

# Mobile Application Development Sign Your App

Waterford Institute of Technology

November 17, 2016

John Fitzgerald

# Sign your app

## Learning objectives

An overview of:

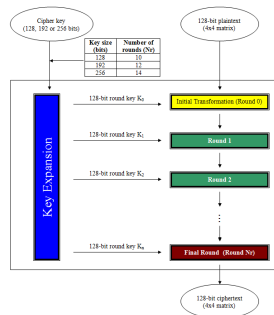
- Symmetric key encryption
- Public key encryption
- Cryptographic hash function
- Exchange secret key in public channel
- Review programming module crypto
- Certificates
- Signing app
- Key storage and security
- Secure shell (SSH)

# Sign your app

## Learning objectives

**Abstraction:** focus on details appropriate target audience

- High level: `ssh mike@192.168.61.8`
- Intermediate:  $c = m \oplus k$
- Low:  $D(k, c') = m_1 \oplus 1$



# Sign your App

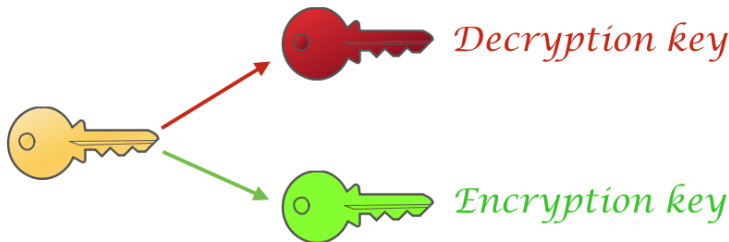
Android Studio

*Android Studio APK Signing*

# Sign your App

## Public Key Encryption

- Two related keys used.
- A private (secret) key (SK) used to decrypt.
- A public key (PK) used to encrypt.
- Keys have inverse functionality.
  - Encrypt with PK  $\Rightarrow$  decrypt with SK.
  - Sign (encrypt) with SK  $\Rightarrow$  verify (decrypt) with PK.



# Sign your app

## Certificates and Keystores

### Public-key certificate

- Also known as:
  - Digital certificate
  - Identity certificate
- Comprises:
  - Public key
  - Meta data
- Certificate owner:
  - Uses private (secret) key

# Sign your app

## Certificates and Keystores

- Android studio includes signing tool.
- Configurable auto or manual.
- App may also be signed using commandline tools.
- Attaches digital certificate to APK.
- Certificate acts as digital fingerprint or signature.
- Uniquely associates APK to author and its private key.
- Verifies future app updates authentic.
- Same certificate must be used during entire app life.

# Sign your app

## Digital Signature Scheme

Digital Signature Scheme comprises 3 algorithms:

- Public-private key-pair generator.
- Signing algorithm:
  - Input: message + private key.
  - Output: signature.
- Signature verifying algorithm:
  - Input: message + public key + signature.
  - Output: message authentic? Yes:No.



# Sign your app

## Digital Signature Scheme

Android implementation (v1):

- Up to and including Marshmallow.
- Uses standard Java Development Kit (JDK) tools:
  - *jarsigner* : signs message.
  - *jarsigner* : verifies authenticity of message.

# Sign your app

## Digital Signature Scheme

Android implementation (v2):

- Applies to Nougat (7.0).
- New app signing scheme.
- Recommended but not mandatory.
  - APK hashed and signed.
  - Resulting *APK Signing Block* inserted in APK.
  - Backward compatible.

# Sign your app

## Run and build from Android Studio IDE

- Uses **debug** version apk.
- Auto signs apk with debug certificate.
- Debug cert stored in debug keystore.
- All signing data auto generated.
- Debug unacceptable Google Play Store.

Run 'app' (^R)



Debug 'app' (^D)

# Sign your app

## Certificates and Keystores - Release build

- Android Studio generates keystore.
- On signing, use keystore and private key.
- Individually password protect store and private keys.
- Consider using password manager.
- Loss of passwords or keys potentially catastrophic.

