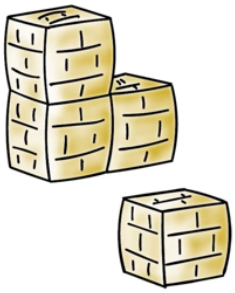
Reference: Computer Security by Stallings and Brown, Chapter 21

Public Key Algorithms
Lesson Introduction

In this lesson, we will first review the modular arithmetic using Public Key cryptography. We will then study RSA and Diffie-Hellman.

- Modular arithmetic
- RSA
- Diffie-Hellman

Modular Arithmetic



- Public key algorithms are based on modular arithmetic
- Modular addition
- Modular multiplication
- Modular exponentiation

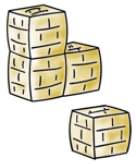
Before we discuss the widely used public key algorithms, RSA and Diffie–Hellman, let's go over the background first.

Both RSA and Diffie–Hellman are based on number theory. An ancient and yet active field in mathematics. These public key algorithms use modular arithmetic, including:

- modular addition,
- modular multiplication, and
- modular exponentiation.

And they make use of results from number theory. Let's briefly introduce the background of modular arithmetic.

Modular Arithmetic



- Addition modulo (MOD) *M*
- Additive inverse: addition MOD *M* yields 0
  - E.g., M=10, for k=2, its inverse is k^{-1}=8 because 2+8 MOD 10 = 0
- Reversible: by adding the inverse
  - Convenient for decryption
  - E.g., for c = 3, p = c+k = 3+2 MOD 10 = 5; $p+k^{-1}$ = 5+8 MOD 10 = 3 = c

First, let's explain modular addition. Given a modular m, x plus y, mod m is the remainder of x plus y divided by M. For example, 2 plus 8 divided by 10 the remainder is 0. So therefore 2 plus 8 MOD 10 is 0 whereas, 3 plus 8 MOD 10 is 1, because 3 plus 8 is 11. And 11 divided by 10, the remainder is 1. Therefore, 3 plus 8 MOD 10 is 1.

The modular addition if we add a number and its inverse, then the result is 0. For example, if the modulus is 10 then for k equal to 2 is inverse is 8 because 2 plus 8 MOD 10 is 0.

Having additive inverse means that modular addition is reversible. That is, by adding the inverse we can reverse the result of modular addition. This is very convenient for encryption and decryption because, as we have discussed, we want the encryption process to be reversible, meaning that ideally, the decryption process is just the reverse of the encryption process.

For example, given plain text, p = 3 Suppose the key k is 2, and the way we include the plain text p is to add the key under modular addition to plain text p and the result is cipher text c. So the c equal to p plus k, which is 3 plus 2 MOD 10. The result is 5. There is the cipher text c of plain text p equal to 3 is 5. Now how do we decrypt? To decrypt we use the same process but instead of using the key we use the key's inverse. We know that the inverse of 2 is 8. So therefore c plus k inverse is 5 plus 8 MOD 10 that is, 13 MOD 10. The result is 3, which is exactly the plain text. And so this is how we can do decryption by using the exact same process as the encryption, and instead of using the key, we use its inverse in decryption. And we take advantage the fact that under modular addition, each number has an inverse. Therefore each key has an inverse and we just use the inverse of the key in decryption.

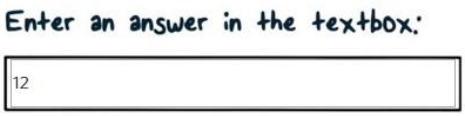
Additive Inverse Quiz

What is the additive inverse of 8 MOD 20?

Enter an answer in the textbox:

Now let's do a quiz. What is the additive inverse of 8 MOD 20? You can write the answer in this box.

The inverse of 8 is a number that when we add to 8 MOD 20 will result in 0. So obviously the answer is 12, because 8 + 12 MOD 20 is 0, because 8 plus 12 is 20, 20 MOD 20 is 0. Therefore, the additive inverse of 8 is 12.



Modular Multiplication

- Multiplication modulo M
- Multiplicative inverse: multiplication MOD M yields 1
  - E.g., M=10, 3 and 7 are inverse of each other because 3×7 MOD 10 = 1
- Only some numbers have inverse
  - But 2, 5, 6, 8 do not have inverse when M=10

For modular multiplication, the story is a bit more complicated.

First, similar to modular addition, in modular multiplication the result of x times y not m is the remainder of x times y divided by m. In modular multiplication, the result of a number times its inverse, MOD M, is one. For example, if the modular is 10, then three and seven are inverse of each other because three times seven is 21 and 21 MOD 10 is one. Because, the remainder of 21 divided by 10 is one.

It turns out, that, for a given modular, only some numbers have inverse. For example, if the modular is 10, then the numbers two, five, six, eight do not have inverse. For example, for the number two, can you find its inverse, say x, such that two times x, MOD 10 is one? There's no such x.

Modular Multiplication

- **Use Euclid's algorithm to find inverse**
  - Given x, n, it finds y such that x×y mod n = 1
- **Only the numbers relatively prime to n** has MOD n multiplicative inverse

The great mathematician Euclid invented the method to find multiplicative inverse. That is, for the modular n given X, Euclid had an algorithm to find Y. Such that X times Y mod N equal to one. Euclid also proved that only the numbers relatively prime to N has MOD N multiplicative inverse.

