Public key cryptology, RSA, ElGamal, Elliptic Curve

Jason Pearson and

November 29, 2015

Key Terms

Plain text: typically a simple text such as this line
Cipher text: a message after it has been encrypted

One-way Function: A function that is easy to compute on every input but hard to invert given the output

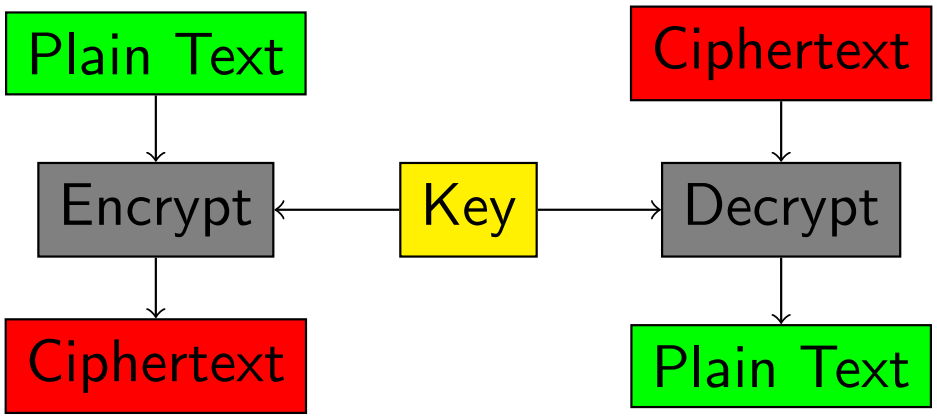
Trap Door One-way Function: A function that is easy to compute but hard to invert given the output unless you know the special information of the trap door

Types of Encryption

Symmetric Encryption
Asymmetric Encryption
Hybrid Encryption

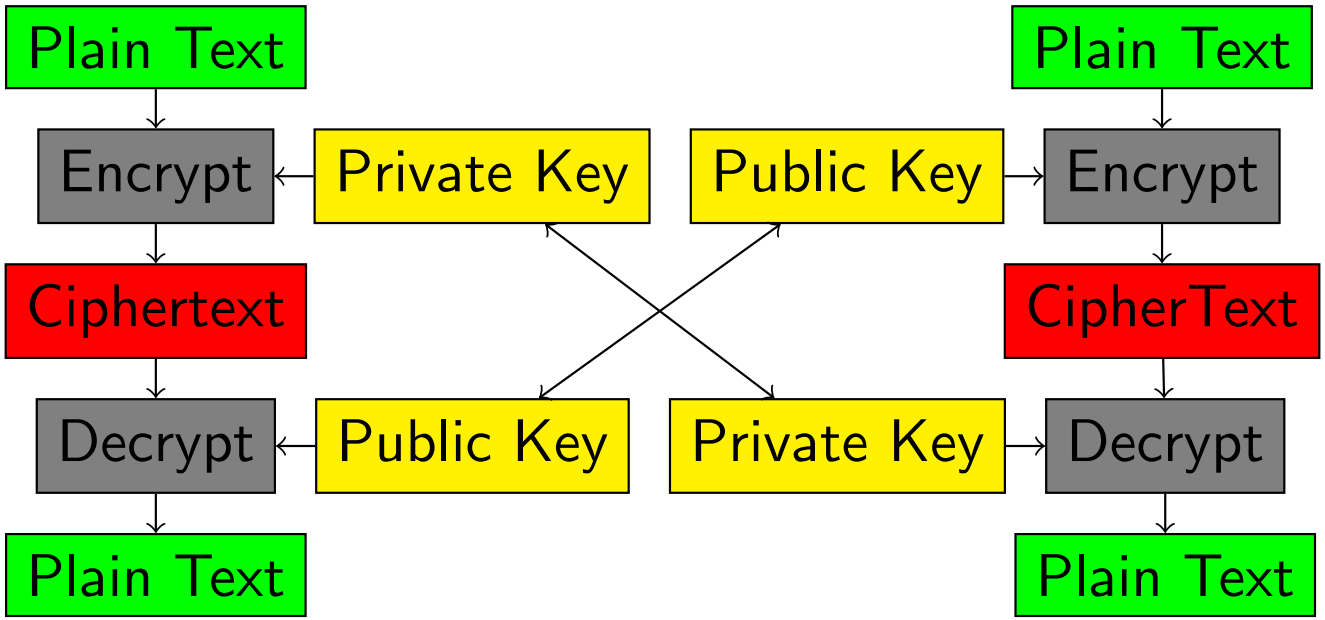
Symmetric Encryption

Encryption and Decryption use the same key
Symmetric systems include the Advanced Encryption Standard (AES), Blowfish and Serpent



