# Security

Biyong SUN

17, MAR 2017

# RSA

# RSA – What is RSA

RSA is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret.

RSA is made of the initial letters of the surnames of Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman, who first publicly described the algorithm in 1977

RSA is a **relatively slow algorithm**, and because of this it is less commonly used to directly encrypt user data. More often, RSA **passes encrypted shared keys for symmetric key** cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

# RSA - Key generation

Choose two distinct prime numbers p and $q$.

For security purposes, the integers $p$ and $q$ should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits'[2] to make factoring harder. Prime integers can be efficiently found using a primality test.

Compute $n = pq$.

$n$ is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the is the key length.

Compute $\lambda(n) = \text{lcm}(\lambda(p), \lambda(q)) = \text{lcm}(p - 1, q - 1)$, where $\lambda$ is Carmichael's totient. This value is kept private.

Choose an integer $e$ such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., $e$ and $\lambda(n)$ are coprime.

Determine $d$ as $d \equiv e^{-1} \pmod{\lambda(n)}$; i.e., $d$ is the modular multiplicative inverse of $e$ (modulo $\lambda(n)$).

# RSA - Key generation Example

1.Choose two distinct prime numbers
p = 7 and q = 17
2.Compute n = pq
n = 7 X 17 = 119
3.Compute the totient of the product as λ(n) = lcm(p − 1, q − 1)
λ ( 119 ) = lcm ( 6 , 16 ) = 48
4.Choose any number 1 < e < 48 that is coprime to 48. Choosing a prime number for e leaves us only to check that e is not a divisor of 48.
Let e = 5
5.Compute d
d = 29
5X29mod48=1

e = 11 d = 35
11X35mod48=1

Key pair: public key: n = 119 e = 5 private key: n = 119 d = 29
public key: n = 119 e = 11 private key: n = 119 d = 35

**Lcm - Least common multiple (最小公倍数)**

# RSA – encryption and decryption

The public key is (n = 119, e = 5). For a padded plaintext message m, the encryption function is:

c(m) = $m^5 \text{mod} 119$

The private key is (n = 119, d = 29 ). For an encrypted ciphertext c, the decryption function is:

m(c) = $c^{29} \text{mod} 119$

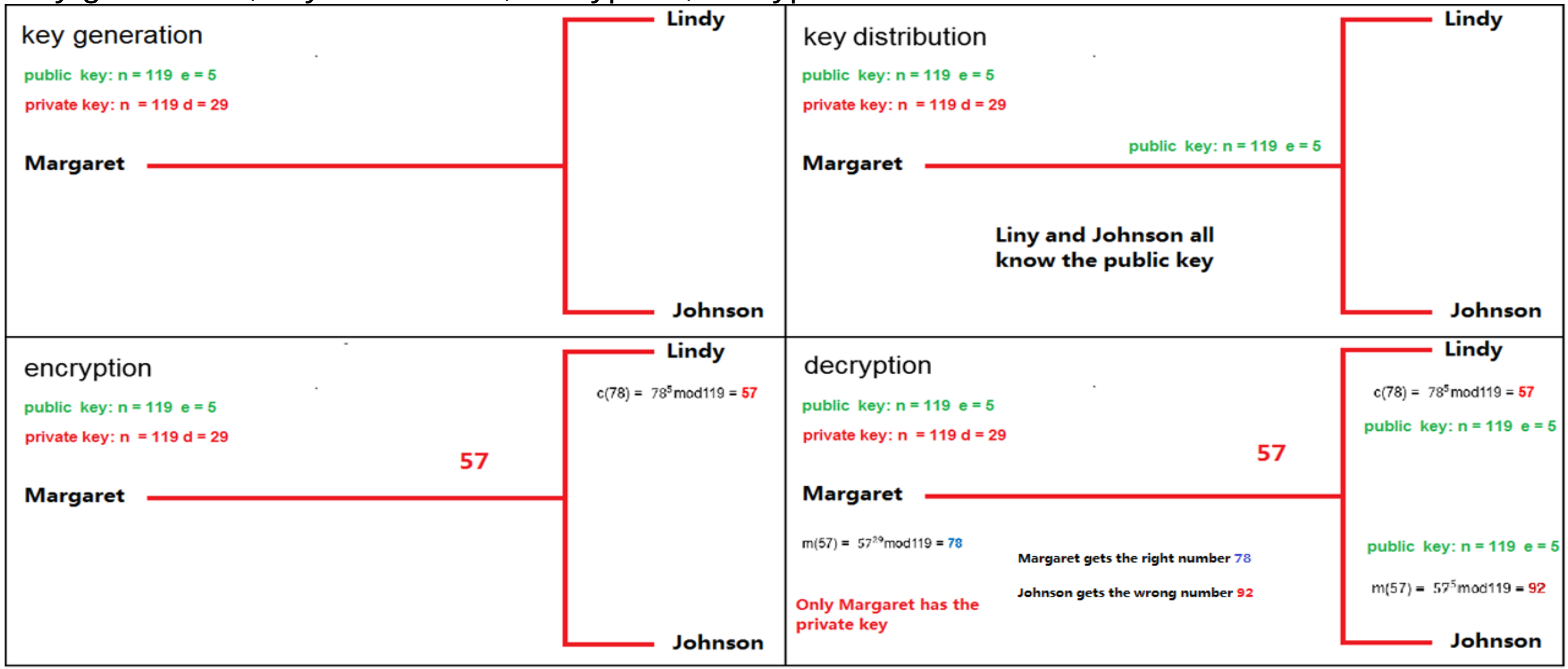
For instance, in order to encrypt m = **78**, we calculate
c(78) = $78^5 \text{mod} 119$ = **57**