Modular Multiplication Quiz

Now, let's do a quiz. What is the multiplicative inverse of 3 MOD 17? Enter your answer here. That is, you need to find a number, let's say X, such that 3 times X MOD 17, equal to 1.

What is multiplicative inverse of 3 MOD 17?

Enter an answer in the textbox:

The answer is six because three times six equals to 18 equal to one mod 17.

Enter an answer in the textbox:

6

Totient Function

$\varphi(n)$

- **x is relatively prime to n**: no common factor other than 1
- **Totient function ø(n):** number of integers smaller than n and relatively prime to n
  - if n is prime, ø(n)=n-1
  - if n=p×q, and p, q are primes, ø(n)=(p-1)(q-1)
  - if n=p×q, and p, q are relative prime to each other, ø(n)=ø(p)ø(q)

Euclid 's theory says that only the numbers that are relatively prime to n have multiplicative inverse mod n. So, what does relatively prime mean? If x is relatively prime to n, that means there is no common factor between x and n other than 1. For example, three is relatively prime to 10 whereas two is not relatively prime to 10.

For given n how many integers are relatively prime to n? This is measured as a totient function of n. How do we compute totient n? If n is a prime number, then totient n is n-1, because every number that's smaller than n is also relatively primed to n because n is prime number itself. If n equal to p times q and p and q are prime numbers, then totient n is p minus 1 times q minus 1. You can verify this easily on your own.

More generally, if n equals to p times q and p and q are relatively prime to each other, then totient n is totient p times totient q.

Totient Quiz

If n = 21, what is ø(n)?

Enter an answer in the textbox:

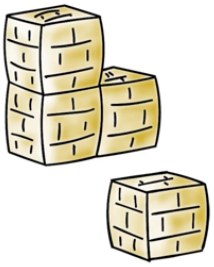
Let's do a quiz. If n is 21, what is totient(n)? Write your answer in this box.

Enter an answer in the textbox:

12

We know that 21 equals to 3 times 7. And 3 and 7 are prime numbers. Therefore, totient 21 should be 2 times 6. And the result is then 12. So 12 is the answer.

Modular Exponentiation



- $x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$
- if $y = 1 \bmod \phi(n)$ then $x^y \bmod n = x \bmod n$

In modular exponentiation there is a very nice property of quotient function from number theory. That is, if you raise x to the power of y mod n, the result is the same as x raised to the power y mod quotient n.

Given this property, if y equal to one mod quotient n then if you raise x to power y mod n, the result is the same as x mod n.

Modular Exponentiation Quiz

Use the totient technique to find c. **Write your answer in the textbox:**

$c = 7^{27} \bmod 30$

c =

Now let's do a quiz. Use the totient technique to compute the result of raise 7 to the power of 27 mod 30. Write your result in this box.

Using totient technique, we know that this is the same as 7 raised to the power of [27 mod totient 30] mod 30. So what is 27, mod, totient 30? First we need to compute what is totient 30? Totient 30 is computed as follows. 30 is 3 times 10, and 3 and 10 are relatively prime to each other. So, therefore, totient 30 is totient 3 times totient 10. Totient 3 is 2, because 3 is a prime number. Totient 10 is 4 because 10 equal to 2 times 5 and they're both prime numbers. Therefore, totient 30 is 2 times 4, equal to 8. So then what is 27 mod 8? The result is 3. Therefore, this is same as 7 raised to the power of 3 mod 30. And the result is 13.

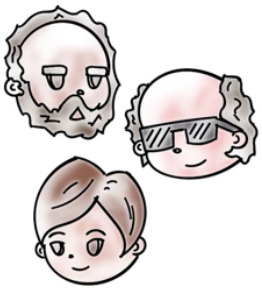
$c = 7^{27} \bmod 30$

$c =$ 13

$$
\begin{aligned}
c = 7^{27} &= 7^{27 \bmod \text{totient}(30)} \bmod 30 \\
&= 7^{27 \bmod [\,\text{totient}(3) * \text{totient}(10)\,]} \bmod 30 \\
&= 7^{27 \bmod [\,2 * 4\,]} \bmod 30 \\
&= 7^{27 \bmod 8} \bmod 30 \\
&= 7^{3} \bmod 30 \\
&= 343 \bmod 30 = 13
\end{aligned}
$$

Instructor Notes: Modular Exponentiation Calculator

RSA (Rivest, Shamir, Adleman)



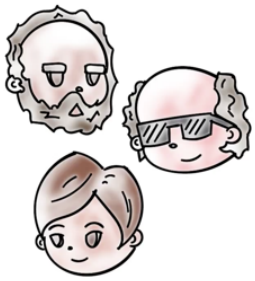
- **Widely used**, and one of the first (1977)
- Support both **public key encryption and digital signature**
- **Assumption/theoretical basis:**
  - Factoring a very large integer is hard

primes.

With a math background we can now describe some of the most widely used public key algorithms.

