



**Министерство науки и высшего образования Российской  
Федерации**  
**Федеральное государственное бюджетное образовательное  
учреждение высшего образования**  
**«Московский государственный технический университет  
имени Н.Э. Баумана**  
**(национальный исследовательский университет)»**  
**(МГТУ им. Н.Э. Баумана)**

---

**ФАКУЛЬТЕТ «Информатика и системы управления»**

**КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»**

**Лабораторная работа № 3**  
**по дисциплине «Защита информации»**

**Тема Реализация алгоритма шифрования с открытым ключом (RSA)**

**Студент Пермякова Е. Д.**

**Группа ИУ7-72Б**

**Преподаватели Руденкова Ю. С.**

Москва, 2025

## **ВВЕДЕНИЕ**

Целью данной работы является разработка алгоритма шифрования с открытым ключом. Шифрование и расшифровка произвольного файла.

Для достижения поставленной цели требуется решить следующие задачи:

- 1) описать алгоритм шифрования и расшифровки с открытым ключом файла;
- 2) реализовать в виде программы алгоритм шифрования и расшифровки с открытым ключом файла;

## **1 Теоретическая часть**

Асимметричное шифрование — это метод шифрования данных, при котором используются два ключа: открытый и закрытый. Ключи связаны математически: то, что зашифровано одним, может быть расшифровано только другим.

Открытый (публичный) ключ используется для шифрования данных, может быть известен всем.

Закрытый (приватный) ключ используется для расшифровки данных, зашифрованных открытым ключом.

## 2 Описание алгоритма шифрования и расшифровки с открытым ключом файла

На рисунке 2.1 приведена схема генерации ключевой пары.

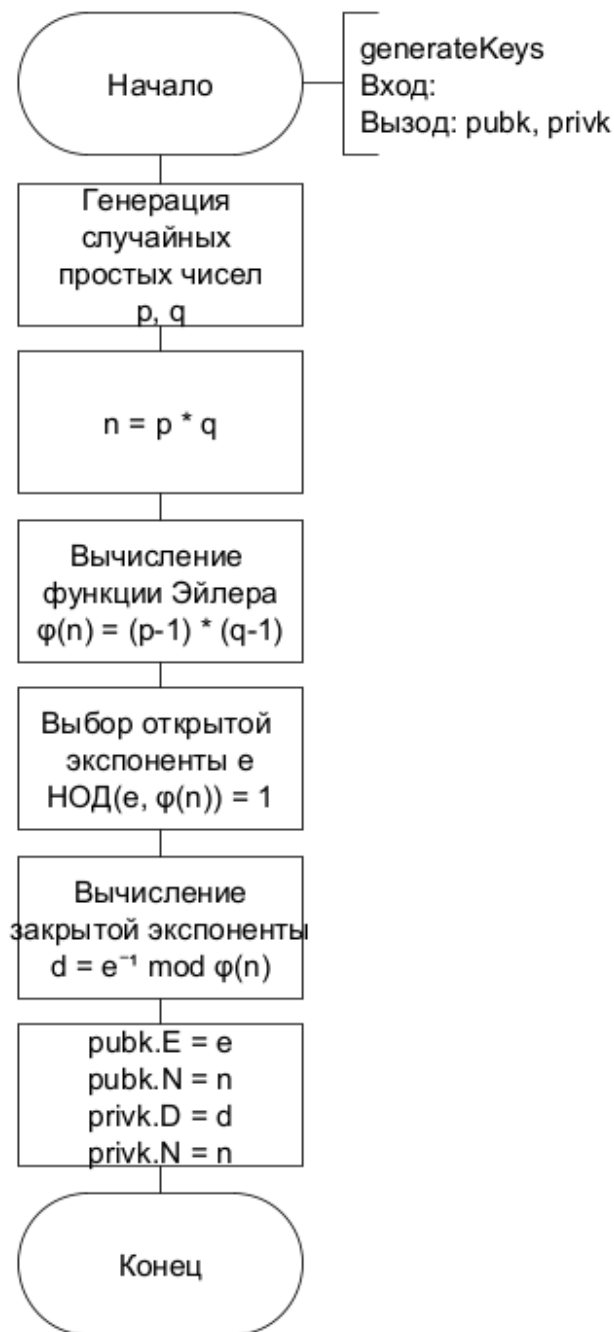


Рисунок 2.1 – Схема генерации ключевой пары

На рисунке 2.2 приведена схема алгоритма шифрования с открытым ключом.

