```
import random

# Asal sayıları oluşturmak ve diğer hesaplamaları yapmak için yardımcı fonksiyonlar

def is_prime(num):
    if num <= 1:
        return False
    if num <= 3:
        return True
    if num % 2 == 0 or num % 3 == 0:
        return False
    i = 5
    while i * i <= num:
        if num % i == 0 or num % (i + 2) == 0:
            return False
        i += 6
    return True

def generate_prime(bits):
    while True:
        num = random.getrandbits(bits)
        if is_prime(num):
            return num

def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = extended_gcd(b % a, a)
        return (g, y - (b // a) * x, x)

def modinv(a, m):
    g, x, y = extended_gcd(a, m)
    if g != 1:
        raise Exception('Modular inverse does not exist')
    else:
        return x % m

# Anahtar çifti oluşturma

def generate_keypair(bits):
    p = generate_prime(bits)
    q = generate_prime(bits)
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537  # Genel anahtar (public key) olarak seçilen genellikle küçük bir asal sayı
    d = modinv(e, phi)  # Özel anahtar (private key) hesaplaması

    return ((n, e), (n, d))

# Şifreleme ve şifre çözme işlemleri

def encrypt(public_key, plaintext):
    n, e = public_key
    ciphertext = [pow(ord(char), e, n) for char in plaintext]
    return ciphertext

def decrypt(private_key, ciphertext):
    n, d = private_key
    plaintext = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plaintext)

# Ana program

if __name__ == '__main__':
    bits = int(input("Anahtar boyutunu (bit cinsinden) girin: "))

    public_key, private_key = generate_keypair(bits)

    print("Genel Anahtar (public key):", public_key)
    print("Özel Anahtar (private key):", private_key)

    plaintext = input("Şifrelemek istediğiniz veriyi girin: ")
    ciphertext = encrypt(public_key, plaintext)
    print("Şifrelenmiş veri:", ciphertext)

    decrypted_text = decrypt(private_key, ciphertext)
    print("Çözülmüş veri:", decrypted_text)
```

RSA (Rivest-Shamir-Adleman) şifreleme algoritması, açık anahtarlı bir şifreleme sistemidir. Bu algoritma kullanıcıların verilerini şifrelemek ve çözmek için bir çift anahtar kullanır: genel anahtar (public key) ve özel anahtar (private key). Genel anahtarla şifreleme yapılırken, özel anahtarla çözme işlemi gerçekleştirilir.

is_prime(n) Fonksiyonu:

Bu fonksiyon, bir sayının asal olup olmadığını kontrol eder. İlk olarak, sayının 1'den küçük veya eşit olup olmadığını kontrol eder ve eğer öyleyse False döndürür, çünkü 1 asal bir sayı değildir. Ardından, sayının 2 veya 3'e eşit veya küçük olup olmadığını kontrol eder ve bu durumda True döndürür, çünkü 2 ve 3 asal sayılardır. Daha sonra, sayının 2'ye veya 3'e bölünüp bölünmediğini kontrol eder ve eğer bölenler varsa False döndürür. Daha sonra, 5'ten başlayarak sayının karekköküne kadar bir döngü başlatır. Bu adım, asal sayının 6'nın katları dışında herhangi bir tam böleni olup olmadığını kontrol etmek için yapılır. 6'nın katları üzerindeki sayılar zaten daha önce kontrol edilmiştir. Döngü içinde, sayının i veya (i + 2) ile bölünüp bölünmediğini kontrol eder ve eğer bölenler varsa False döndürür. Eğer yukarıdaki koşullardan hiçbiri karşılanmazsa, sayının asal olduğunu ifade eden True döndürür.

find_primes(limit) Fonksiyonu:

Bu fonksiyon, 2'den belirtilen bir üst sınıra kadar olan asal sayıları bulur. Bir boş bir liste olan primes oluşturulur. Bir döngü aracılığıyla 2'den başlayarak limit (kullanıcı tarafından belirtilen üst sınır) dahil olacak şekilde tüm sayılar kontrol edilir. Her sayı, is_prime fonksiyonu kullanılarak asal olup olmadığı kontrol edilir. Eğer bir sayı asalsa, primes listesine eklenir. Döngü sona erdikten sonra, primes listesi asal sayıları içerir ve bu liste döndürülür.

Ana Program:

Kullanıcıdan bir üst sınır (limit) girmesi istenir. find_primes(limit) fonksiyonu kullanılarak asal sayılar bulunur ve prime_numbers adlı bir liste elde edilir.