Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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 № 4
по дисциплине «Защита информация»

Тема Реализация алгоритма шифрования с открытым ключом (RSA).

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва, 2025

Содержание

ВВЕДЕНИЕ

Цель работы: Разработка алгоритма шифрования с открытым ключом. Шифрование и расшифровка архивных файлов.

Задачи:

1) проанализировать алгоритм шифрования архивного файла с открытым ключом;
2) проанализировать алгоритм расшифровки архивного файла с открытым ключом;
3) реализовать описанный алгоритм шифрования и расшифровку;
4) провести шифрование и расшифровку архивного файла;

1 Аналитическая часть

1.1 Определения

Ассиметричное шифрование – метод шифрования данных, при котором используются два ключа:

— Открытый (публичный) — для шифрования данных, может передаваться по незащищённым каналам.

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

Оба ключа генерируются стороной, которая будет принимать сообщения.

Закрытый ключ хранится в секрете у того, кто будет принимать зашифрованные сообщения. Открытый ключ передаётся всем, кто будет отправлять зашифрованные сообщения. Пара ключей устроена так, что сообщение зашифрованное открытым ключом может быть расшифровано только с помощью парного ей закрытого ключа, при этом алгоритмы ассиметричного шифрования строятся на том, что вычислительно сложно по открытому ключу подобрать соответствующий ему открытый.

1.2 Алгоритм RSA

Алгоритм RSA – ассиметричный алгоритм шифрования, в основе которого лежит сложность факторизации двух простых чисел. То есть сложность подбора простых множителей по заданному произведению. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложение числа на простые множители.

1.2.1 Алгоритм создания открытого и закрытого ключей

Алгоритм создания ключей в системе RSA имеет следующий вид:

1) выбираются два различных случайных простых числа p и q заданного размера в битах;
2) вычисляется их произведение $n = p * q$, которое называется модулем;
3) вычисляется значение функции Эйлера от числа n: $\phi(n) = (p-1)*(q-$

1);

4) выбирается число $e : 1 < e < \phi(n)$, взаимно простое со значением $\phi(n)$. Число e называется открытой экспонентной;

5) вычисляется число d, мультипликативно обратное к числу e по модулю n;

6) пара (e, n) называется открытым ключом, а пара (d, n) – закрытым.

1.2.2 Алгоритм шифрования RSA

Пусть архивный файл состоит из m байт, а размер модуля ключей n байт. Тогда алгоритм шифрования имеет вид:

1) Делим архивный файл на $k = m/n$ блоков размер n. Последний блок дополнить 0 до размера n.

2) Цикл по блокам архивного файла:

   2.1) представить блок как беззнаковое число r;

   2.2) зашифровать число с использованием открытого ключа (e, n) : $c = r^e \ mod \ n$;

3) Соединить зашифрованные блоки для получения зашифрованного архива.

1.2.3 Алгоритм расшифровки RSA

Пусть зашифрованный архивный файл состоит из m байт, а размер модуля ключей n байт. Тогда алгоритм шифрования имеет вид:

1) Делим зашифрованный архивный файл на $k = m/n$ блоков размер n. Последний блок дополнить 0 до размера n.

2) Цикл по блокам зашифрованного архивного файла:

   2.1) представить зашифрованный блок как беззнаковое число c;

   2.2) зашифровать число с использованием закрытого ключа (d, n) : $r = c^d \ mod \ n$;

3) Соединить расшифрованные блоки для получения исходного архивного файла.

2 Технологическая часть

2.1 Средства реализации

В качестве языка программирования для программной реализации алгоритма RSA был выбран Go.

2.2 Код разработанной программы

На листингах 2.1- 2.3 представлена разработанная программа.