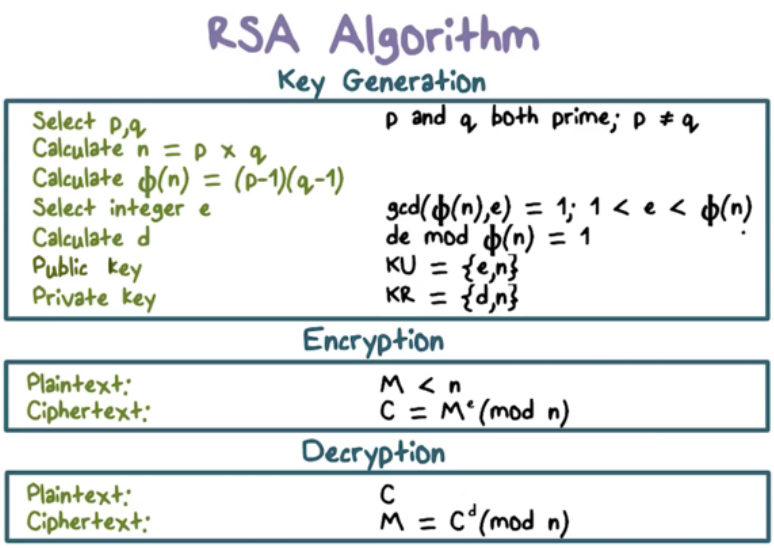
The first is RSA. Named after its inventors. R backup RSA is the most widely used public key algorithm and one of the very first. RSA supports both public key encryption and digital signature. The security strength of RSA is based on the hypothesis that, factoring a very large number into two primes is a very hard problem. This is, given a large number it is computationally very hard to factor it into two

RSA Characteristics



- **Variable key length**
- **Variable plaintext block size**
  - Plaintext treated as an integer, and must be “smaller” than the key
  - Ciphertext block size is the same as the key length

RSA can support variable key lengths. In practice, most people use a key length of 1,024 bits, or 2,048 bits, or even 4,096 bits. The plaintext length can also be variable. In RSA, every data is treated as an integer because we can interpret any data with bits of one and zero as an integer. And the requirement here is that the plaintext input has to be smaller in integer value than the key. The size of ciphertext is the same as the key length. This summarizes the RSA algorithm.

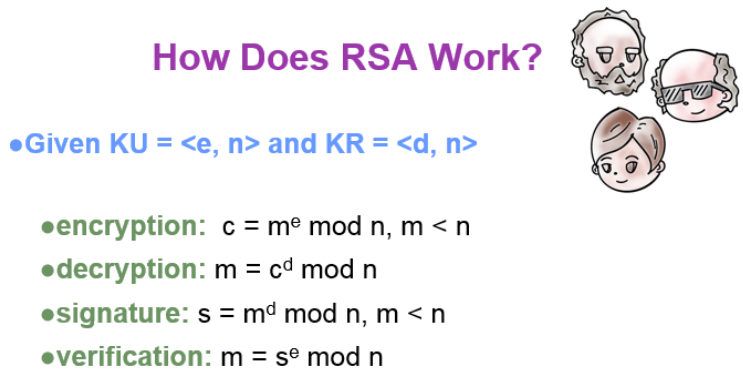


The first step is key generation.

- First we select two prime numbers, p and q. Say each of them are at least 512 bits long.
- And then we compute p times q. The result's n.
- And while we have p and q we also compute totient n. Which is p minus one times q minus one because both p and q are prime numbers.
- Then we select an integer e that is relatively prime to totient n.
- After we have selected e, we find the multiplicative inverse of e marked totient n.
- Having computed n, e and d, now we can forget about p and q and forget about totient n.
- The public key is e and n.

The private key, which we have to keep to ourself and secure it, is d and n.

Now for encryption, suppose Alice has published her public key. And Bob wishes to send a message, M, to Alice. And he wants only Alice to be able to read M. So Bob obtains Alice's public key and to encrypt message M, Bob computes M raised to the power of e (mod n). For decryption on receipt of this ciphertext, C, Alice will use her own private key, D, and compute C raised to the power of d mod n. And this would result in the original plaintext M. The property of RSA guarantees that only Alice Can decrypt this message because only she has the private key that's paired with the public key that was used to encrypt the message.



Again, every user can publish his or her own public key and keep his or her own private key securely to himself or herself. To encrypt message to a user say, Alice, Bob would obtain Alice's public key and compute the ciphertext by raising the plaintext to the power of e mod n. To decrypt, Alice would use her own private key to raise the ciphertext to the power of d and then mod n.

What about creating digital signature? Alice will use her own private key to raise the message m to a power of d mod n. To verify, Bob would get Alice's public key, raise the signature to a power of e and mod n, and this will produce the original message m and verify the signature. Again here the property of RSA guarantees that Bob will know that the signature was created by Alice, because only she has the private key that is paired with the public key that Bob was able to use to verify the signature.

Why does RSA Work?

Given pub = <e, n> and priv = <d, n>

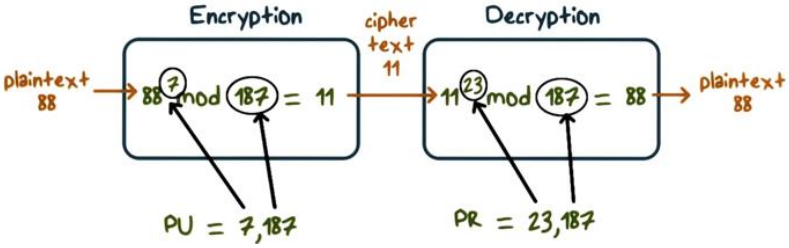
- n =p×q, ø(n) =(p-1)(q-1)
- e×d = 1 mod ø(n)
- $x^{e \times d} = x \bmod n$
- **encryption:** $c = m^e \bmod n$
- **decryption:** $m = c^d \bmod n =$
  $m^{e \times d} \bmod n =$
  $m \bmod n = m$ (since $m < n$)
- digital signature (similar)

Here's the math on why RSA works.

