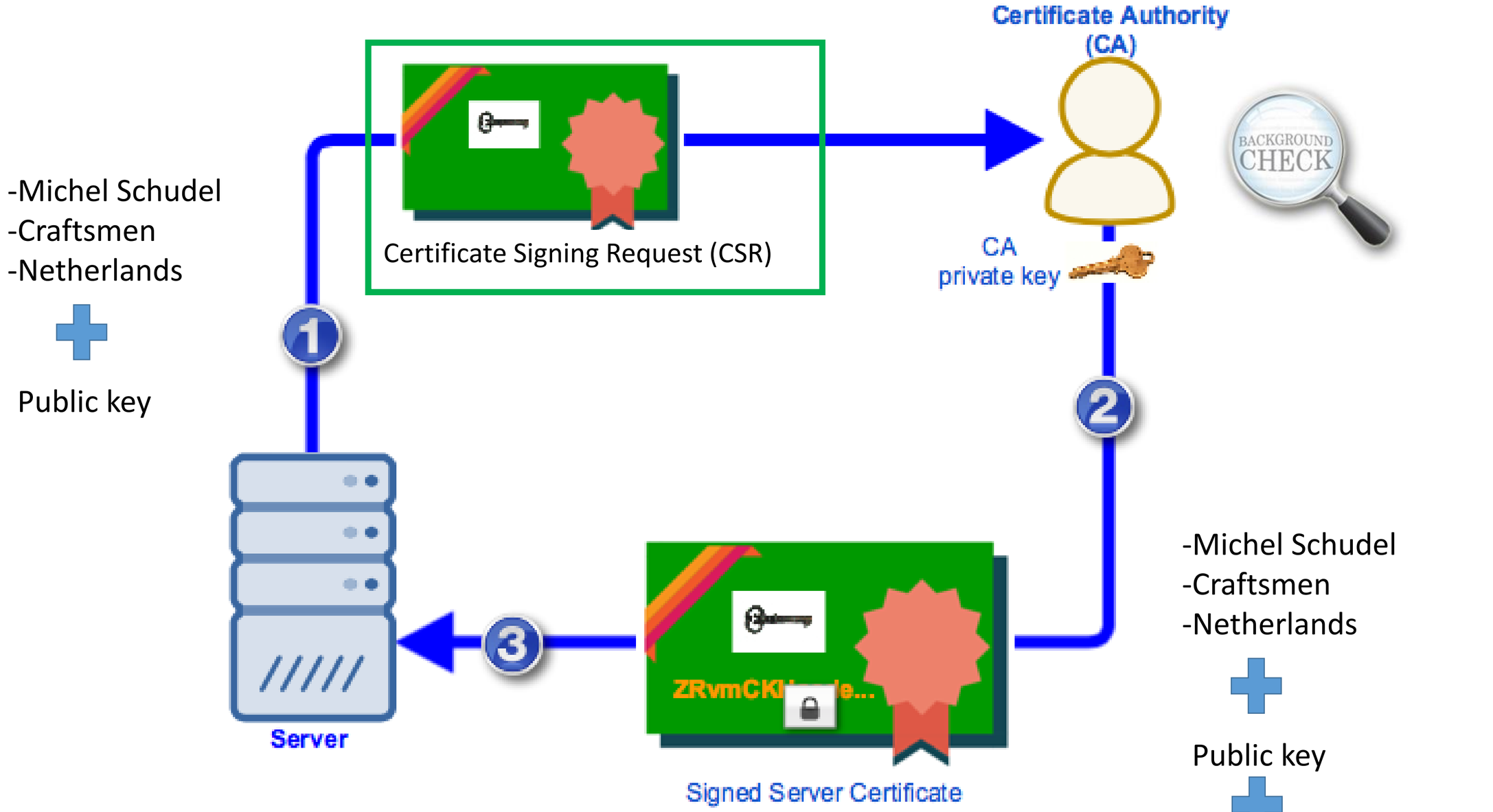
I , Certificate Authority **DigiCert**, hereby state that:

Michel Schudel
Craftsmen
Utrecht
Netherlands

Is who he/she claims to be, and that his/her *public key* is:
34a78b94858fd6c34a87f38c3a53cb85

digital signature of **DigiCert**
4843933df567dc





https://www.jscape.com/hubfs/images/csr-ca-signed-server-certificate.png

What is in a certificate signing request?

