QUARTA LISTA DE PROGRAMAS

Criptografia e Segurança de Redes – 01-2019

Disponibilizada em 27/05/2019 – A ser entregue em 28/06/2019

1. Escreva um programa que implemente o algoritmo RSA e que seja capaz de cifrar e decifrar um texto de mensagem que aceite todos os caracteres da tabela ASCII estendida (de 0 a 255). O programa terá como entradas os valores dos primos p e q, e o valor da chave pública e, e deverá calcular o valor de $n = p \times q$, e o valor de $\phi(n)$.

   Em seguida o programa deverá checar a validade da chave pública provida e (ela deve ser prima relativamente ao valor de $\phi(n)$, isto é mdc(e, $\phi(n)$) = 1). Caso a chave pública e não seja válida, o programa deve imprimir uma mensagem de erro e solicitar outra chave pública, e continuar fazendo assim até que uma chave pública válida seja fornecida.

   Quando a chave pública e provida for válida, o programa deve calcular a chave privada d.

   Em seguida o programa deverá solicitar ao usuário se a ação desejada é cifração ou decifração. Finalmente o programa irá solicitar ao usuário o *path* do arquivo onde se encontra o texto a ser cifrado ou decifrado e prosseguir com a cifração (usando o valor de e) ou a decifração (usando o valor de d) do texto. O texto resultante da cifração ou decifração deverá ser armazenado em um outro arquivo.

2. Escreva um programa que calcule e liste todos os pontos de uma curva elíptica do tipo Z_p, $y^2 = x^3 + ax + b \pmod p$, com sua respectiva ordem (isto é, quantas instâncias do número devem ser somadas para se obter o ponto no infinito O). O programa terá como entradas os valores de a, b e p. Além de listar os pontos da curva e sua ordem, o programa também terá como saída o número total de pontos da curva elíptica e um dos pontos de maior ordem (pode ter mais de um ponto com a maior ordem).

3. Escreva um programa que implemente um algoritmo de criptografia de curva elíptica (cifração e decifração) do tipo Z_p, $y^2 = x^3 + ax + b \pmod p$. Este programa deverá ter como entradas:

   a) Os valores de a, b e p;
   b) Um ponto base G (G_x , G_y) para ser utilizado na cifração e decifração; e
   c) Um ponto da curva a ser cifrado P (P_x, P_y).

   O programa deverá cifrar o ponto dado P gerando os dois pontos cifrados correspondente C_1 e C_2. Em seguida, o programa deverá decifrar os pontos C_1 e C_2, usando o algoritmo de decifração da mesma curva elíptica, obtendo o ponto P', que deverá ser igual a P. Os pontos C_1 e C_2, e depois P', serão mostrados na tela durante o processo de decifração.

   Dica: use o programa do item 2 para obter o ponto base G e o ponto P a ser cifrado.

4. Escreva um programa que utilize uma curva elíptica definida no programa do item 2 e implemente o seguinte:

    a) O algoritmo de geração e verificação de assinaturas digitais apresentado a seguir (ECDSA _ Elliptic Curve Digital Signature Algorithm), utilizando o algoritmo de hash SHA256 (que você deverá também implementar);

    b) Um programa que seja capaz de tomar um arquivo contendo um texto de mensagem qualquer, gerar a assinatura digital para a mensagem contida neste arquivo (utilizando o ECDSA implementado no item a) e guardar a esta assinatura em outro arquivo.

    c) Um programa que seja capaz de tomar o arquivo de mensagem e o arquivo contendo a assinatura digital gerada pelo programa do item b e verificar se a assinatura é válida (utilizando o ECDSA implementado no item a).

13.5 ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM

As was mentioned, the 2009 version of FIPS 186 includes a new digital signature technique based on elliptic curve cryptography, known as the **Elliptic Curve Digital Signature Algorithm (ECDSA).** ECDSA is enjoying increasing acceptance due to the efficiency advantage of elliptic curve cryptography, which yields security comparable to that of other schemes with a smaller key bit length.

First we give a brief overview of the process involved in ECDSA. In essence, four elements are involved.

1. All those participating in the digital signature scheme use the same global domain parameters, which define an elliptic curve and a point of origin on the curve.

2. A signer must first generate a public, private key pair. For the private key, the signer selects a random or pseudorandom number. Using that random number and the point of origin, the signer computes another point on the elliptic curve. This is the signer's public key.
3. A hash value is generated for the message to be signed. Using the private key, the domain parameters, and the hash value, a signature is generated. The signature consists of two integers, r and s.
4. To verify the signature, the verifier uses as input the signer's public key, the domain parameters, and the integer s. The output is a value v that is compared to r. The signature is verified if the $v = r$.

Let us examine each of these four elements in turn.

Global Domain Parameters

Recall from Chapter 10 that two families of elliptic curves are used in cryptographic applications: prime curves over Z_p and binary curves over GF(2^m). For ECDSA, prime curves are used. The global domain parameters for ECDSA are the following:

q	a prime number
a, b	integers that specify the elliptic curve equation defined over Z_q with the equation $y^2 = x^3 + ax + b$
G	a base point represented by $G = (x_g, y_g)$ on the elliptic curve equation
n	order of point G; that is, n is the smallest positive integer such that $nG = O$. This is also the number of points on the curve.

Key Generation

Each signer must generate a pair of keys, one private and one public. The signer, let us call him Bob, generates the two keys using the following steps:

1. Select a random integer d, $d \in [1, n-1]$
2. Compute $Q = dG$. This is a point in $E_q(a, b)$.
3. Bob's public key is Q and private key is d.