We know that the public key is e and n and the private key is <d, n>. We also know that n equals to p times q, and p and q are prime numbers. Therefore, totient n is p minus 1 times q minus 1. The public key e and the public key d are multiplicative inverse of each other, mod totient n. That's why we say the public key and the private key are paired together. And therefore when we compute X raised the power of e times d it's the same as X raised to the power of e times d mod totient n which happens to be 1. That is, let me compute X raised to the power of e times d is the same as X mod n.

So that is a quick recap of the properties of the RSA public and private keys. Now, to encrypt a message m, we raise m to the power of e mod n. To decrypt, we take the ciphertext, raise it to the power of d mod n. This is the same as raising m to the power of e times d mod n, which is the same as m mod n according to this property. Since we require that the plain text input is smaller than the key, that means m mod n equal to m and that's the original plaintext. So this shows that decryption indeed works. As an exercise, you can similarly verify that digital signature under RSA also works.

Why does RSA Work?



Here's an example of RSA, the keys are generated as follows. We select two prime numbers. p equal to 17, q equal to 11 so n is p times q, which is 187 and totient n is p minus 1 times q minus 1 because p and q are prime numbers. So in this case, totient 187 is 16 times 10 which is 160. For public key we select a number that's relatively prime to totient n. In this case, we select 7, because 7 is relatively prime to 160. The private key d, is the multiplicative inverse of e Mod potion n. So the multiplicative inverse of 7 mod 160 is 23, therefore the private key is 23.

To summarize the key generation process, we have the public key 7, 187, public key 23, 187. Suppose we use a public key to encrypt the plaintext message 88. We raised 88 to a power of 7 mod 187, the result is 11. To decrypt we raise 11 to power of 23 and then mod 187. The result is 88, the same as the plaintext.

RSA Quiz

Fill in the text boxes:

Given p = 3 and q = 11

1. Compute n: n= ☐

2. Compute φ(n):
φ(n) = ☐

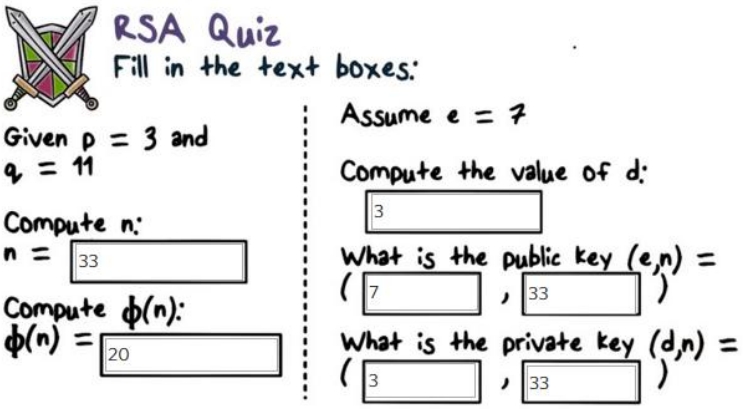
3. Assume e = 7
Compute the value of d:
d = ☐

4. What is the public key
(e,n) = (☐ , ☐)

5. What is the private key
(d,n) = (☐ , ☐)

Now let's do a quiz. This quiz has two parts. The first part is about key generation. Please fill in the answers in text boxes.

Given two prime numbers, p and q, we need to compute n, totient n, and then we need to select a public key, and from there, we need to compute a private key.



We know that n = p x q. Therefore, n is 33.

And since p and q are prime numbers, totient n is (p-1) x (q-1). Therefore, totient n is 2 x 10 = 20.

We selected public key e = 7, which is a prime number. Obviously, because it is a prime number, it's relatively prime to totient n, it [e] will have a multiplicative inverse. What is the value of t such that e times d mod totient n is equal to 1? Given that e is 7, it's obvious that d is 3 because 3 x 7 is 21 and 21 minus 20 is 1.

So what's the public key? The public key is the pair e,n. Therefore, it is 7, 33.

The private key is d,n. Therefore it is 3, 33.

p= 3, q = 11
n = (p – 1) * (q – 1) = 2 * 10 = 20
e = 7
(e * d) mod totient n = 1 → (e * 3) mod 20 = 1
Public key = <7, 33>
Private key = <3, 33>

RSA Encryption Quiz

Given:

- Public key is (e, n) => (7, 33)
- Private key is (d, n) => (3, 33)
- Message m = 2

What is the encryption of m:

What formula is used to decrypt m?

(Use ** for denoting an exponent)

In the second part of the quiz, given that we have generated the public key and the private key, then given the Message m, what is the encryption of m? And how do we decrypt to get m? You can use ** to denote an exponent.

So what's the encryption of m? To encrypt, we raised the plain text which is two to the power of public key e, and then Mod n. And the result is 29. To decrypt the cipher text which is 29, raise it to the power of the private key, which is three, and mark n, which is 33. And the result is two, which is the plain text.

Encrypt: $2^7 \bmod 33 = 29$
Decrypt: $29^3 \bmod 33 = 2$

Why RSA is Secure?

- Factoring an integer with at least 512-bit is very hard!
- **But if you can factor big number n then given public key <e,n>, you can find d, and hence the private key by:**
  - Knowing factors p, q, such that, n = p×q
  - Then compute ø(n) =(p-1)(q-1)
  - Then find d such that e×d = 1 mod ø(n)

Why is RSA secure? The security of RSA is based on the hypothesis that factoring a very large number is very, very hard. When we say a very large number, we mean an integer that is at least 512 bit, but often in the range of 1024 bits to 4096 bits. That is, even such a large integer is going to take a long time to factor it into two primes.