To decrypt c = **57**, we calculate
m(57) = $57^{29} \text{mod} 119$ = **78**

Note: Public key n =119, e =5 and the c = 57 is on an **unsecure** data transmission channel.

# RSA – Operation

The RSA algorithm involves four steps:
key generation, key distribution, encryption, decryption



# RSA – Oversized

Any "oversized" private exponents not meeting that criterion may always be reduced modulo $\lambda(n)$ to obtain a smaller equivalent exponent.
some standards like Digital Signature Standard (DSS) FIPS 186-4 may require that d < λ(n).

Why:

Key pair: public key: n = 119 e = 5 private key: n = 119 d = 29

m = 119
c(119) = $119^5 \bmod 119 = 0$ **c = 0**

m = 118
c(118) = $118^5 \bmod 119 = 118$ **c = m = 118**

m = 200 > n = 119

c(200) = $200^5 \bmod 119 = 30$
m(30) = $30^{29} \bmod 119 =$ **81 ≠ 200**

# RSA problem – Integer factorization

The RSA problem is defined as the task of taking eth roots modulo a composite n: recovering a value m such that c ≡ me (mod n), where (n, e) is an RSA public key and c is an RSA ciphertext. Currently the most promising approach to solving the RSA problem is to factor the modulus n. With the ability to recover prime factors, an attacker can compute the secret exponent d from a public key (n, e), then decrypt c using the standard procedure.

Multiple polynomial quadratic sieve (MPQS) can be used to factor the public modulus *n*. The time taken to factor 128-bit and 256-bit *n* on a desktop computer (Processor: Intel Dual-Core i7-4500U 1.80GHz) are respectively 2 seconds and 35 minutes.

| Bits | Time |
|---|---|
| 128 | Less than 2 seconds |
| 192 | 16 seconds |
| 256 | 35 minutes |
| 260 | 1 hour |

# RSA problem – Integer factorization(Cont.)

In 2009, Benjamin Moody has factored an RSA-512 bit key in 73 days using only public software (GGNFS) and his desktop computer (dual-core Athlon64 at 1,900 MHz). Just under 5 gigabytes of disk was required and about 2.5 gigabytes of RAM for the sieving process. The first RSA-512 factorization in 1999 required the equivalent of 8,400 MIPS years over an elapsed time of about 7 months.

As of 2010, the largest factored RSA number was 768 bits long (232 decimal digits, see RSA-768). Its factorization, by a state-of-the-art distributed implementation, took around fifteen hundred CPU years (two years of real time, on many hundreds of computers). No larger RSA key is publicly known to have been factored. In practice, RSA keys are typically 1024 to 4096 bits long.

