Cheat Sheet For Cybersecurity

RSA Public Key Cryptography

Unlike Diffie-Hellman, RSA can be used for key exchange as well as digital signatures and the encryption of small blocks of data. Today, RSA is primarily used to encrypt the session key used for secret key encryption (message integrity) or the message's hash value (digital signature). RSA's mathematical hardness comes from the simplicity in calculating large numbers and the difficulty in finding the prime factors of those large numbers. Although employed with numbers using hundreds of digits, the math behind RSA is relatively straight-forward.

To create an RSA public/private key pair, here are the basic steps:

1. Choose two prime numbers, p and q. From these numbers you can calculate the modulus, n = pq.
2. Select a third number, e, that is relatively prime to (i.e., it does not divide evenly into) the product (p-1) (q-1). The number e is the public exponent.
3. Calculate an integer d from the quotient (ed-1)/[(p-1) (q-1)]. The number d is the private exponent.

The public key is the number pair (n, e). Although these values are publicly known, it is computationally infeasible to determine d from n and e if p and q are large enough.

To encrypt a message, M, with the public key, create the ciphertext, C, using the equation:

$$C = M^e \bmod n$$

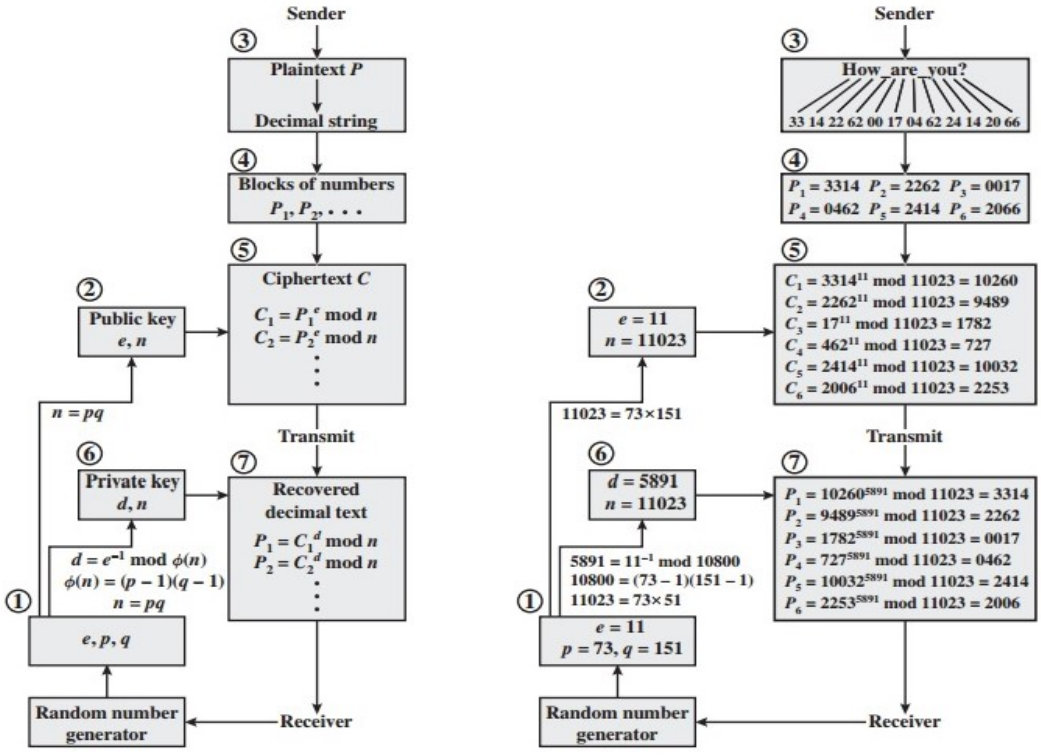
The receiver then decrypts the ciphertext with the private key using the equation:

$$M = C^d \bmod n$$

Example:

1. Select p=3 and q=5.
2. The modulus n = pq = 15.
3. The value e must be relatively prime to (p-1) (q-1) = (2)(4) = 8. Select e=11.
4. The value d must be chosen so that (ed-1)/[(p-1) (q-1)] is an integer. Thus, the value (11d-1)/[(2)(4)] = (11d-1)/8 must be an integer. Calculate one possible value, d=3.
5. Let's suppose that we want to send a message — maybe a secret key — that has the numeric value of 7 (i.e., M=7).
6. The sender encrypts the message (M) using the public key value (e, n) = (11,15) and computes the ciphertext (C) with the formula $C = 7^{11} \bmod 15 = 1977326743 \bmod 15 = 13$.
7. The receiver decrypts the ciphertext using the private key value (d, n) = (3,15) and computes the plaintext with the formula $M = 13^3 \bmod 15 = 2197 \bmod 15 = 7$.