On the other hand, if someone finds a very efficient way to factor a large number into primes, then the security of RSA can be broken. Here's why.

The public key is given, so we know n and we know e. If we can efficiently factor n into p x q, where p and q are two primes, then we can computer totient(n) as (p- 1)(q- 1). And if we can compute totient(n) knowing the public key e, we can then find its multiplicative inverse, mod totient(n), and this inverse is the public key d. Therefore if we can efficiently factor a large integer into two primes that would mean that from the public key we can derive the private key. And that would break the security of RSA because if you know the private key then you can decrypt confidential messages or you can also forge digital signatures.

RSA in Practice

- **Issues with schoolbook RSA**
  - **Deterministic:**
    - For the same key, a particular plaintext is always mapped to a particular ciphertext
  - Special-case plaintexts 0, 1, or -1 produce ciphertexts 0, 1, or -1 regardless of keys
  - **Malleable:**
    - Transforming a ciphertext into another leads to predictable transformation to plaintext
      - For $c = m^e \bmod n$, attacker change c to $s^e \times c$
      - Receiver gets s×m instead of m

We need to consider a few issues when we use RSA in practice.

First, for given key the same plain text message will always be encrypted into same ciphertext.

A related issue is that for some special-case plaintexts, such as zero, one or minus one, the ciphertexts are always zero, one or minus one regardless of the keys. These shortcomings allow the attacker to know the plaintext even though he doesn't know the key.

Another issue is that RSA is malleable. That is if the attacker transforms a ciphertext into another, it would produce a predictable transformation of the decrypted plaintext.

Suppose Bob sends Alice an encrypted message using Alice's public key. The attacker intercepts the ciphertext. Then the attacker changes the ciphertext. Then when Bob receives this modified ciphertext, he's going to decrypt it as s times m instead of m. That is, the attacker has the ability to change the ciphertext in such a way that it would produce predictable effect on the decrypted plaintext. In this case, the plain text is increased by a factor of S.

RSA in Practice

- **PKCS** (public key cryptography standard) uses **OAEP** (optimal asymmetric encryption padding)
  - Append padding (seeded from random byte) as prefix to m

In practice, the standard uses some sort of padding. That is, add some random bytes as prefix to m, and this addresses the issues that we just discussed.

RSA in Practice Quiz

Select the best answer.

When implementing RSA, it is best to use:

- ☐ Your own custom software, to ensure a secure system
- ☐ Use the standard libraries for RSA

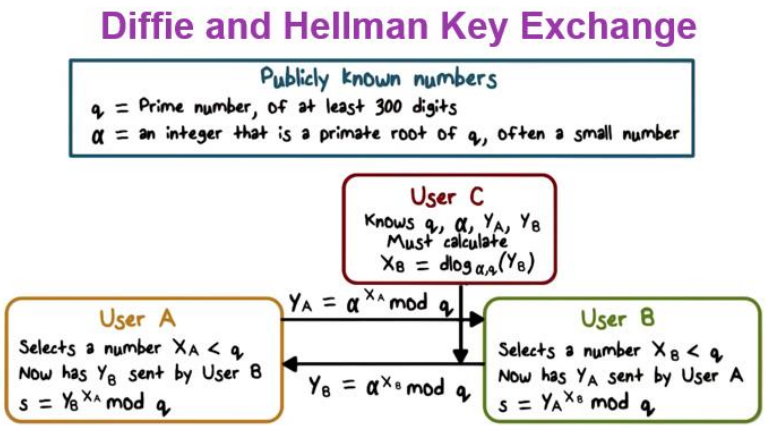
Now let's do a quiz. Select the best answer. When implementing RSA is it best to use your own custom software to ensure a secure system, or should you use the standard libraries for RSA?

The answer is, you should use the standard libraries. The reason is that the standard libraries have been reviewed and tested by the security committee and therefore are more likely to be more secure.

- ○ Your own custom software, to ensure a secure system
- ◉ Use the standard libraries for RSA

Diffie and Hellman Key Exchange

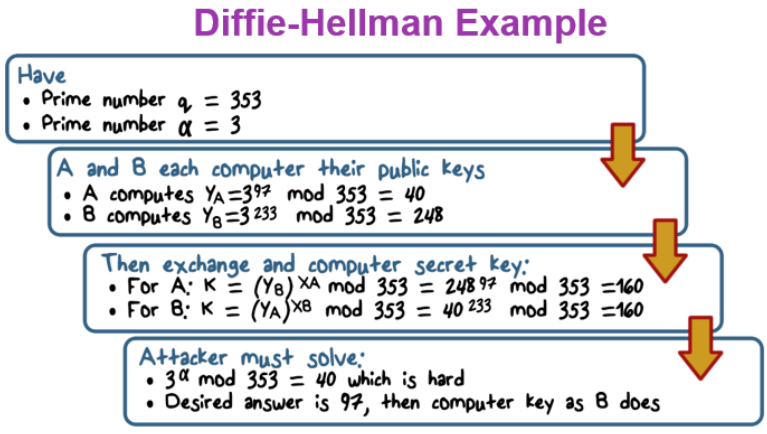
- **First published** public-key algorithm
- By Diffie and Hellman in 1976 along with the **exposition of public key concepts**
- Used in a number of commercial products
- **Practical method to exchange a secret key** securely that can then be used for subsequent encryption of messages
- Security **relies on difficulty of computing discrete logarithms**



Now let's look into the details of the Diffie-Hellman key exchange algorithm.