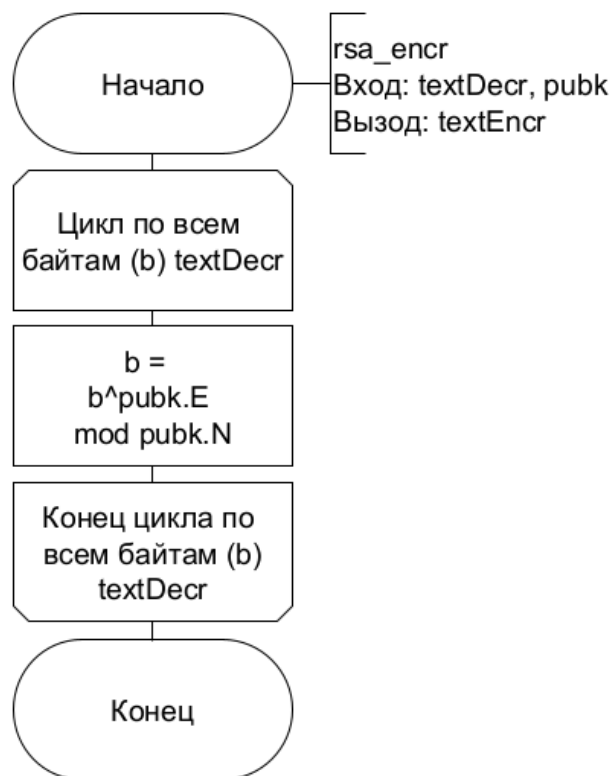


Рисунок 2.2 – Схема алгоритма шифрования с открытым ключом

На рисунке 2.3 приведена схема алгоритма расшифровки с открытым ключом.

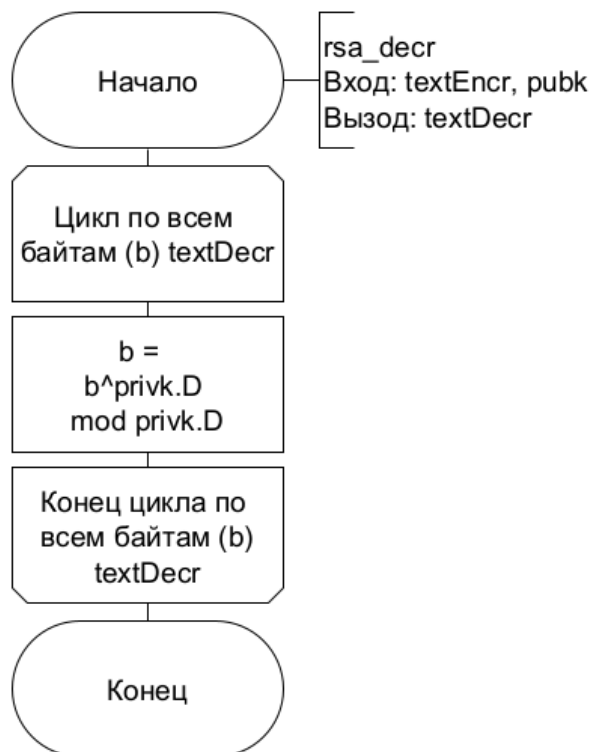


Рисунок 2.3 – Схема алгоритма расшифровки с открытым ключом

### 3 Пример работы алгоритма шифрования и расшифровки с открытым ключом файла

На рисунке 3.1 приведен пример работы алгоритма шифрования и расшифровки с открытым ключом файла.

```
kathrine@Viva:~/vuz/InfoSec/is_3$ go run ./src/main.go

Выберите действие:
1. Сгенерировать новые ключи
2. Зашифровать файл
3. Расшифровать файл
4. Выход
Ваш выбор: 1
Новые ключи сгенерированы и сохранены
Ваш выбор: 2
Введите путь к файлу для шифрования: ./data/input.txt
Файл зашифрован и сохранен как ./data/input.txt.encrypted
Ваш выбор: 3
Введите путь к зашифрованному файлу: ./data/input.txt.encrypted
Файл расшифрован и сохранен как ./data/input.txt.encrypted.decrypted
Ваш выбор: 4
Выход...
kathrine@Viva:~/vuz/InfoSec/is_3$ cat ./data/input.txt
----- ABC - abc 12345.... 678 |||| touch
input.txt
kathrine@Viva:~/vuz/InfoSec/is_3$ cat ./data/input.txt.encrypted
#####i#####W#####
#####o#####n#####T#####i#####p#####M#####^#####i#####t#####t#####t#####ti#####H5[#####o#####p#####i#####6#####\#####[#####6#####kathrine@Viva:~/vuz/
InfoSec/is_3$ cat ./data/input.txt.encrypted.decrypted
----- ABC - abc 12345.... 678 |||| touch
input.txt
kathrine@Viva:~/vuz/InfoSec/is_3$
```