# RSA problem – Integer factorization(Cont.)

A tool called yafu can be used to optimize this process. The automation within YAFU is state-of-the-art, combining factorization algorithms in an intelligent and adaptive methodology that minimizes the time to find the factors of arbitrary input integers. Most algorithm implementations are multi-threaded, allowing YAFU to fully utilize multi- or many-core processors (including SNFS, GNFS, SIQS, and ECM).The time taken to factor *n* using yafu on the same computer was reduced to 103.1746 seconds. Yafu requires the GGNFS binaries to factor N that are 320 bits or larger. It took about 5720s to factor *320bit-N* on the same computer.

| Bits | Time | Memory used |
|---|---|---|
| 128 | 0.4886 seconds | 0.1 MiB |
| 192 | 3.9979 seconds | 0.5 MiB |
| 256 | 103.1746 seconds | 3 MiB |
| 300 | 1175.7826 seconds | 10.9 MiB |

# RSA – the key generating for this presentation

**Algorithm – Biyong's Muscle Computing**
**Using two calculators manually adjust the e and d till find the keys.**

**For Example: p = 7 and q = 17**

One calculator 1 for λ ( 119 ) = lcm ? ( 6 , 16 ) = 48
One calculator 2 for e = 5 d = 29

calculator 1 48 …… 96 ….. 144(48X3)
calculator 2 5 10 15 20 25 30 35 40 45 50 55 ……145(5X29)

Note: The p and q here is very small, when someone looks at the public key.
It is easy to factor and find the p and q.

# Real World – OpenSSL

cat Message.TXT
Hello RSA

openssl genrsa -out test_key_pair.key 1024
openssl rsa -in test_key_pair.key -pubout -out test_pub.key
openssl rsautl -encrypt -in Message.TXT -inkey test_pub.key -pubin -out Message.TXT.en

cat Message.TXT.en
[Ã¦Ã»kÃ¤Â¦Ã¾Ã»
<81>^OVÃµÃ§<8a>t^QÃµÃ¦Ã«Ã°Â£d{Ã“Â°Ã™#ÃŽ^UÃ™:Â½<93>&^U^PÃm{Â<85>ÃŸÃªÃSÂ¼
ÃœÃ“Ã«^Qp%H^\UÂºÃ«KÃ¡bÃŒÂ¾Ã•Gk<98>Ã‚xÃŒ^W^_<9a><95>Ã\
ÂµÃ¡]Ã¹Ã”JeÃÂ¬P^^TÃ¼Ã±Ã˜WÂ¦Ã¨^LÃœ
¯Ã™Ã·J<80>Ã¡Ãm<81>xÃ¥Âµ<93>F^jÂ¾^]{4<9f><91f><91>

openssl rsautl -decrypt -in Message.TXT.en -inkey test_key_pair.key -out Message.TXT.de

cat Message.TXT.de
Hello RSA

# Real World – OpenSSL(Cont.)

Using the NXP CryptoDev security driver causes the system to run much faster than without it.
The CAAM drivers are accelerated through the CryptoDev interface. The openssl command can be used to show the system speed without CryptoDev .
openssl speed -evp aes-128-cbc -engine cryptodev
An example of the key portion of the output is as follows. Library load errors may occur but they can be ignored.
Doing aes-128-cbc for 3s on 16 size blocks: 4177732 aes-128-cbc's in 2.99s
Doing aes-128-cbc for 3s on 64 size blocks: 1149097 aes-128-cbc's in 3.01s
Doing aes-128-cbc for 3s on 256 size blocks: 297714 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 75118 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks: 9414 aes-128-cbc's in 3.00s

Start CryptoDev and run the openssl command again. This time you should be able to see that the timeing values show the accelerated values. As the block sizes increase, the elapsed time decreases.
**modprobe cryptodev**
openssl speed -evp aes-128-cbc **-engine cryptodev**
Here is an example of the accelerated output:
Doing aes-128-cbc for 3s on 16 size blocks: 36915 aes-128-cbc's in 0.10s
Doing aes-128-cbc for 3s on 64 size blocks: 34651 aes-128-cbc's in 0.05s
Doing aes-128-cbc for 3s on 256 size blocks: 25926 aes-128-cbc's in 0.10s
Doing aes-128-cbc for 3s on 1024 size blocks: 20274 aes-128-cbc's in 0.04s
Doing aes-128-cbc for 3s on 8192 size blocks: 5656 aes-128-cbc's in 0.02s

# SIGNING MESSAGES

# Private Key to Sign Messages

Suppose Johnson uses Margaret's public key to send her an encrypted message. In the message, he can claim to be Johnson but Margaret has no way of verifying that the message was actually from Johnson since anyone can use Margaret's public key to send her encrypted messages. In order to verify the origin of a message, RSA can also be used to sign a message.