Digital Signature Generation and Authentication

With the public domain parameters and a private key in hand, Bob generates a digital signature of 320 bytes for message m using the following steps:

1. Select a random or pseudorandom integer k, $k \in [1, n-1]$
2. Compute point $P = (x, y) = kG$ and $r = x \bmod n$. If $r = 0$ then goto step 1
3. Compute $t = k^{-1} \bmod n$
4. Compute $e = H(m)$, where H is the SHA-1 hash function, which produces a 160-bit hash value
5. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = O$ then goto step 1
6. The signature of message m is the pair (r, s).

Alice knows the public domain parameters and Bob's public key. Alice is presented with Bob's message and digital signature and verifies the signature using the following steps:

1. Verify that r and s are integers in the range 1 through $n - 1$
2. Using SHA-1, compute the 160-bit hash value $e = \mathrm{H}(m)$
3. Compute $w = s^{-1} \bmod n$
4. Compute $u_1 = ew$ and $u_2 = rw$
5. Compute the point $X = (x_1, y_1) = u_1G + u_2Q$
6. If $X = O$, reject the signature else compute $v = x_1 \bmod n$
7. Accept Bob's signature if and only if $v = r$

Figure 13.6 illustrates the signature authentication process. We can verify that this process is valid as follows. If the message received by Alice is in fact signed by Bob, then

$$s = k^{-1}(e + dr) \bmod n$$

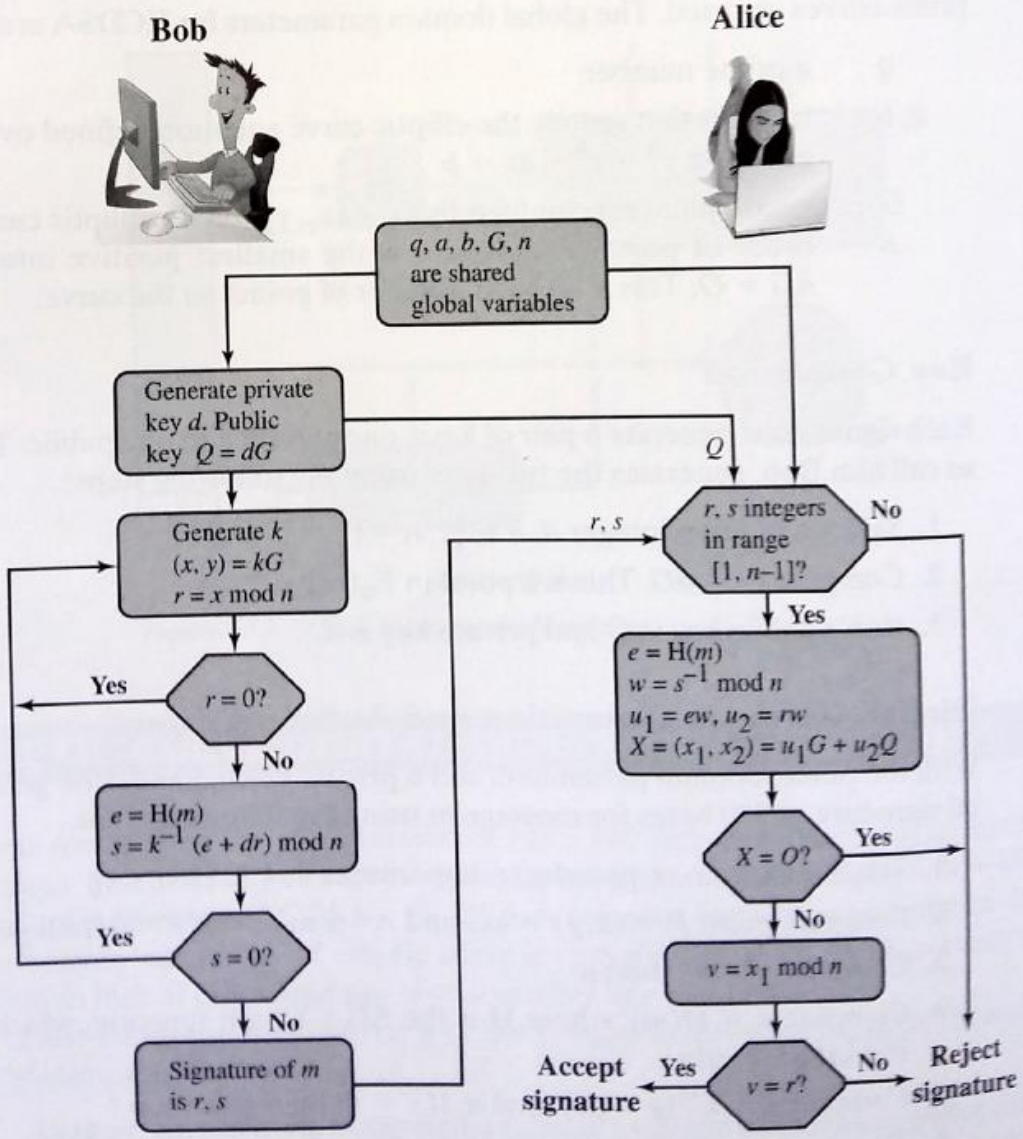

Figure 13.6 ECDSA Signing and Verifying

Then

$$k = s^{-1}(e + dr) \bmod n$$
$$k = (s^{-1}e + s^{-1}dr) \bmod n$$
$$k = (we + wdr) \bmod n$$
$$k = (u_1 + u_2 d) \bmod n$$

Now consider that

$$u_1 G + u_2 Q = u_1 G + u_2 dG = (u_1 + u_2 d)G = kG$$

In step 5 of the verification process, we have $v = x_1 \bmod n$, where point $X = (x_1, y_1) = u_1 G + u_2 Q$. Thus we see that $v = r$ since $r = x \bmod n$ and x is the x coordinate of the point kG and we have already seen that $u_1 G + u_2 Q = kG$.