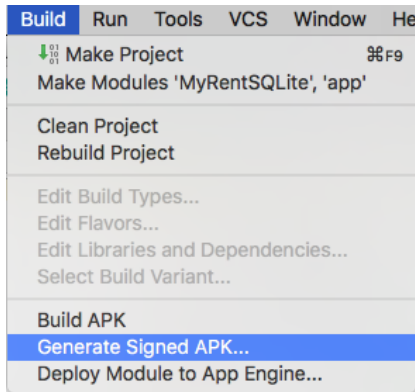
# Sign your app

## Certificate usage

- Sign all your APKs with same cert.
  - Throughout entire app lifespan.
- Facilitates upgrades.
  - Avoids loss installed client base
- Takes advantage of signature-based permissions policy.
  - Apps can share code and data securely.
- Facilitates modularization.
  - Multiple apps runnable as one in same process.

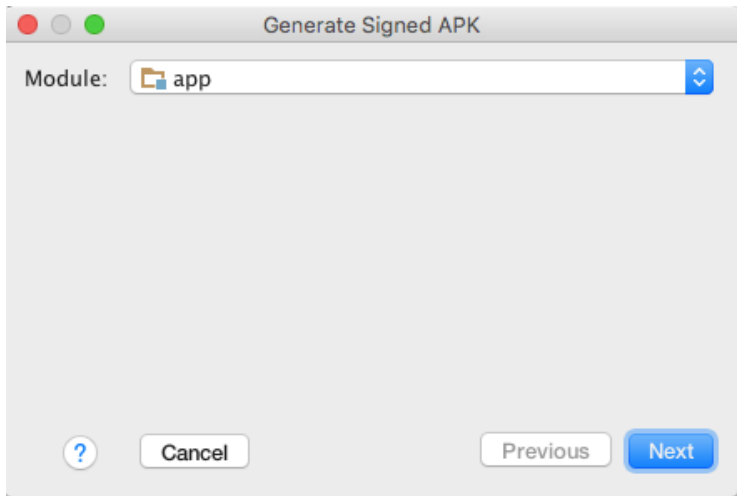
# Sign your app

Manually using Android Studio



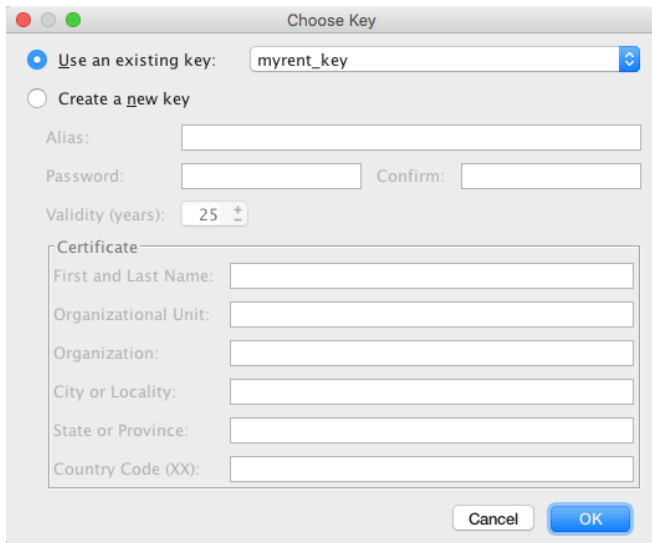
# Sign your app

Manually using Android Studio



# Sign your app

## Manually using Android Studio



The screenshot shows the 'Choose Key' dialog box in Android Studio. It has a title bar with standard macOS window controls (red, yellow, green buttons) and the title 'Choose Key'. The dialog contains two radio buttons at the top: 'Use an existing key:' (selected) and 'Create a new key'. The 'Use an existing key:' option has a text field next to it containing 'myrent\_key' and a dropdown arrow. Below these are fields for 'Alias:', 'Password:', and 'Confirm:'. The 'Validity (years):' field is set to '25' with a small '+' and '-' button. A 'Certificate' section is expanded, showing fields for 'First and Last Name:', 'Organizational Unit:', 'Organization:', 'City or Locality:', 'State or Province:', and 'Country Code (XX):'. At the bottom right are 'Cancel' and 'OK' buttons.