For make the explanation easy, Johnson will send 30 digits as a message.
Johnson also need to use the sum of the last 5 digits for signing this message.

Johnson wants to send a message of "103849194856472381948572356239".
The last 5 digits are "56239". The sum of "56239" is 25.

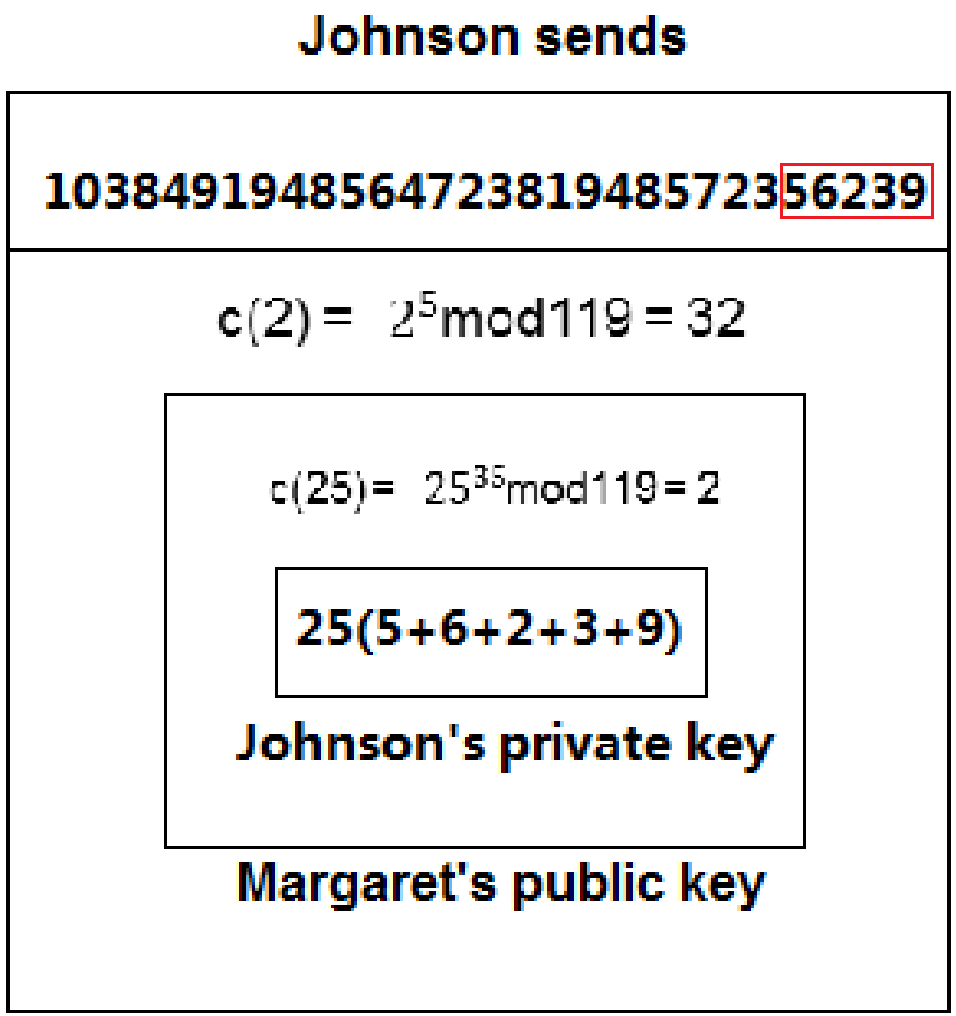
Margaret's public key: n = 119 e = 5 private key: n = 119 d = 29
Johnson's public key: n = 119 e = 11 private key: n = 119 d = 35

