## RSA (Rivest-Shamir-Adleman)

RSA (Rivest-Shamir-Adleman) is a widely-used public-key cryptosystem that is based on the mathematical concepts of prime numbers and modular arithmetic. It allows for secure communication over an insecure channel by encrypting a message with the intended recipient's public key which only the recipient is able to decrypt using their private key.

**The RSA algorithm involves the following steps:**

1. Key Generation:
   - Choose two large distinct prime numbers 'p' and 'q'
   - Calculate n = pq
   - Choose an integer e, such that $1 < e < \Phi(n)$ and $\gcd(e, \Phi(n)) = 1$
   - Calculate d, such that d  e^-1 (mod $\Phi(n)$)

   Public key: (e, n)
   Private key: (d, n)
2. Encryption:
   - Convert the message into a number 'm'
   - Apply the following rule:

   `c = m^e(mod n)`

   where c is the encrypted message (ciphertext)
3. Decryption:
   - Apply the following rule:

   `m = c^d(mod n)`

   where m is the original message (plaintext)

**The mathematical formulas used in RSA algorithm:**

- Prime Number: A prime number is a number that is only divisible by 1 and itself. Example: 2, 3, 5, 7, 11, 13, etc.
- Public key: (e, n) where e is the public exponent
  `n = pq`, p and q are two distinct prime numbers
- Private key: (d, n) where d is the private exponent

```
d  e^-1 (mod Φ(n))
Φ(n) = (p-1)(q-1)
```

- Encryption: `c = m^e(mod n)` where m is the plaintext message
  c is the ciphertext message
  e is the public exponent
  n is the product of two primes (i.e. n = pq)
- Decryption: `m = c^d(mod n)` where c is the ciphertext message
  m is the decrypted plaintext message
  d is the private exponent
  n is the product of two primes (i.e. n = pq)

**Example of JavaScript code for RSA Encryption and Decryption:**

```
// RSA Encryption
function rsaEncrypt(publicKey, plaintextMsg) {
  const [e, n] = publicKey;
  const ciphertext = BigInt(plaintextMsg) ** BigInt(e) % BigInt(n);
  return ciphertext.toString();
}

// RSA Decryption
function rsaDecrypt(privateKey, ciphertextMsg) {
  const [d, n] = privateKey;
  const plaintext = BigInt(ciphertextMsg) ** BigInt(d) % BigInt(n);
  return plaintext.toString();
}
```

**Note:**

Due to the security concerns of using small prime numbers and integer overflows in JavaScript.