Choose Key

☒ Use an existing key: myrent\_key

☐ Create a new key

Alias:

Password:  Confirm:

Validity (years): 25

**Certificate**

First and Last Name:

Organizational Unit:

Organization:

City or Locality:

State or Province:

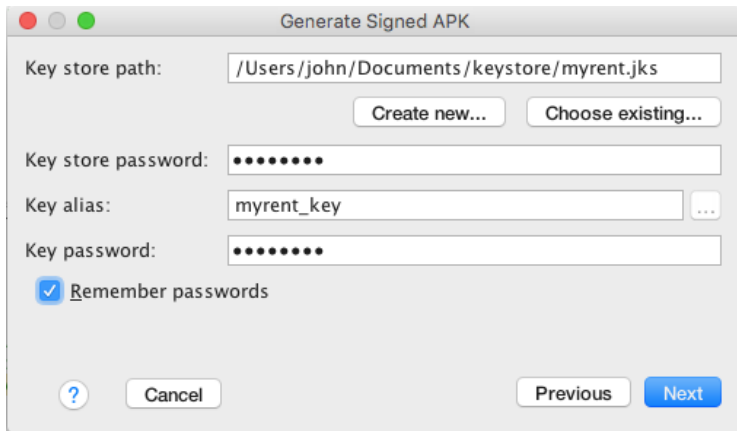
Country Code (XX):

Cancel OK



# Sign your app

## Manually using Android Studio



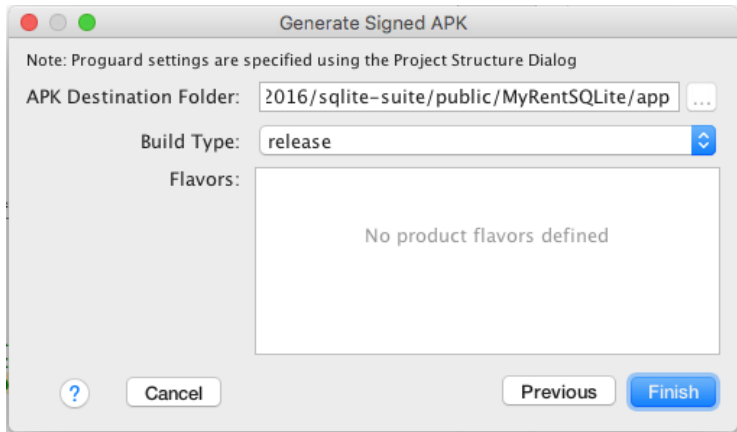
The screenshot shows the 'Generate Signed APK' dialog box in Android Studio. The dialog has a title bar with standard macOS window controls (red, yellow, green buttons). The main content area contains the following fields and controls:

- Key store path:** A text field containing the path `/Users/john/Documents/keystore/myrent.jks`. Below this field are two buttons: **Create new...** and **Choose existing...**.
- Key store password:** A text field filled with ten black dots, indicating a masked password.
- Key alias:** A text field containing the text `myrent_key`. To the right of the field is a small button with three dots.
- Key password:** A text field filled with ten black dots, indicating a masked password.
- Remember passwords:** A checkbox with a blue checkmark, followed by the text **Remember passwords**.

At the bottom of the dialog, there are four buttons: a help button (a circle with a question mark), a **Cancel** button, a **Previous** button, and a **Next** button.



# Sign your app








## Manually using Android Studio



# Sign your app

Manually using Android Studio

**Generate Signed APK**  
APK(s) generated successfully.  
[Reveal in Finder](#)

Name	^	Date Modified	Size	Kind
 app-release.apk		13:10	1.2 MB	Document
 app.iml		13:10	10 KB	Document
▶  build		Yesterday	--	Folder
 build.gradle		08/10/2016	562 bytes	Sublim...ument
▶  libs		08/10/2016	--	Folder
 proguard-rules.pro		08/10/2016	662 bytes	Document
▶  src		08/10/2016	--	Folder

# Key store and private keys

## Password management

- Passwords - critically important to:
  - Retain securely,
  - Retain indefinitely.
- Consider using password manager.
- Password Safe (Windows): <https://pwsafe.org/>
- Gorilla (Cross platform): <http://bit.ly/2elPsav>
- pwSafe (Mac & iOS): <https://pwsafe.info/>

# Sign your app

## Digital Signature

- Electronic analogue of physical signature
- Binds document & identity
- Not easily forged
- Various digital signature schemes:
  - Rivest, Shamir, Adleman (RSA)
  - Digital Signature Standard (DSS)

# Sign your app

## Digital Certificate

- Electronic document that can prove ownership.
- Pair of associated electronic keys used.
- Private key and public key.
- Signing tool attaches certificate to apk.