RSA – Real-Life example:



(a) General approach (b) Example

RSA Processeing of Multiple Blocks

****

ElGamal public-key encryption

ElGamal encryption is based on the Diffie-Hellman Key Exchange method. It uses the same domain parameters (p, q, g) and private/public key pair (b, B = g^b mod p) for a recipient B. The plaintext message to be encrypted needs to be encoded as an integer mm in the range [1, p−2].

Algorithm: ELGAMAL ENCRYPTION

INPUT: Domain parameters (p, q, g); recipient's public key B; encoded message m in range $0 < m < p - 1$.
OUTPUT: Ciphertext (c_1, c_2).

1. Choose a random k in the range $1 < k < p - 1$.
2. Compute $c_1 = g^k \bmod p$
3. Compute $c_2 = mB^k \bmod p$
4. Return ciphertext (c_1, c_2).

Algorithm: ELGAMAL DECRYPTION

INPUT: Domain parameters (p, q, g); recipient's private key b; ciphertext (c_1, c_2).
OUTPUT: Message representative, m.

1. Compute $m = c_1^{p-b-1} c_2 \bmod p$
2. Return m.

Example – 1:

ElGamal public-key encryption:

INPUT: Domain parameters $(p = 283, q = 47, g = 60)$
INPUT: Bob's public key, $B = 216$;
INPUT: encoded message, $m = 101$, such that $0 < m < p - 1$.
1. Alice chooses a random $k = 36$ in the range $[2, q - 2]$
2. Alice computes $c_1 = g^k \bmod p = 60^{36} \bmod p = 78$.
3. Alice computes $c_2 = mB^k \bmod p = 101 \cdot 216^{36} \bmod p = 218$.
4. Alice sends ciphertext $(c_1, c_2) = (78, 218)$ to Bob

ElGamal public-key decryption:

INPUT: Domain parameters $(p = 283, q = 47, g = 60)$
INPUT: Bob's private key, $b = 7$;
INPUT: ciphertext $(c_1, c_2) = (78, 218)$.
1. Bob computes $m = c_1^{p-b-1} c_2 \bmod p$
$= 78^{283-7-1} \cdot c_2 = 116 \cdot 218 \bmod p$
$= 101$

Example – 2:

A. Key generation

Entity **A** selects the prime **p = 17** and a generator **α = 3** of $\mathbf{Z}_p^*$. **A** chooses the private key **a = 6** and computes $\alpha^a \bmod p = 15$, also denoted by $\beta \equiv \alpha^a \bmod p = 15$.
A's public key is **(p = 17, α = 3, α^a = β = 15).**

B. Encryption

To encrypt a message **x = 11**, **B** selects a random integer **k = 3** and computes $y_1 = \alpha^k \bmod p = 3^3 \bmod 17 = 10$ and $y_2 = x\beta^k \bmod p = 11 \times 15^3 \bmod 17 = 11 \times 9 = 14$, where $\beta = \alpha^a$. So **B** sends **y1=10** and **y2=14**, it means **(10,14)** is the encrypted message.

C. Decryption

To decrypt, **A** computes $y_1^{p-1-a} \bmod p = 10^{10} \bmod 17 = 2$ and recovers **x** by computing $x = (y_1^{p-1-a} \bmod p).y_2 \bmod p = 2 \times 14 \bmod 17 = 11$.

****

Digital Signature Algorithm (DSA)

The Digital Signature Algorithm (DSA) was introduced in 1994 by the U.S. Department of Commerce and National Institute of Standards and Technology. It uses the same Diffie-Hellman domain parameters (p, q, g) and private/public key pair (a, $A = g^a \bmod p$) for a signing party A.