Рисунок 3.1 – Пример работы алгоритма шифрования и расшифровки с открытым ключом файла

На рисунке 3.2 приведен пример работы алгоритма шифрования и расшифровки с открытым ключом файла при попытке расшифровки файла сторонним ключом.

```
kathrine@Viva:~/vuz/InfoSec/is_3$ go run ./src/main.go

Выберите действие:
1. Сгенерировать новые ключи
2. Зашифровать файл
3. Расшифровать файл
4. Выход
Ваш выбор: 1
Новые ключи сгенерированы и сохранены
Ваш выбор: 2
Введите путь к файлу для шифрования: ./data/input.txt
Файл зашифрован и сохранен как ./data/input.txt.encrypted
Ваш выбор: 1
Новые ключи сгенерированы и сохранены
Ваш выбор: 3
Введите путь к зашифрованному файлу: ./data/input.txt.encrypted
Файл расшифрован и сохранен как ./data/input.txt.encrypted.decrypted
Ваш выбор: 4
Выход...
kathrine@Viva:~/vuz/InfoSec/is_3$ cat ./data/input.txt
----- ABC - abc 12345.... 678 |||| touch
input.txt
kathrine@Viva:~/vuz/InfoSec/is_3$ cat ./data/input.txt.encrypted
7#*77M4 "779
kathrine@Viva:~/vuz/InfoSec/is_3$ 7-==@F7
kathrine@Viva:~/vuz/InfoSec/is_3$ cat ./data/input.txt.encrypted.decrypted
FFFFF1FgZ,$piiiiwmiiNZO0s%Niiisikathrine@Viva:~/vuz/InfoSec/is_3$
```

Рисунок 3.2 – Пример работы алгоритма шифрования и расшифровки с открытым ключом файла при попытке расшифровки файла сторонним ключом

## 4 Реализация алгоритма шифрования и расшифровки с открытым ключом файла

В качестве средства реализации алгоритма шифрования и расшифровки с открытым ключом файла был выбран язык Go.

```
type RSA struct {
    publicKey  *PublicKey
    privateKey *PrivateKey
    keysFile   string
}

type PublicKey struct {
    E *big.Int
    N *big.Int
}

type PrivateKey struct {
    D *big.Int
    N *big.Int
}

func NewRSA(loadKeys bool, keysFileName string) (*RSA, error) {
    rsa := &RSA{
        keysFile: keysFileName,
    }

    if loadKeys && FileExists(rsa.keysFile) {
        rsa.loadKeys()
    } else {
        rsa.generateKeys()
        err := rsa.saveKeys()
        if err != nil {
            return nil, fmt.Errorf("NewRSA: %w", err)
        }
    }

    return rsa, nil
}

func (r *RSA) generatePrime(min, max int64) *big.Int {
    for {
```



```

        n, err := rand.Int(rand.Reader, big.NewInt(max-min+1))
        if err != nil {
            panic(err)
        }
        n.Add(n, big.NewInt(min))

        if n.ProbablyPrime(20) {
            return n
        }
    }
}

func (r *RSA) generateKeys() {
    p := r.generatePrime(100, 300)
    q := r.generatePrime(100, 300)
    for p.Cmp(q) == 0 {
        q = r.generatePrime(100, 300)
    }

    n := new(big.Int).Mul(p, q)
    phi := new(big.Int).Mul(
        new(big.Int).Sub(p, big.NewInt(1)),
        new(big.Int).Sub(q, big.NewInt(1)),
    )

    e := big.NewInt(65537)
    for new(big.Int).GCD(nil, nil, e, phi).Cmp(big.NewInt(1))
        != 0 {
        e, _ = rand.Int(rand.Reader, new(big.Int).Sub(phi, big.
            NewInt(3)))
        e.Add(e, big.NewInt(3))
    }

    d := new(big.Int).ModInverse(e, phi)

    r.publicKey = &PublicKey{E: e, N: n}
    r.privateKey = &PrivateKey{D: d, N: n}
}

func (r *RSA) saveKeys() error {
    keys := map[string]interface{}{