# Sign your app

## Digital Certificate

- Signed apk uniquely associated with signing author.
- Prevents forgery.
- Ensures any updates originate from signing author.

# Supporting cryptograpic technology

A brief exploration

## *Basics of Cryptographic Technology*



# Supporting cryptographic technology

## Three types cryptography

Single key used for both encryption and decryption.

### Symmetric key cryptography



# Supporting cryptographic technology

## Three types cryptography

Key pair: secret and public.

Public key cryptography (asymmetric)


plaintext (m)  ciphertext (c)  plaintext (m)

# Supporting cryptographic technology

## Three types cryptography

Public cryptographic hash function used. No key - plaintext not recoverable.

Hash function (one-way)

plaintext (m)  ciphertext (c)

# Supporting cryptographic technology

## Cryptographic hash function

- Uses include digital signatures, message authentication.
- Hash function maps any-size data to fixed-size data.
- Function output: hash values, codes, sums or hashes.
- Also input: message; output (message) digest.
- Collision-resistant: 2 inputs same output hard to find.
- Output does not leak input information.
- Output looks random.
- Small input change - large output change.

# Sign your app

## Public key cryptography

Cryptographic system that:

- Uses associated pair of keys - public & private.
- Public key may be distributed widely.
- Private key should be kept secure by owner.

# Sign your app

## Public key cryptography

Document encrypted using public key:

- Use private key to decrypt.

Document encrypted using private key:

- Use public key to decrypt.
- This is essence of digital signing.

# Supporting cryptographic technology

Encountered to date in programming module

- Caesar cipher
- Vigenere cipher
- One-time pad (OTP)

# Supporting cryptographic technology

Encountered to date in course

## Caesar cipher

- Message text or plain text
- Cipher text: encrypted plain text
- Encrypt: shift plain text character
- Example: shift by 3 thus A becomes D





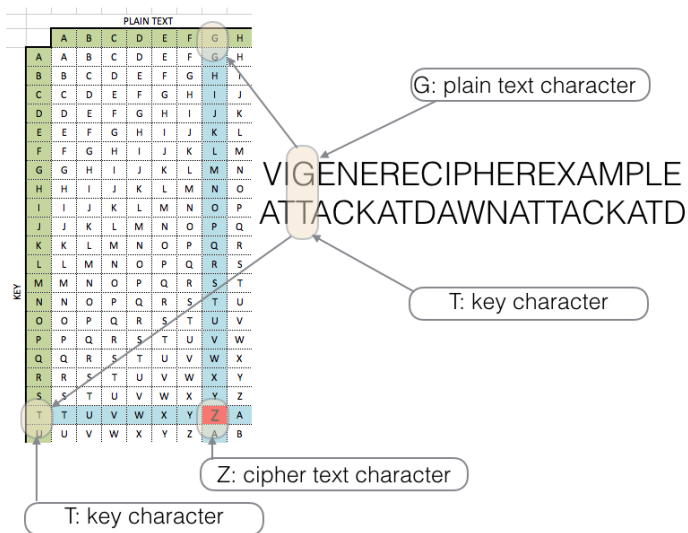
# Vigenere Cipher

Key length matches plain text

- Plain text
  - VIGENERECIPHEREXAMPLE
- Key same length plaintext
  - ATTACKATDAWNATTACKATD

PLAIN TEXT																									
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

## Encryption - Decryption



# Supporting cryptographic technology

Encountered to date in course

Potentially perfect secrecy - but practical difficulties.

- One-time pad (OTP)

# One Time Pad

Key same length as plaintext

Exclusive OR denoted by  $\oplus$ .