- **Common Name (CN):** The fully qualified domain name (FE *.mydomain.com)
- **Organization** (FE myCompany)
- **Organizational Unit** (FE IT)
- **City** (FE “Oslo”)
- **State /Country/ Region** (“Norway”)
- **Email address**
- **Public key**

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIC3TCCAcUCAQAwaDELMAkGA1UEBhMCTkwxCzAJBgNVBAgTAk5MMRMwEQYDVQQH
...
-----END NEW CERTIFICATE REQUEST-----
```

I , Certificate Authority **DigiCert**, hereby state that:

Michel Schudel

Is who he/she claims to be, *public key* is:
34a78b94858fd6c34a87f38c3a53cb85
digital signature of **DigiCert:** 4843933df567dc

Signed by

I , Certificate Authority **MasterCert**, hereby state that:

DigiCert

Is who he/she claims to be, *public key* is:
8969cd4385acd3efg69483d6c3
digital signature of **DigiCert:** 836cb84cacb34

Signed by

Root CA

I , Certificate Authority **MasterCert**, hereby state that:

MasterCert

Is who he/she claims to be, *public key* is:
56cd4b6f86a48d5810c384a12
digital signature of **MasterCert :** c5e473d1a8774

Self-Signed Certificate

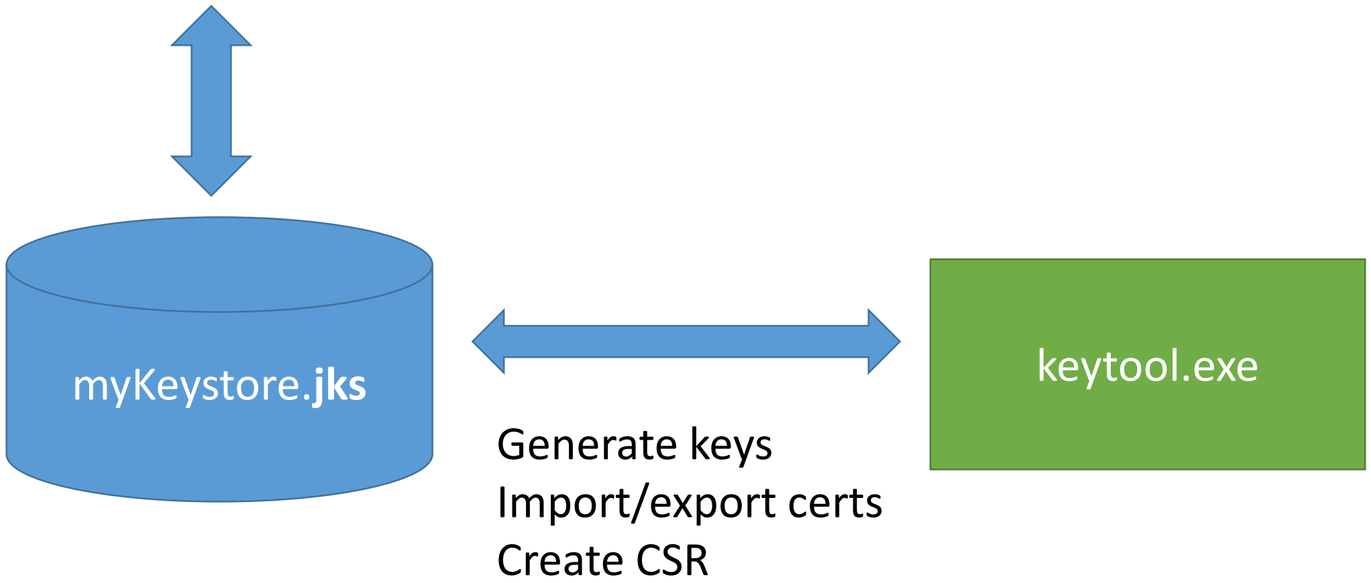

DigiNotar

Storing key and certificate material

JCA Keystore