There are two publicly known numbers, a prime number Q and an integer alpha that is a primitive root of Q.

Now suppose users A and B wish to exchange a key using the Diffie-Hellman key exchange algorithm. User A selects a random integer XA that's less than q, and computes you that's raised alpha to the power of XA and then mod q. Likewise, User B independently selects a random integer XB that's less than q, and then computes YB, that is, alpha raised to the power of XB and then mod q. Each side keeps the X value private and then sends the Y value to the other side. Because the Y value is transmitted, in effect it is public. User A upon receiving YB would compute a key as YB raised to the power of XA mod q. This is actually the same as alpha raised to power of XB times XA then mod q. Likewise, User B upon receiving you computes a key that is you raised to the power of XB mod q, which is the same as alpha raised to the power of XA times XB mod q. That is, both sides have, in effect, computed the same key. In other words, now User A and User B have a shared secure key. Furthermore, XA and XB are private, and an adversary such as User C only knows Q and alpha, which are public, and you and YB, which is transmitted, and they are also public. If the attacker wants to compute the shared key, he must first compute the X value. For example, the attacker must compute the discrete log of YB in order to get XB. If the attacker can compute a discrete log and obtain XB, he can then compute the shared secret key by using the you value. The security of the Diffie-Hellman key exchange algorithm lies in the fact that while it's relatively easy to calculate the exponents modulo a prime, it is very difficult to compute a discrete logarithm. For large primes such as Q at at least 300 digit long, this task is considered infeasible.

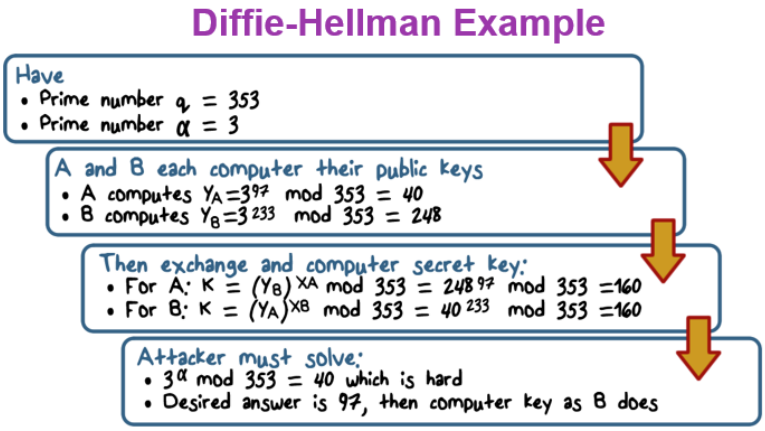


Now let's illustrate Diffie-Hellman key exchange algorithm using an example. The prime number q is 353, and the alpha, which is the primitive root of q, is 3.

User A selects a random number, which is a secret to herself, and this value is 97. So as you compute you, which is 3 raised to the power of 97 mod 353, and the result is 40. User B selects a random number, which is a secret to himself and this value is 233. And you compute YB, which is 3 raised to the power of 233 Mod 353, and the result is 248. And they exchange these values, that is user A sends 40 to user B. And user B sends 248 to user A.

Then user A computes the following, Y8 rates to the power of X8. In this case, will be 248 Raise to the power of 97 mod 353. And the result is 160. Likewise, User B] computes you raised to the power of XB, mod q. In this case, it is 40 raised to the power 233 mod 353. And the result is 160. So as you can see, by exchanging you and YB, both users A and B compute the same secret value, which is 160. And this would become the shared secret key between users A and B.

We assume that an attacker would be able to obtain these publicly exchanged values which encode in public keys, and they also know the publicly known numbers Q and alpha. In this case, because the integers involved are very small. It is feasible that the attacker will be able to figure out the secret value Xa and Xb, which is 97 and 333. However if very large number is involved, such as when the prime number Q is at least 200 bits long, then a task of finding Xa, the secret value, from Ya the public value is not feasible.



Diffie-Hellman Quiz

Alice and Bob agree to use
prime q = 23 and primitive root α = 5

Alice chooses secret a = 6
Bob chooses secret b= 15

What number does Alice send Bob? ____

What number does Bob send Alice? ____

Now let's do a quiz. Suppose Alice and Bob agree to use prime q = 23 and primitive root alpha = 5. And these numbers can be made public. And Alice chooses a secret, a = 6. Likewise Bob chooses a secret, b = 15. The question is, what number does Alice send to Bob? And that is the public number XA. Likewise, what number does Bob send to Alice? And that is the public number YB.

We know that according to the Diffie-Hellman key exchange algorithm, you is alpha which is 5 raised to the power of xa which is 6 mod q which is 23. And the result is 8. Likewise, yb is alpha which is 5 raised to the power of xb which is 15 mod q which is 23. And the result is 19.

Diffie-Hellman Quiz

Alice and Bob agree to use
prime $q = 23$ and primitive root $\alpha = 5$

Alice chooses secret $a = 6$
Bob chooses secret $b = 15$

What number does Alice send Bob? $5^6 \bmod 23 = 8$

What number does Bob send Alice? $5^{15} \bmod 23 = 19$

Instructor Notes:

Modulo Calculator
Exponent Calculator
Diffie Hellman

Diffie-Hellman Security

- **Shared key (the secret) itself never transmitted**
- Discrete logarithm is very hard
- $Y = \alpha^X \bmod q$
- **Conjecture:** given Y, α, and q, it is extremely hard to compute the value of X because q is a very large prime (discrete logarithm)

Now let's analyze the security of the Diffie-Hellman key exchange algorithm.