- $m$  denotes plaintext or message text
- $k$  denotes key
- $c$  denotes the cipher text or encrypted message
- $c = m \oplus k$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

m	0	1	1	0	1	1
k	1	0	1	1	0	0
c	1	1	0	1	1	1

# One Time Pad

Key same length as plaintext

Observe from table:

- $c = m \oplus k$
- $m = c \oplus k$

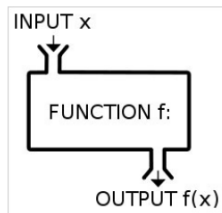
m	0	1	1	0	1	1
k	1	0	1	1	0	0
c	1	1	0	1	1	1
$c \oplus k$	0	1	1	0	1	1

# Hashing

What are hashes & how are they generated?

- What is a hash?
  - A fixed-length string.
  - The output from a function.
  - Known as hash function.
  - Whose input is a string of any length.

**x**: variable-length string

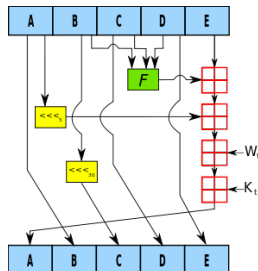


**f(x)**: fixed-length string

# Hashing example

Git versioning system uses SHA-1 hash function.

- Git uses SHA-1 hash function.
  - Purpose: ensure consistency.
  - Input: any number of bytes.
  - Output: 20-bytes.



# SHA-1 hashing examples

Observe differences between inputs and outputs

ICTSkills-2015

c83007996185ec1269ae9d1e78ef12d51ac0b078

ICTSkills-2016

33f87c1b7e03bc33b34e62313a638123260ca0b0



# Hash algorithm

## The internals of a hash function

- Hash algorithm
  - Algorithm: series of computations.
  - Producing solution to problem.
  - Hash algorithm: the internals of hash function.

# Hashes

What are they used for?

- Hashes are used:
  - To ensure data & message integrity.
  - To validate passwords.
  - In signing Android APKs.

# Hashing

## Hash function properties

- One-way functions.
  - Easy to compute output given an input.
  - Difficult to compute input given output.
- Small input variation.
- Result: large output variation.

# Creating shared secret key

Diffie-Hellman key exchange

*Diffie-Hellman*

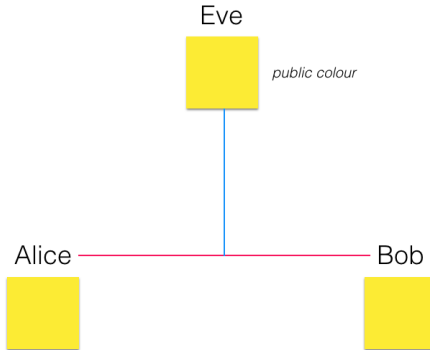
# Creating shared secret key

## Diffie-Hellman key exchange

- Securely exchange cryptographic keys over public channel
- PK crypto envisaged by James Ellis & mathematically proven by Clifford Cocks in GCHQ (1973).
- Malcolm Williamson in attempting to disprove PK discovered secure key exchange (1973).
- Immediately classified but made public in 1997.
- Independently discovered by Whitfield Diffie & Martin Hellman (1976).