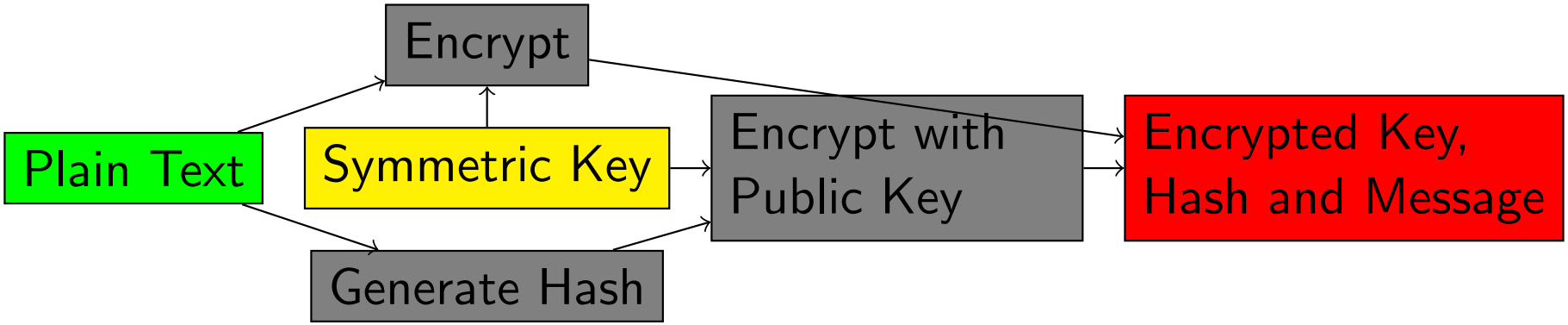
Asymmmetric Encryption

Public key and private key pair
Creating the key tends to be computationally expensive
Examples of asymmetric systems are ElGamal, RSA, DSS



Hybrid Encryption

Uses ideas from symmetric and asymmetric encryption methods
An asymmetric cryptosystem is used for key encapsulation and an symmetric system is used for data encapsulation
An example of a hybrid system is OpenPGP



RSA Cryptosystem

First designed in 1973 and declassified in 1997.
Named after its rediscoverers Ron Rivest, Adi Shamir and Leonar Adleman
Uses large prime numbers to create a private and public key
Security arises from the presumed difficulty of factoring large prime numbers

RSA Generating Public and Private Keys

Generate two large prime numbers and use to calculate n
Compute Euler's Totient Function
Create Public Key and Private Key

Generating Large Prime Numbers

Generate two large prime numbers.
Typically uses AKS testing and/or the Miller-Rabin test for prime numbers
These two values we will call p and q
p = 991
q = 821
We next calculate n = p * q
So we have n = 991 * 821 = 813611

Public Information	Secret Information
n = 813611	q = 821
	p = 991

Euler's Totient

The $\phi(n)$ value counts the number of positive numbers relatively prime to n
$\phi(7)$ would be 6 because 1,2,3,4,5,6 are all relatively prime to 7
We then can determine Euler's totient value by using the following equation.
$\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$
And for $\phi(n) = \phi(813611) = 811800$

Public Information	Secret Information
n = 813611	q = 821
	p = 991
	$\phi(n) = 811800$

Create Public and Private Keys

To create a public key we pick an arbitrary number e between $1 < e < \phi(n)$ for example we will use e = 7423

To create a private key we need to find the modular multiplicative inverse of e.

$d \equiv e^{-1} \pmod{(\phi(n))}$ or

$d * 7423 \equiv 1 \pmod{(\phi(n))}$

$788287 * 7423 \equiv 1 \pmod{(\phi(n))}$

$5851454401 \equiv 1 \pmod{(\phi(n))}$

$1 \equiv 1 \pmod{(\phi(n))}$

This is commonly done using the Extended Euclidean Algorithm.

This makes our value of d = 788287

Public Information	Secret Information
n = 813611	q = 821
e = 7423	p = 991
	$\phi(n) = 811800$
	d = 788287

Working Example

Using these values we can create a cipher text c and decrypt it using the following equations
$c \equiv m^e \, (mod \, (n))$ and $m \equiv c^d \, (mod \, (n))$
Finishing up our example we will encrypt "Hi" using our new values plain text m = 72105, e = 7423, d = 788287, n = 813611