```
Keystore keyStore = keyStore.getInstance("JKS");
keyStore.load(inputstream, keystorepassword);
Key key = keyStore.getKey(alias, keypassword);
Certificate certificate = keyStore.getCertificate(alias);
```

*.JKS
*.JCEKS
*.PKCS12



Keytool.exe

- Generate keypair

```
keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password -validity 360 -keysize 2048
```

- List contents

```
keytool -list -keystore keystore.jks -storepass password
```

- Generate certificate request

```
keytool -certreq -alias selfsigned -keystore keystore.jks -storepass password
```

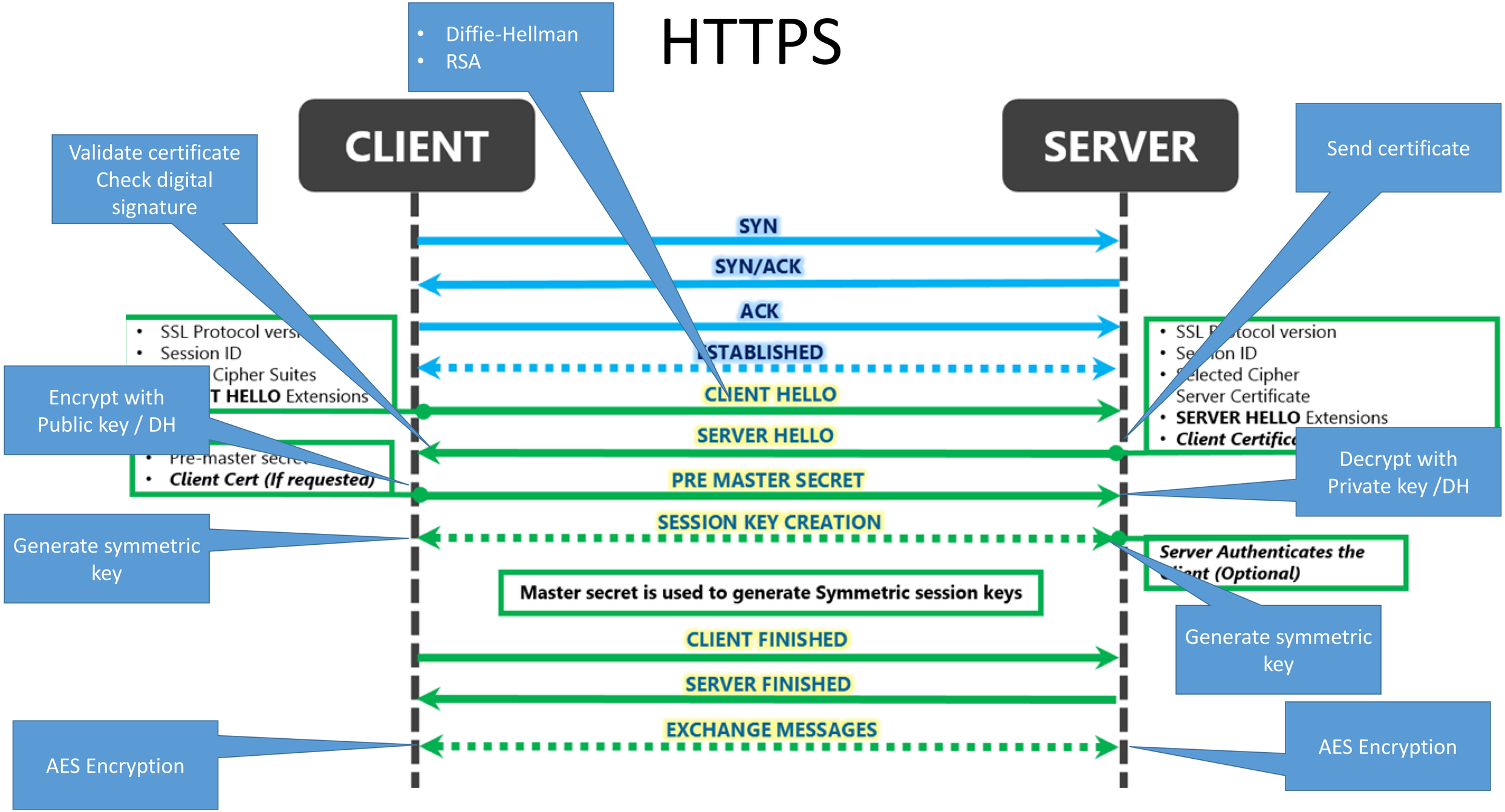
- Export certificate

- `keytool -exportcert -keystore keystore.jks -storepass password -alias selfsigned > out.cer`

1. Hashing ✓
2. Symmetric encryption ✓
3. Asymmetric encryption ✓
4. Digital signatures ✓
5. Certificates ✓

ALL TOGETHER NOW

HTTPS



https://www.cryptologie.net/article/340/tls-pre-master-secrets-and-master-secrets/

Basic understanding of cryptographic concepts and how they’re done in

Java


Q
&
A

https://github.com/MichelSchudel/crypto-demo

www.craftsmen.nl
michel@craftsmen.nl

@MichelSchudel

CRAFTSMEN
SOFTWARE ENGINEERS

Demo slides

(in case you didn’t see the demos)

Obtain a key