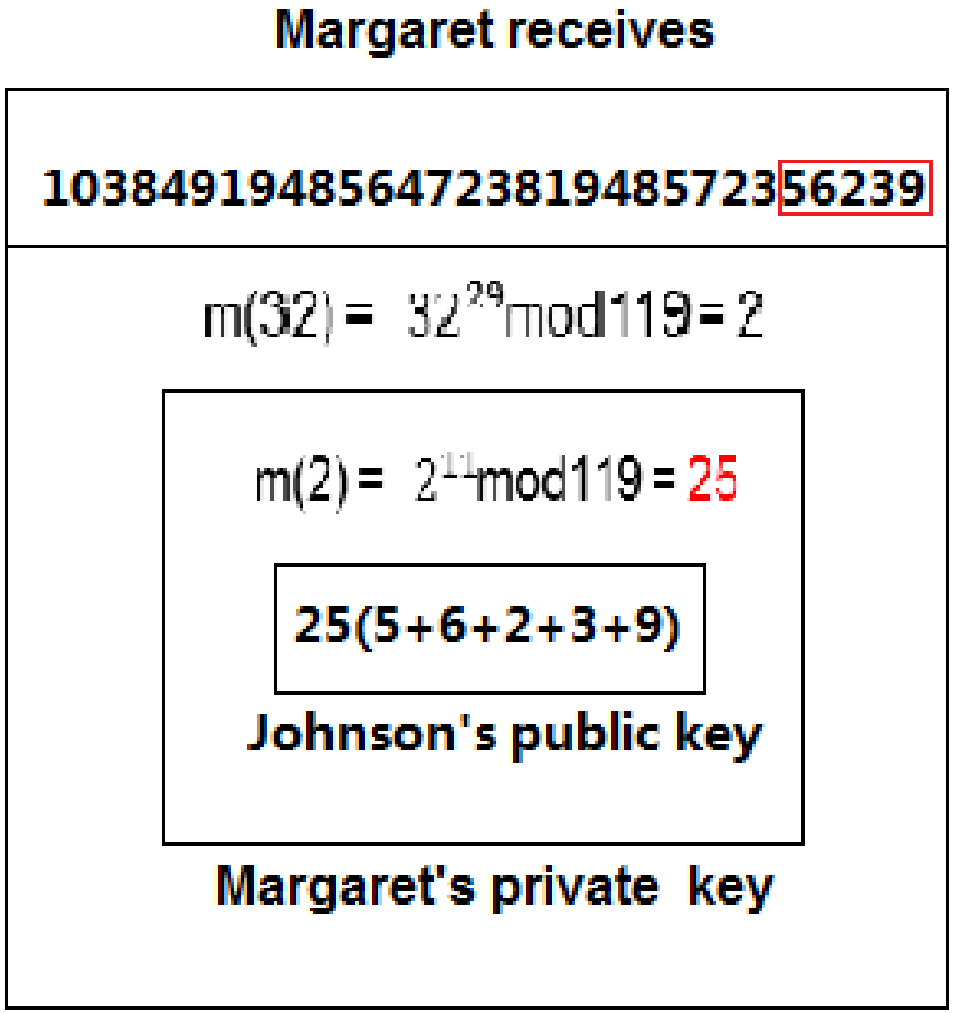
Johnson uses his private key to sign 25: c(25) = $25^{35} \bmod 119 = 2$
Johnson encrypts the message with Margaret's public key: c(2) = $2^{5} \bmod 119 = 32$
Johnson sends the message with his sign and claims this is Johnson sent: 103849194856472381948572356239 + 32

Margaret receives the message and the sign: 103849194856472381948572356239 + 32
Margaret calculates the last 5 digits in the message is 25.
Margaret uses her private key to decrypt the sign: m(32) = $32^{29} \bmod 119 = 2$
Margaret uses Johnson's public key to decrypt the sign: m(2) = $2^{11} \bmod 119 = 25$
Margaret verify this 25 with the 25 she calculates from message. It is equal. So it is from Johnson.

Notes: In the real world, the "103849194856472381948572356239" also need to be encrypted.

# Private Key to Sign Messages(Cont.)



SECURE CONNECTIONS
FOR A SMARTER WORLD