Листинг 2.1 — Код создания ключей RSA

```
package rsa

import (
  "crypto/rand"
  "fmt"
  "math/big"
)

type RSA struct {
  Module          *big.Int
  PublicExponent  *big.Int
  PrivateExponent *big.Int
}

func NewRSA(bitSize int) (*RSA, error) {
  if (bitSize % 2) != 0 {
    return nil, fmt.Errorf("bitSize must be even")
  }

  p, err := rand.Prime(rand.Reader, bitSize/2)
  if err != nil {
    return nil, err
  }

  q, err := rand.Prime(rand.Reader, bitSize/2)
  if err != nil {
    return nil, err
  }
```

```
  for p.Cmp(q) == 0 {
    q, err = rand.Prime(rand.Reader, bitSize/2)
    if err != nil {
      return nil, err
    }
  }

  n := big.NewInt(0).Mul(p, q)
  phi := big.NewInt(0).Mul(big.NewInt(0).Sub(p, big.NewInt(1)), big.
     NewInt(0).Sub(q, big.NewInt(1)))

  e := big.NewInt(0).SetInt64(65537)

  // Check if e and phi are coprime
  gcd := big.NewInt(0).GCD(nil, nil, e, phi)
  if gcd.Cmp(big.NewInt(1)) != 0 {
    e := big.NewInt(17)
    gcd := big.NewInt(0).GCD(nil, nil, e, phi)
    for gcd.Cmp(big.NewInt(1)) != 0 {
      e = big.NewInt(0).Add(e, big.NewInt(2))
      gcd = big.NewInt(0).GCD(nil, nil, e, phi)
    }
  }

  d := big.NewInt(0).ModInverse(e, phi)
  return &RSA{
    Module:          n,
    PublicExponent:  e,
    PrivateExponent: d,
  }, nil
}

func (r *RSA) GetPublicKey() *RSAPublicKey {
  return NewRSAPublicKey(r.Module, r.PublicExponent)
}

func (r *RSA) GetPrivateKey() *RSAPrivateKey {
  return NewRSAPrivateKey(r.Module, r.PrivateExponent)
}
```

Листинг 2.2 — Код шифрования открытым ключом

```
package rsa

import (
  "fmt"
  "io"
  "math/big"
)

type RSAPublicKey struct {
  Module         *big.Int
  PublicExponent *big.Int
}

func NewRSAPublicKey(n *big.Int, e *big.Int) *RSAPublicKey {
  return &RSAPublicKey{
    Module:         n,
    PublicExponent: e,
  }
}

func (r *RSAPublicKey) Encrypt(message []byte) ([]byte, error) {
  blockSize := (r.Module.BitLen()) / 8
  if blockSize <= 0 {
    return nil, fmt.Errorf("blockSize must be greater than 0")
  }
  result := make([]byte, 0, len(message))
  for i := 0; i < len(message); i += blockSize {
    end := i + blockSize
    if end > len(message) {
      end = len(message)
    }
    block := message[i:end]

    num := big.NewInt(0).SetBytes(block)
    if num.Cmp(r.Module) != -1 {
      return nil, fmt.Errorf("block is bigger than modulus")
    }
    encrypted := big.NewInt(0).Exp(num, r.PublicExponent, r.Module)
    encryptedBytes := encrypted.Bytes()
```

```
    if len(encryptedBytes) > blockSize {
      return nil, fmt.Errorf("encrypted block is bigger than
        blockSize")
    }
    if len(encryptedBytes) < blockSize {
      encryptedBytes = append(make([]byte, blockSize-len(
        encryptedBytes)), encryptedBytes...)
    }
    result = append(result, encryptedBytes...)
  }

  return result, nil
}

func (r *RSAPublicKey) Dump(reader io.Writer) error {
  _, err := fmt.Fprintf(reader, "Public Key:\n")
  if err != nil {
    return err
  }
  _, err = fmt.Fprintf(reader, "%s\n", r.Module.String())
  if err != nil {
    return err
  }
  _, err = fmt.Fprintf(reader, "%s\n", r.PublicExponent.String())
  if err != nil {
    return err
  }

  return nil
}

func LoadPublicRSA(reader io.Reader) (*RSAPublicKey, error) {
  var moduleStr string
  var publicExponentStr string

  _, err := fmt.Fscanf(reader, "Public Key:\n")
  if err != nil {
    return nil, err
  }
  _, err = fmt.Fscanf(reader, "%s\n", &moduleStr)
```

```
  if err != nil {
    return nil, err
  }
  _, err = fmt.Fscanf(reader, "%s\n", &publicExponentStr)
  if err != nil {
    return nil, err
  }
  var ok bool
  var n, p *big.Int
  n, ok = big.NewInt(0).SetString(moduleStr, 10)
  if !ok {
    return nil, fmt.Errorf("invalid module")
  }
  p, ok = big.NewInt(0).SetString(publicExponentStr, 10)
  if !ok {
    return nil, fmt.Errorf("invalid publicExponent")
  }
  return NewRSAPublicKey(n, p), nil
}
```