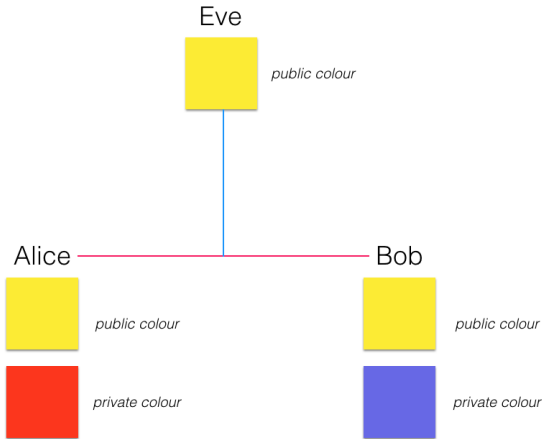
# Key exchange explained using colours

A random colour published



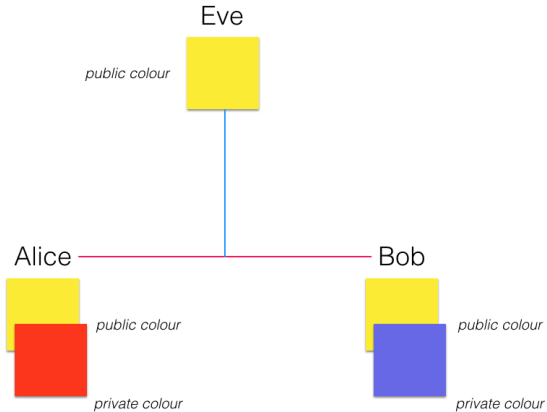
# key exchange

Alice & Bob each randomly select a secret colour



# Key exchange

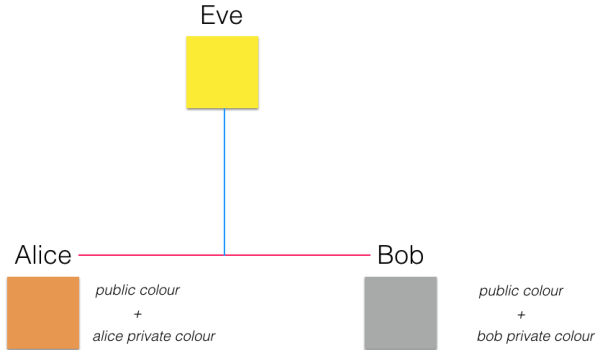
Alice & Bob mix public colour and secret colour - this is **easy**