First, we observe that the shared key, meaning the shared secret, that is computed by both users A and B, itself is never transmitted. Further, the local secret of user A or B itself is also not transmitted. That is, the local secret, XA, is kept to user A, and the local secret, XB, is kept to user B. In summary, no secret value is being transmitted.

On the other hand, the values YA and YB are transmitted. Then the question is from you which is alpha raised to the power of XA mod q. And given that alpha and q and you are known can an emissary compute xa? The security assumption here, is that this grid algorithm when large number is involved is very hard. That is given y, alpha, and q, it is extremely hard to compute the value of x, or it is not practical to compute x. Because q is a very large prime. For example, q is a least 300 bit long.

On the other hand, if this conjecture is not true, that means an adversary knowing you and knowing alpha and q, can easily compute xa which is the local secret to user a. This would mean that the adversary can compute secret key as they're shared between user A and B, just like user A. Because now, the adversary knows the local secret of user A.

Diffie-Hellman Limitations

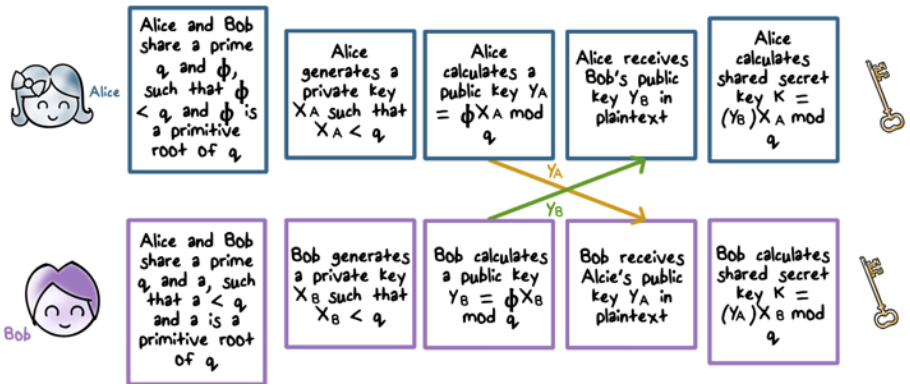
- Expensive exponential operation
  - DoS possible
- The scheme itself **cannot be used to encrypt anything –** it is for secret key establishment
- **No authentication**, so you cannot sign anything

There are a few shortcomings in Diffie-Hellman key exchange algorithm.

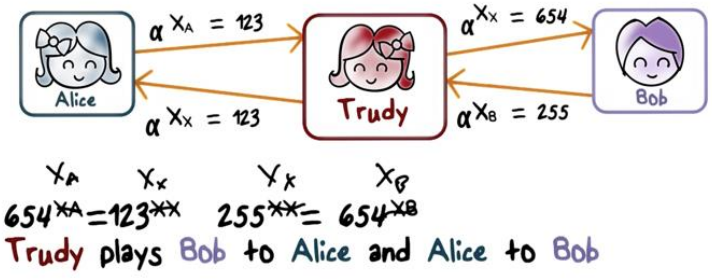
First, suppose that Alice tells Bob to use Diffie-Helman. The first thing Bob has to do is to compute his YB. And this computation involves exponentiation, which is very expensive. Therefore, if Alice is a malicious attacker and Bob is a server, then Alice can request multiple Diffie-Hellman sessions with Bob at the same time. And this would cause Bob to spend a lot of CPU cycles on exponentiations, and this can then result in denial of service.

Second, Diffie-Hellman is only for key exchange. It itself does not offer encryption. And the Diffie-Hellman algorithm does not authenticate either Alice or Bob. That is, there's no authentication and it cannot provide digital signatures like RSA.

Implementing the Diffie-Hellman Key Exchange



Bucket Brigade Attack, Man-in-the-Middle(MIM)



The biggest threat to Diffie-Hellman key exchange is the so-called bucket brigade attack. It is a kind of man-in-the-middle attack. That is Trudy, the adversary, can intercept the message YA that Alice sends to Bob. And instead, Trudy sends her own YX to Bob. And fooling Bob to accept this as YA. Likewise, Trudy intercepts YB that Bob sends to Alice, and instead sends her own YX to Alice. Fooling Alice to believe that YX is actually YB. And the result is that the shared key that Alice computes, is actually the shared key between Alice and Trudy. And likewise, the shared key that Bob computes, is actually the shared key between Trudy and Bob. In other words, Trudy Plays Bob to Alice, and Alice to Bob.

This man in the middle attack is possible because the Diffie–Hellman key exchange protocol does not authenticate Alice or Bob. For example, there's no way for Alice to know that the message that she receives is really from Bob.

There are a number of ways to fix this. For example, everyone, for example, Alice and Bob, can publish her or his public key, YA or YZ. That is, instead of having to send it and risk interception and forgery, just publish YA or YB at a public trusted site. Or, if Alice has already published her RSA public key, she can signed you when she sends it to Bob, so that Bob would know that YA is really from Alice. Because Bob can verify that uses Alice's RSA public key.