In the original standard, the prime factor q was specified to be 160 bits long and the message representative to be signed was the 160-bit SHA-1 hash of the message M.

Algorithm: DSA Signature Generation

INPUT: Domain parameters (p, q, g); signer's private key a; message-to-be-signed, M; a secure hash function $\mathrm{Hash}()$ with output of length $|q|$.
OUTPUT: Signature (r, s).

1. Choose a random k in the range $[1, q-1]$.
2. Compute $X = g^k \bmod p$ and $r = X \bmod q$. If $r = 0$ (unlikely) then go to step 1.
3. Compute $k^{-1} \bmod q$.
4. Compute $h = \mathrm{Hash}(M)$ interpreted as an integer in the range $0 \leq h < q$.
5. Compute $s = k^{-1}(h + ar) \bmod q$. If $s = 0$ (unlikely) then go to step 1.
6. Return (r, s).

Algorithm: DSA Signature Verification

INPUT: Domain parameters (p, q, g); signer's public key A; signed-message, M; a secure hash function $\mathrm{Hash}()$ with output of length $|q|$; signature (r, s) to be verified.
OUTPUT: "Accept" or "Reject".

1. Verify that r and s are in the range $[1, q-1]$. If not then return "Reject" and stop.
2. Compute $w = s^{-1} \bmod q$.
3. Compute $h = \mathrm{Hash}(M)$ interpreted as an integer in the range $0 \leq h < q$.
4. Compute $u_1 = hw \bmod q$ and $u_2 = rw \bmod q$.
5. Compute $X = g^{u_1} A^{u_2} \bmod p$ and $v = X \bmod q$.
6. If $v = r$ then return "Accept" otherwise return "Reject".

In practice, q is a 160-bit prime and h is a 160-bit integer representative of the SHA-1 hash of the message M, but the algorithm still works for our small numbers.

INPUT: Domain parameters $(p = 283, q = 47, g = 60)$
INPUT: Alice's private key, $a = 24$
INPUT: Message M with message digest $h = \text{Hash}(M) = 41$.

1. Alice chooses a random $k = 15$ in the range $[1, q-1]$
2. Alice computes $X = g^k \bmod p = 60^{15} \bmod 283 = 207$ and
   $r = X \bmod q = 207 \bmod 47 = 19$.
   $r \neq 0$ so continue.
3. Alice computes $k^{-1} \bmod q = 15^{-1} \bmod 47 = 22$.
4. Alice computes $h = \text{Hash}(M) = 41$.
5. Alice computes $s = k^{-1}(h + ar) \bmod q = 22(41 + 24 \cdot 19) \bmod 47 = 30$.
   $s \neq 0$ so continue.
6. Alice issues the message M and signature $(r, s) = (19, 30)$.

Example of DSA verification by Bob (or anyone)

INPUT: Domain parameters $(p = 283, q = 47, g = 60)$
INPUT: Alice's public key, $A = 158$
INPUT: Message M with message digest $h = \text{Hash}(M) = 41$.
INPUT: Signature $(r, s) = (19, 30)$.

1. Bob verifies that $0 < r = 19 < 47$ and $0 < s = 30 < 47 \Rightarrow$ OK, so continue.
2. Bob computes $w = s^{-1} \bmod q = 30^{-1} \bmod 47 = 11$.
3. Bob computes $h = \text{Hash}(M) = 41$.
4. Bob computes $u_1 = hw \bmod q = 41 \cdot 11 \bmod 47 = 28$ and
   $u_2 = rw \bmod q = 19 \cdot 11 \bmod 47 = 21$.
5. Bob computes $X = g^{u_1} A^{u_2} \bmod p = 60^{28} \cdot 158^{21} \bmod 283 = 106 \cdot 42 \bmod 283 = 207$ and
   $v = X \bmod q = 207 \bmod 47 = 19$.
6. Bob checks that $v = 19 = r$, so he accepts the signature.