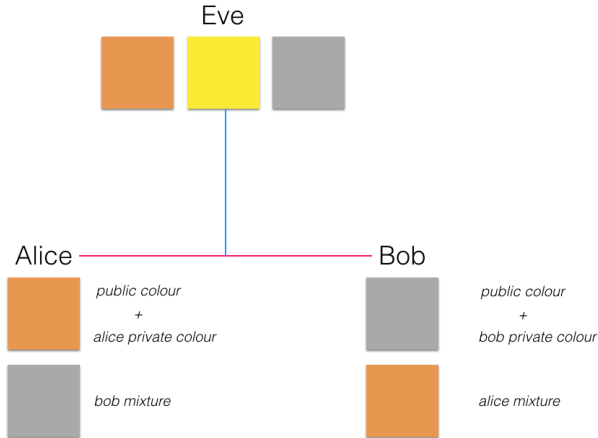
# Key exchange

Alice's & Bob's mixed colours - finding original colours is **hard**



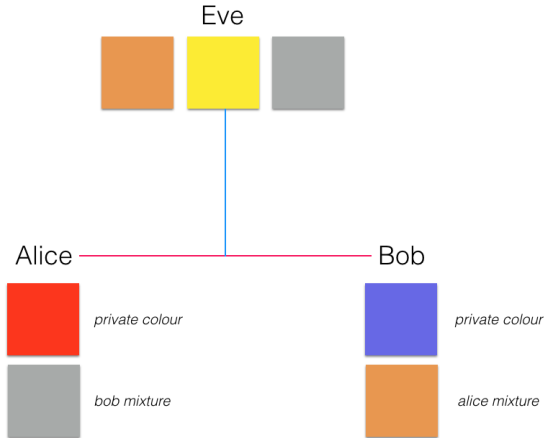
# Key exchange

Alice sends Bob her mixed colour - Bob sends Alice his mixed colour



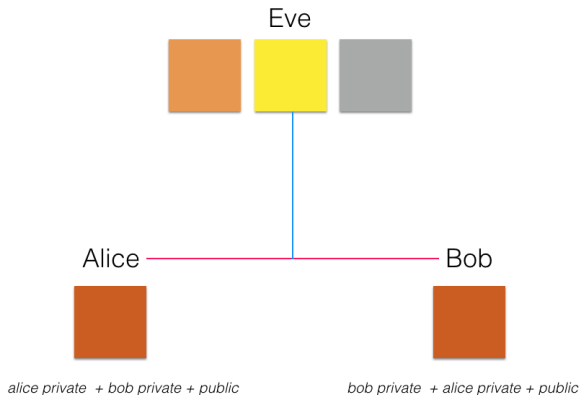
# Key exchange

Alice & Bob each add private colour to mixed colors



# Key exchange

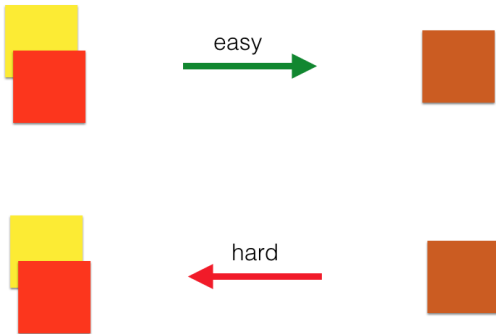
The two final mixtures are exactly the same colour - this is shared secret key



Alice's & Bob's shared secret key

# Key Exchange

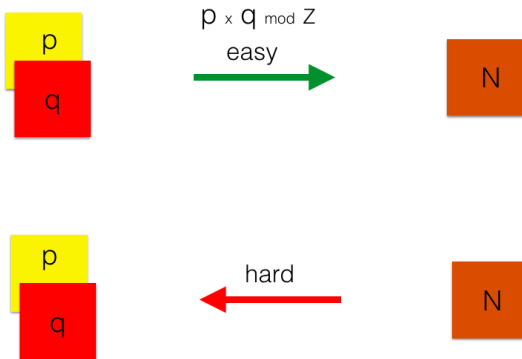
Uses One-Way Function



One-Way function

# Key Exchange

Uses One-Way Function



One-Way function

# Public Key Cryptography

Public-private key pair

*RSA*

# RSA Encryption

public-private key pair

Alice:

- Creates lock & key
- Key is private.
- Kept securely.
- Lock is public.





# RSA Encryption

public-private key pair

Alice:

- Sends open lock to Bob.
- Could send same lock multiple people.



# RSA Encryption

public-private key pair

Bob:

- Locks message.
- Returns to Alice.



# RSA Encryption

public-private key pair

Alice:

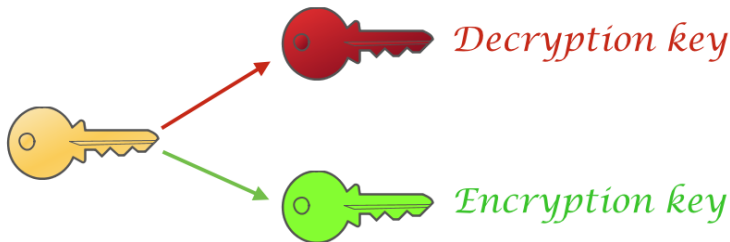
- Uses secret key.
- Unlocks Bob's message.
- Could unlock many messages.
- Secured with same lock.



# RSA Encryption

## Public Key Cryptography (PK)

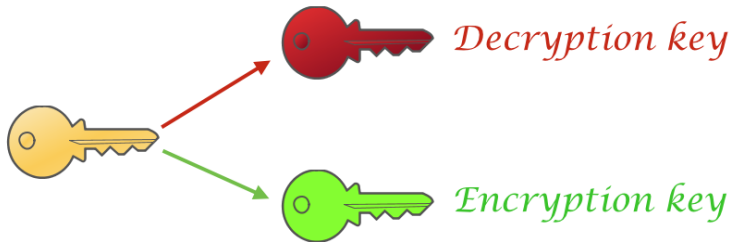
- PK crypto envisaged by James Ellis & mathematically proven by Clifford Cocks in GCHQ (1973).
- Ron **R**ivest, Adi **S**hamir & Leonard **A**dleman discovered independently (1977)



# RSA Encryption

## Public Key Cryptography (PK)

- Key generator produces two components.
- The private (secret) key (SK) used to decrypt.
- The public key (PK) used to encrypt.
- Keys have inverse functionality.
  - Encrypt with PK  $\Rightarrow$  decrypt with SK.
  - Sign (encrypt) with SK  $\Rightarrow$  verify (decrypt) with PK.