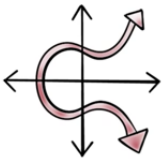
Other Public-Key Algorithms

Digital Signal Standard:

- Makes use of SHA-1 and the Digital Signature Algorithm (DSA)
- Originally proposed in 1991, revised in 1993 due to security concerns, and another minor revision in 1996
- **Cannot be used for encryption or key exchange**
- Uses an algorithm that is designed to provide only the digital signature function

In addition to RSA and Diffie-Hellman which are the most widely used algorithms. There are other public key algorithms. First, the National Institute of Standards and Technology or NIST, has published the digital signature standard. It is based on the secure hash function SHA1. Note that this algorithm is only for signature. Is not for encryption or key exchange.

Other Public-Key Algorithms



Elliptic-Curve Cryptography (ECC):

- Equal security for smaller bit size than RSA
- Seen in standards such as IEEE P1363
- Confidence level in ECC is not yet as high as that in RSA
- **Based on a mathematical construct known as the elliptic curve**

RSA is the most widely used public key cartography algorithm for encryption and digital signatures.

Recently a competing system called elliptic curve cryptography has begun to challenge RSA. The main advantage of ECC over RSA is that if offers the same security with a far smaller bit size. That is, for the same security strength, it can be much more efficient than RSA.

On the other hand, RSA has been subject to a lot of cryptanalysis work over the years, whereas for ECC, the cryptanalysis work is still at a beginning, therefore we are not as confident in the ECC as we are in RSA.

ECC is more difficult explained in the form that medical description is beyond the scope of this course. But briefly, the technique is based on the use of a mathematical construct known as the elliptic curve. Roughly speaking, two points, p

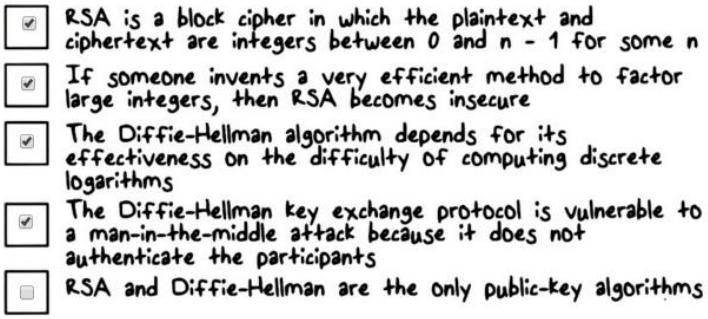
and t on the elliptic curve are known or made public. And p and t have the following relations, t times p equal t, under the so called point multiplication and the fact that d is the private key. Mathematically, even p and t, which are public, it is very hard to find t, which is private.

RSA, Diffie-Hellman Quiz

Check the statements that are True:

- ☐ RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n
- ☐ If someone invents a very efficient method to factor large integers, then RSA becomes insecure
- ☐ The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms
- ☐ The Diffie-Hellman key exchange protocol is vulnerable to a man-in-the-middle attack because it does not authenticate the participants
- ☐ RSA and Diffie-Hellman are the only public-key algorithms

Now, let's do a quiz on RSA and Diffie-Hellman. Check the statements that are true. First statement. RSA is a block cipher in which the plaintext and ciphertext are integers between zero and n-1 for some n. Second. If someone invents a very efficient method to factor large integers, then RSA becomes insecure. Third, the Diffie-Hellman algorithm depends, for its effectiveness, on the difficulty of computing discrete logarithms. Fourth, the Diffie-Hellman key exchange protocol is vulnerable to a man-in-the-middle attack, because it does not authenticate the participants. Fifth. RSA and Diffie-Hellman are the only public-key algorithms.



First, RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and n- 1 for some n. This is true because for RSA we require that the plaintext input is smaller than n and the server text has the same length as the plaintext. Therefore the servertext is also smaller than n.

Second, if someone invents a very efficient method to factor large integers then RSA becomes insecure. This is true because if you can factor efficiently that means for public key n, you can factor n equal to p times q where p and q are prime numbers. And knowing p and q, you can compute totient n. And given the public key e, then you can compute a private key d, which is the multiplicative inverse of e, mod totient n. In short, if you can factor efficiently, that means you can recover the RSA private key.

Third, The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. This is true, because if it is easy to compute discrete logarithms, then given the publicly exchanged value say YA, an adversary can compute the local secret XA. And knowing the XA, the adversary can then compute a shared secret s.

Four, the Diffie-Hellman key exchange protocol is vulnerable to man-in-the-middle attack because it does not authenticate the participants. This is true. Because the protocol does not authenticate. When Alice receives a message, she has no way of knowing that the message is really from Bob. That is, Alice has no way of knowing that the value of YB, which she uses to compute a shared secret s, is really from Bob. Which means Trudy can send YX to Alice and fool Alice to believe that it is YB.

Fifth, RSA and Diffie-Hellman are the only public-key algorithms. This is false. For example, there are other public key algorithms such as the digital signature standard and the elliptic curve cryptography.

Public Key Algorithms

Lesson Summary

- Modular arithmetic the foundations of several public key algorithms
- RSA can be used for encryption and signature, its security is based on assumption that factoring a larger integer into two primes is hard
- Diffie-Hellman is used for key exchange, its security is based on the assumption that discrete logarithm on large numbers is hard
  - No authentication means man-in-the-middle attack possible

RSA can be used for encryption and signature. Its security is based on the assumption that factoring a large number is very hard. Diffie-Hellman is used for key exchange. Its security is based on the assumption that the discrete log on large numbers is very hard.