```
KeyGenerator generator = KeyGenerator.getInstance("AES", "BC");
generator.init(192);
Key key = generator.generateKey();
```

Init the algorithm with the key

```
Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding", "BC");
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Encrypt the data

```
byte[] encryptedOutput = cipher.doFinal(“Hello, devoxx!!!”.getBytes());
```

CE 41 54 A4 0F E0 2B 0C 7F C7 4B 2F 6E B5 B4 7A CE 41 54 A4 0F E0
2B 0C 7F C7 4B 2F 6E B5 B4 7A CE 41 54 A4 0F E0 2B 0C 7F C7 4B 2F 6E B5 B4 7A

Decrypting

```
cipher.init(Cipher.DECRYPT_MODE, key);
cipher.doFinal(CE 41 54 A4 0F E0 2B 0C 7F C7 4B 2F 6E B5 B4 7A CE 4
2B 0C 7F C7 4B 2F 6E B5 B4 7A CE 41 54 A4 0F E0 2B 0C 7F C7 4B 2F 6E B5 B4
```

```
hello, devoxx
```

Obtain a key

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(2048);
KeyPair kp = kpg.generateKeyPair();
Key pub = kp.getPublic();
Key pvt = kp.getPrivate();
```

Encryption

```
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, privateKey);
return cipher.doFinal(message.getBytes());
```

Creating a digital signature of a payload

```
Signature dsa = Signature.getInstance("SHA256withRSA");
dsa.initSign(keyPair.getPrivate());
dsa.update(“Hi devoxx!!!!”.getBytes());
byte[] signature = dsa.sign();
```

Verifying a signature

```
Signature dsa = Signature.getInstance("SHA256withRSA ");
dsa.initVerify(kp.getPublic());
dsa.update(“Hi devoxx!!!!”.getBytes());
boolean signatureIsOk = dsa.verify(signature);
```

Symmetric signing (HMAC)

```
KeyGenerator generator = KeyGenerator.getInstance("HMACSha256");
Key key = generator.generateKey();

// create signature
Mac mac = Mac.getInstance("HMACSha256");
mac.init(key);
byte[] input = "Hello, world!".getBytes();
byte[] signature = mac.doFinal(input);

// validation of signature
byte[] recievedInput = "Hello, world! ".getBytes();
byte[] newSignature = mac.doFinal(recievedInput);

// now compare newly generated signature with received signature
assertEquals(new String(signature), new String(newSignature));
```

Obtain a hash function

```
MessageDigest digester = MessageDigest.getInstance("SHA-256“);
```

Put some data in the function

```
digester.update(“Hi there”.getBytes());
```

Digest the data

```
digester.digest();
```

68 B1 28 2B 91 DE 2C 05 4C 36 62 9C B8 DD 44 7F 12 F0 96 D3 E3 C5 87 97 8D C2 24 84 44 63 34 83