****

ElGamal Digital Signatures

- signature variant of ElGamal, related to D-H
  - so uses exponentiation in a finite (Galois)
  - with security based difficulty of computing discrete logarithms, as in D-H
- use private key for encryption (signing)
- uses public key for decryption (verification)
- each user (e.g., A) generates their key
  - chooses a secret key (number): $\mathbf{1 < x_A < q-1}$
  - compute their **public key**: $\mathbf{y_A = a^{x_A} \bmod q}$

ElGamal Digital Signature

- Alice signs a message M to Bob by computing
  - the hash $\boldsymbol{m = H(M)}$, $\mathbf{0 <= m <= (q-1)}$
  - chose random integer **K** with $\mathbf{1 <= K <= (q-1)}$ and $\mathbf{gcd(K,q-1)=1}$
  - compute temporary key: $\mathbf{S_1 = a^k \bmod q}$
  - compute K^{-1} the inverse of **K** $\mathbf{\bmod (q-1)}$
  - compute the value: $\mathbf{S_2 = K^{-1}(m-x_A S_1) \bmod (q-1)}$
  - signature is: $\mathbf{(S_1,S_2)}$
- any user B can verify the signature by computing
  - $\mathbf{V_1 = a^m \bmod q}$
  - $\mathbf{V_2 = y_A^{S1} S_1^{S2} \bmod q}$
  - signature is valid if $\mathbf{V_1 = V_2}$

ElGamal Signature Example

- use field GF(19) q=19 and a=10
- Alice computes her key:
  - A chooses x_A=16 & computes $y_A=10^{16} \bmod 19 = 4$
- Alice signs message with hash m=14 as (3,4):
  - choosing random K=5 which has gcd(18,5)=1
  - computing $S_1 = 10^5 \bmod 19 = 3$
  - finding $K^{-1} \bmod (q-1) = 5^{-1} \bmod 18 = 11$
  - computing $S_2 = 11(14-16.3) \bmod 18 = 4$
- any user B can verify the signature by computing
  - $V_1 = 10^{14} \bmod 19 = 16$
  - $V_2 = 4^3.3^4 = 5184 = 16 \bmod 19$
  - since 16 = 16 signature is valid

****

Elliptic Curves

Is an approach to public key cryptography based on the algebraic structure of elliptic curves over finite fields. It requires smaller keys compared to non elliptic curve cryptography to provide equivalent security. Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks. Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme.

For current cryptographic purposes, an elliptic curve is a plane curve over a finite field (rather than the real numbers) which consists of the points satisfying the equation

$$F : y^2 = x^3 + ax + b$$

An elliptic curve consists of all the points that satisfy an equation of this form, where $4a^3+27b^2 \neq 0 \pmod p$ and $p > 3$. Elliptic curves are symmetric about the x-axis.

Example:

If you take $a = 27$ and $b = 2$, we have

$$y^2 = x^3 + 27x + 2$$

When we take $x = 2$, we get two points $(2, 8)$ and $(2, -8)$.

Point addition can be done by adding two points on the elliptic curve to get a third point on the curve. To add two points on the curve together, you first find the line that goes through those two points. Then you determine where that line intersects the curve at a third point. Then you reflect that third point across the x-axis (i.e. multiply the y-coordinate by -1) and whatever point you get from that is the result of adding the first two points together creating $R = P + Q$. One thing to keep in mind that there are always p points on the curve, so if $p = 11$ then $11P = 0$. The procedure is thus:

- Draw a line from P to Q
- Find the third intersection on F
- Reflect on the x-axis

Given $F : y^2 = x^3 + ax + b \pmod p$ and two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, the result will be $R = (x_3, y_3)$. The rules we have apply are:

- If $x_1 = x_2$ and $y_1 = -y_2$ (points are reflected on x-axis) then $(x_3, y_3) = 0$ meaning it is a special point at infinity
- Else we have

$$x_3 = \lambda^2 - x_1 - x_2 \pmod p$$
$$y_3 = \lambda(x_1 - x_3) - y_1 \pmod p$$

where

$$\lambda = (y_2 - y_1)/(x_2 - x_1) \pmod p \quad \text{if } P \neq Q$$
$$\lambda = (3x_1^2 + a)/(2y_1) \pmod p \quad \text{if } P = Q$$

The Zero-Point is: P+ (−P) = 0, Q+ (−Q) = 0 and P+ 0 = 0 + P=P.