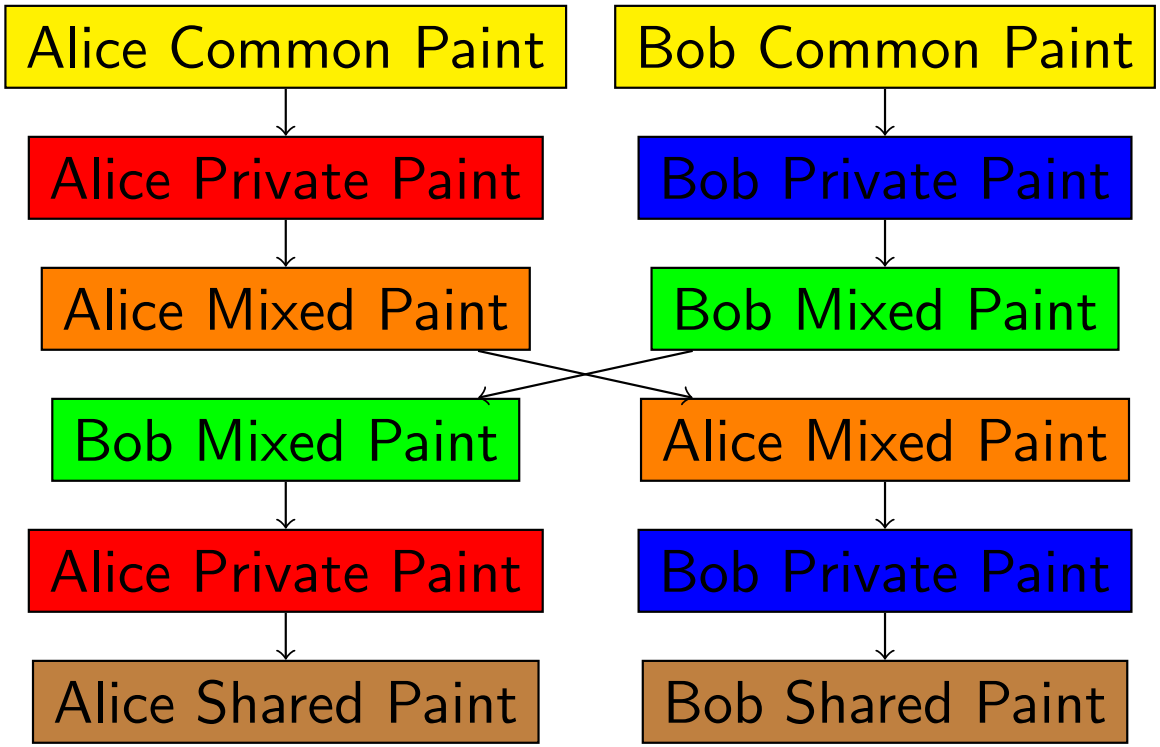
Public Encrypt	Private Encrypt
$c \equiv m^e \, (mod \, (n))$	$c \equiv m^d \, (mod \, (n))$
$c \equiv 72105^{7423} mod(813611)$	$c \equiv 72105^{788287} mod(813611)$
$c = 707473$	$c = 616895$
Private Decrypt	**Public Decrypt**
$m \equiv c^d \, (mod \, (n))$	$m \equiv c^e \, (mod \, (n))$
$m \equiv 707473^{788287} mod(813611)$	$m \equiv 616895^{7423} mod(813611)$
$m = 72105$	$m = 72105$

RSA Practical Considerations

Used in many secure applications escpecially on the internet
Keys need to be very large and continue to lengthen as computers become more powerful
A quantum computer can factor in polynomial time breaking RSA

Diffie-Hellman Key Exchange

This key exchange allows for secret communication over a public network.



ElGamal Cryptosystem

First described by Taher Elgamal in 1985
ElGamal can be represented over any cyclic group
This method for cryptogphy uses discrete logarithms with a large prime modulus

Generating Large Prime Numbers

First we generate a large prime number p
For us p = 17
We then create a generator g of multiplicative group $\mathbb{Z}_p^*$ of integers modulo p
A generator when raised to a power x is evenly distributed over $\mathbb{Z}_p^*$
For this example g = 6
So $< g >= \{6^1, 6^2, 6^3, 6^4, 6^5, 6^6, 6^7, 6^8, 6^9, 6^{10}, ...\}$
Or $\{6, 36, 216, 1296, 7776, 46656, 279936, 1679616, 10077696, ...\}$
Thus $\{6, 2, 12, 4, 7, 8, 14, 16, 11, 15, ...\}$

Public Information	Private Information
p = 17	
g = 6	

ElGamal Creating Public and Private Keys

We then select a private key a where $1 \leq a \leq p-2$
For this example a $= 5$
We can then use this to generate the last part of the public key
$g^a \mathrm{mod} p = 6^5 \mathrm{mod} 17 = 7$

Public Information	Private Information
p $= 17$	a $= 5$
g $= 6$	
$g^a \mathrm{mod} p = 7$	

Encrypting a Message

We will have our message m $= 13$
A public sender to send a message to the private key holder picks a random value k for this example k $= 10$
We then compute $c_1 = g^k mod p = 15$
Now $c_2 = m * g^k \mathrm{mod} p = 13 * 6^{10} mod 17 = 8$
Cipher text sent through c_1 and c_2 to private key holder

Public Information	Private Information
p $= 17$	a $= 5$
g $= 6$	
$g^a \mathrm{mod} p = 7$	

Decrypting a Message

First we must calculate the shared secret
$s = (c_1{}^a) * c_2 \mathrm{mod} p = \left(15^5\right) * 8 \mathrm{mod} 17 = 16$
We then take the modular inverse of s and multiply it by c_2
Finding the modular inverse is commonly done with the Extended Euclidean Algorithm
$m = \left(c_2 {}^* s^{-1}\right) \mathrm{mod} p = (8 * 8) \mathrm{mod} 17 = 64 \mathrm{mod} 17 = 13$

Public Information	Private Information
p = 17	a = 5
g = 6	
$g^a \mathrm{mod} p = 7$	

ElGamal Practical Usage

Cipher text double the size in bits than the message
Commonly used in hybrid Cryptosystems
El Gamal is probablistic meaning that the same message encrypted won't always give the same ciphertext