```

```

        "public_key": map[string]string{
            "e": r.publicKey.E.String(),
            "n": r.publicKey.N.String(),
        },
        "private_key": map[string]string{
            "d": r.privateKey.D.String(),
            "n": r.privateKey.N.String(),
        },
    }

    jsonData, err := json.MarshalIndent(keys, "", "  ")
    if err != nil {
        return fmt.Errorf("saveKeys: %w", err)
    }

    err = os.WriteFile(r.keysFile, jsonData, 0644)
    if err != nil {
        return fmt.Errorf("saveKeys: %w", err)
    }
    return nil
}

func (r *RSA) loadKeys() {
    data, err := os.ReadFile(r.keysFile)
    if err != nil {
        panic(err)
    }

    var keys map[string]interface{}
    err = json.Unmarshal(data, &keys)
    if err != nil {
        panic(err)
    }

    publicKey := keys["public_key"].(map[string]interface{})
    e := new(big.Int)
    e.SetString(publicKey["e"].(string), 10)
    n := new(big.Int)
    n.SetString(publicKey["n"].(string), 10)
    r.publicKey = &PublicKey{E: e, N: n}
}

```

```

privateKey := keys["private_key"].(map[string]interface{})
d := new(big.Int)
d.SetString(privateKey["d"].(string), 10)
n = new(big.Int)
n.SetString(privateKey["n"].(string), 10)
r.privateKey = &PrivateKey{D: d, N: n}
}

func (r *RSA) encryptByte(b byte) *big.Int {
    byteInt := big.NewInt(int64(b))
    return new(big.Int).Exp(byteInt, r.publicKey.E, r.publicKey.N)
}

func (r *RSA) decryptByte(encrypted *big.Int) byte {
    decrypted := new(big.Int).Exp(encrypted, r.privateKey.D, r.privateKey.N)
    return byte(decrypted.Int64())
}

func (r *RSA) EncryptFile(inputPath, outputPath string) error {
    inputFile, err := os.Open(inputPath)
    if err != nil {
        return err
    }
    defer inputFile.Close()

    outputFile, err := os.Create(outputPath)
    if err != nil {
        return err
    }
    defer outputFile.Close()

    buffer := make([]byte, 1)
    for {
        _, err := inputFile.Read(buffer)
        if err == io.EOF {
            break
        }
        if err != nil {
            return err
        }
    }
}

```

```

    }

    encrypted := r.encryptByte(buffer[0])
    encryptedBytes := encrypted.Bytes()

    length := byte(len(encryptedBytes))
    if _, err := outputFile.Write([]byte{length}); err !=
        nil {
        return err
    }
    if _, err := outputFile.Write(encryptedBytes); err !=
        nil {
        return err
    }
}

return nil
}

func (r *RSA) DecryptFile(inputPath, outputPath string) error {
    inputFile, err := os.Open(inputPath)
    if err != nil {
        return err
    }
    defer inputFile.Close()

    outputFile, err := os.Create(outputPath)
    if err != nil {
        return err
    }
    defer outputFile.Close()

    for {
        lengthBuf := make([]byte, 1)
        _, err := inputFile.Read(lengthBuf)
        if err == io.EOF {
            break
        }
        if err != nil {
            return err
        }
    }
}

```

```

length := int(lengthBuf[0])
if length == 0 {
    continue
}

encryptedBytes := make([]byte, length)
_, err = inputFile.Read(encryptedBytes)
if err != nil {
    return err
}

encrypted := new(big.Int).SetBytes(encryptedBytes)
decrypted := r.decryptByte(encrypted)

if _, err := outputFile.Write([]byte{decrypted}); err
    != nil {
        return err
    }
}

return nil
}

```

Листинг 4.1 – Реализация алгоритма шифрования и расшифровки с открытым ключом файла

## **ЗАКЛЮЧЕНИЕ**

В ходе лабораторной работы был реализован алгоритм шифрования с открытым ключом.

В процессе выполнения данной работы были выполнены все задачи:

- 1) описать алгоритм шифрования и расшифровки с открытым ключом файла;
- 2) реализовать виде программы алгоритм шифрования и расшифровки с открытым ключом файла;