Example:

We have $F : y^2 = x^3 + 1 \cdot x + 5 \pmod{11}$ and $P = Q = (0, 7)$. We want to compute $R = P + Q$ where $a = 1, b = 5, p = 11$. We can see that the situation if $P = Q$, so:

$$\lambda = (3x_1^2 + a)/(2y_1) = (3 \cdot 0^2 + 1)/(2 \cdot 7) = 1/14 \pmod{11} = 14^{-1} \pmod{11} = 4 \pmod{11}$$
$$x_3 = \lambda^2 - x_1 - x_2 = 4^2 - 0 - 0 = 5 \pmod{11}$$
$$y_3 = \lambda(x_1 - x_3) - y_1 = 4(0 - 5) - 7 = -27 \pmod{11} = 6 \pmod{11}$$
$$R = (x_3, y_3) = (5, 6)$$

Example:

We have $F : y^2 = x^3 + 1 \cdot x + 5 \pmod{11}$ and $P = (0, 7)$ and $Q = (5, 6)$. We want to compute $R = P + Q$ where $a = 1, b = 5, p = 11$. We can see that the situation if $P \neq Q$, so:

$$\lambda = (y_2 - y_1)/(x_2 - x_1) = (6 - 7)/(5 - 0) = -1/5 \pmod{11} = 10/5 \pmod{11} = 2 \pmod{11}$$
$$x_3 = \lambda^2 - x_1 - x_2 = 2^2 - 0 - 5 = -1 \pmod{11} = 10 \pmod{11}$$
$$y_3 = \lambda(x_1 - x_3) - y_1 = 2(0 - 10) - 7 = -27 \pmod{11} = 6 \pmod{11}$$
$$R = (x_3, y_3) = (10, 6)$$

Diffie and Hellman Key Exchange

Diffie and Hellman introduced the concept of public key cryptography. The mathematical "trick" of Diffie-Hellman key exchange is that it is relatively easy to compute exponents compared to computing discrete logarithms. Diffie-Hellman allows two parties — the ubiquitous Alice and Bob — to generate a secret key; they need to exchange some information over an unsecure communications channel to perform the calculation but an eavesdropper cannot determine the shared secret key based upon this information.

Alice and Bob start by agreeing on a large prime number, N. They also have to choose some number G so that G<N.

There is actually another constraint on G, namely that it must be primitive with respect to N. *Primitive* is a definition that is a little beyond the scope of our discussion but basically G is primitive to N if the set of N-1 values of G^i mod N for i = (1, N-1) are all different. As an example, 2 is not primitive to 7 because the set of powers of 2 from 1 to 6, mod 7 (i.e., 2^1 mod 7, 2^2 mod 7, ..., 2^6 mod 7) = {2,4,1,2,4,1} (2, 4, & 1 are repeated). On the other hand, 3 is primitive to 7 because the set of powers of 3 from 1 to 6, mod 7 = {3,2,6,4,5,1} (all values are unique).

Alice...	**Bob...**
1. Choose a large random number, X_A < N. This is Alice's private key.	1. Choose a large random number, X_B < N. This is Bob's private key.
2. Compute $Y_A = G^{X}{}_A$ mod N. This is Alice's public key.	2. Compute $Y_B = G^{X}{}_B$ mod N. This is Bob's public key.
3. Exchange public key with Bob.	3. Exchange public key with Alice.
4. Compute $K_A = Y_B{}^{X}{}_A$ mod N	4. Compute $K_B = Y_A{}^{X}{}_B$ mod N

Note that X_A and X_B are kept secret while Y_A and Y_B are openly shared; these are the private and public keys, respectively. Based on their own private key and the public key learned from the other party, Alice and Bob have computed their secret keys, K_A and K_B, respectively, which are equal to $G^{X_A X_B}$ mod N.

Alice...	**Bob...**
1. Choose private key; $X_A = 2$	1. Choose private key; $X_B = 3$
2. Compute public key; $Y_A = 3^2 \bmod 7 = 2$	2. Compute public key; $Y_B = 3^3 \bmod 7 = 6$
3. Exchange public key with Bob	3. Exchange public key with Alice
4. $K_A = Y_B{}^{X}{}_A$ mod N = 6^2 mod 7 = 1	4. $K_B = Y_A{}^{X}{}_B$ mod N = 2^3 mod 7 = 1

* * * * * Relax and Learn * * * * * All the Best * * * * *