# Secure Shell (SSH)

Underlying cryptographic technologies

*Secure Shell (SSH)*

# Secure Shell (SSH)

Underlying cryptographic technologies

- Symmetric Encryption
- Public Key (Asymmetric) Encryption
- Hashing



# Secure Shell (SSH)

Underlying cryptographic technologies

Goal is to achieve:

- Authentication
- Message encryption
- Message integrity



**SHHH!**  
**IT'S A**  
**SECRET.**

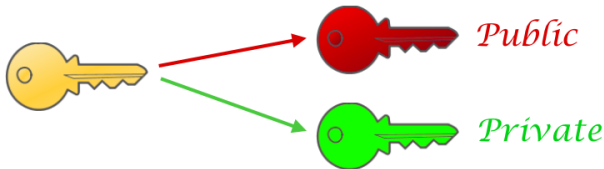


# Secure Shell (SSH)

## Underlying cryptographic technologies

### Asymmetric Encryption

- Client generates public-private key pair.
- Public key sent to server.
- Used in symmetric key set up.
- Used for authentication.

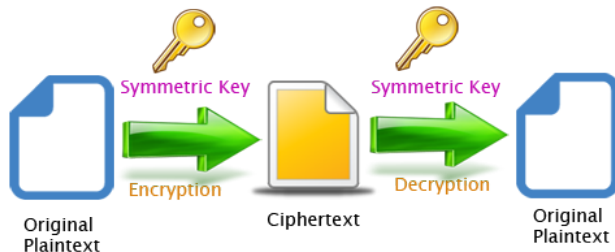


# Secure Shell (SSH)

## Underlying cryptographic technologies

### Symmetric Encryption

- Key exchange algorithm establishes shared secret key.
- This key used to encrypt data.

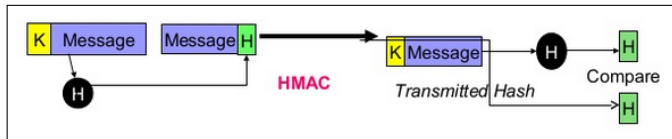


# Secure Shell (SSH)

## Underlying cryptographic technologies

### Hashing

- Hash-based message authentication code (HMAC).
- Used to ensure message integrity (not tampered).
- MAC signing algorithm generates tag using key and message.
- MAC verification algorithm uses key, message and tag.



# References

## Encryption & Digital Signing

1. Official documentation: Sign Your App

<http://bit.ly/2eIDwQE> [Accessed 2016-10-19]

2. Khan Academy: Journey into Cryptography

<http://bit.ly/2eIyBP0>

[Accessed 2016-10-27]

3. Mathematical Cryptosystems (1 of 2): Symmetric Cryptography

<http://bit.ly/2ey52Ti>

[Accessed 2016-10-27]

# References

## Encryption & Digital Signing

4. Mathematical Cryptosystems (2 of 2): Symmetric Cryptography

<http://bit.ly/2e0TpFV>

[Accessed 2016-10-27]

5. KhanAcademy: Digital Signatures High-level Description

<http://bit.ly/2eUCT5I>

[Accessed 2016-10-27]

6. Public Key Encryption & Digital Signature: How do they work?

<https://goo.gl/1HHsRo>

[Accessed 2016-10-28]

# References

## Encryption & Digital Signing

### 6. How PGP Works

<https://goo.gl/2UKnR5>

[Accessed 2016-10-28]

### 7. KhanAcademy: Modern Cryptography

<https://goo.gl/xXg4w9>

[Accessed 2016-11-06]

### 8. Android Development: Keytool, creating a keystore?

<https://goo.gl/ZzDto4>

[Accessed 2016-11-06]

# References

## Encryption & Digital Signing

9. Oracle SE Documentation: jarsigner

<https://goo.gl/YMSL4f>

[Accessed 2016-11-06]

10. Android APK Signature Scheme V2

<https://goo.gl/R8d4Bz>

[Accessed 2016-11-06]

11. Android Application Signing

<https://goo.gl/A95Tv9>

[Accessed 2016-11-06]

# References

## Encryption & Digital Signing

### 12. Understanding SSH

<https://goo.gl/CqwN4s>

[Accessed 2016-11-16]





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>



Waterford Institute of Technology  
INSTITIÚD TEICNEOLAÍOCHTA PHORT LÁIRGE