Листинг 2.3 — Код расшифроваки закрытым ключом

```
package rsa

import (
  "fmt"
  "io"
  "math/big"
)

type RSAPrivateKey struct {
  Module          *big.Int
  PrivateExponent *big.Int
}

func NewRSAPrivateKey(n *big.Int, d *big.Int) *RSAPrivateKey {
  return &RSAPrivateKey{
    Module:          n,
    PrivateExponent: d,
  }
}
```

```
func (r *RSAPrivateKey) Decrypt(message []byte) ([]byte, error) {
  blockSize := (r.Module.BitLen()) / 8
  if blockSize <= 0 {
    return nil, fmt.Errorf("blockSize must be greater than 0")
  }
  result := make([]byte, 0, len(message))
  for i := 0; i < len(message); i += blockSize {
    end := i + blockSize
    if end > len(message) {
      end = len(message)
    }
    block := message[i:end]

    num := big.NewInt(0).SetBytes(block)
    decrypted := big.NewInt(0).Exp(num, r.PrivateExponent, r.Module)
    decryptedBytes := decrypted.Bytes()

    result = append(result, decryptedBytes...)
  }

  return result, nil
}

func (r *RSAPrivateKey) Dump(reader io.Writer) error {
  _, err := fmt.Fprintf(reader, "Private Key:\n")
  if err != nil {
    return err
  }
  _, err = fmt.Fprintf(reader, "%s\n", r.Module.String())
  if err != nil {
    return err
  }
  _, err = fmt.Fprintf(reader, "%s\n", r.PrivateExponent.String())
  if err != nil {
    return err
  }

  return nil
}
```

```
func Load(reader io.Reader) (*RSAPrivateKey, error) {
  var moduleStr string
  var privateExponentStr string

  _, err := fmt.Fscanf(reader, "Private Key:\n")
  if err != nil {
    return nil, err
  }
  _, err = fmt.Fscanf(reader, "%s\n", &moduleStr)
  if err != nil {
    return nil, err
  }
  _, err = fmt.Fscanf(reader, "%s\n", &privateExponentStr)
  if err != nil {
    return nil, err
  }
  var ok bool
  var n, q *big.Int
  n, ok = big.NewInt(0).SetString(moduleStr, 10)
  if !ok {
    return nil, fmt.Errorf("invalid module")
  }
  q, ok = big.NewInt(0).SetString(privateExponentStr, 10)
  if !ok {
    return nil, fmt.Errorf("invalid privateExponent")
  }

  return NewRSAPrivateKey(n, q), nil
}
```

2.3 Пример работы программы

Разработанное ПО представляет из себя 3 программы с интерфейсом командной строки.

Первая программа создаёт открытый и закрытый ключи. Она принимает три аргумента:

— **-b** – размер модуля генерируемых ключей в битах;

— **-pri** – имя файла, куда поместить закрытый ключ;

— **-pub** – имя файла, куда поместить открытый ключ.

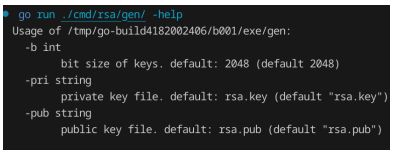

Рисунок 2.1 — Описание аргументов программы генерации ключа

Вторая программа использует открытый ключ для шифровки произвольного файла. Она принимает три аргумента:

— **-input** – имя файла с открытым текстом;

— **-output** – имя файла, куда сохранить зашифрованный текст;

— **-key** – имя файла с открытым ключом.

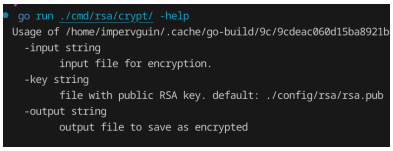

Рисунок 2.2 — Описание аргументов программы шифрования

Третья программа использует закрытый ключ для расшифровки файла. Она принимает три аргумента:

— **-input** – имя файла с зашифрованным текстом;

— **-output** – имя файла, куда сохранить расшифрованный текст;

— **-key** – имя файла с закрытым ключом.

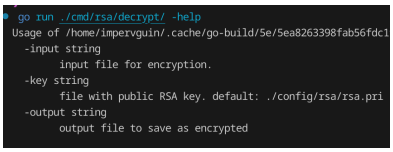

Рисунок 2.3 — Описание аргументов программы расшифровки

На рисунке 2.4 представлен запуск программ шифрования и расшифровки архивного файла.

На рисунках 2.5- 2.7 представлены попытки открытия исходного, зашифрованного и расшифрованного архива.

```
~/Projects/InfoSecurity main*
❯
go run ./cmd/rsa/crypt/ -input code.zip -output code.enc.zip

~/Projects/InfoSecurity main*
❯
go run ./cmd/rsa/decrypt/ -input code.enc.zip -output code.dec.zip
```

Рисунок 2.4 — Шифрование и расшифровка архивного файла

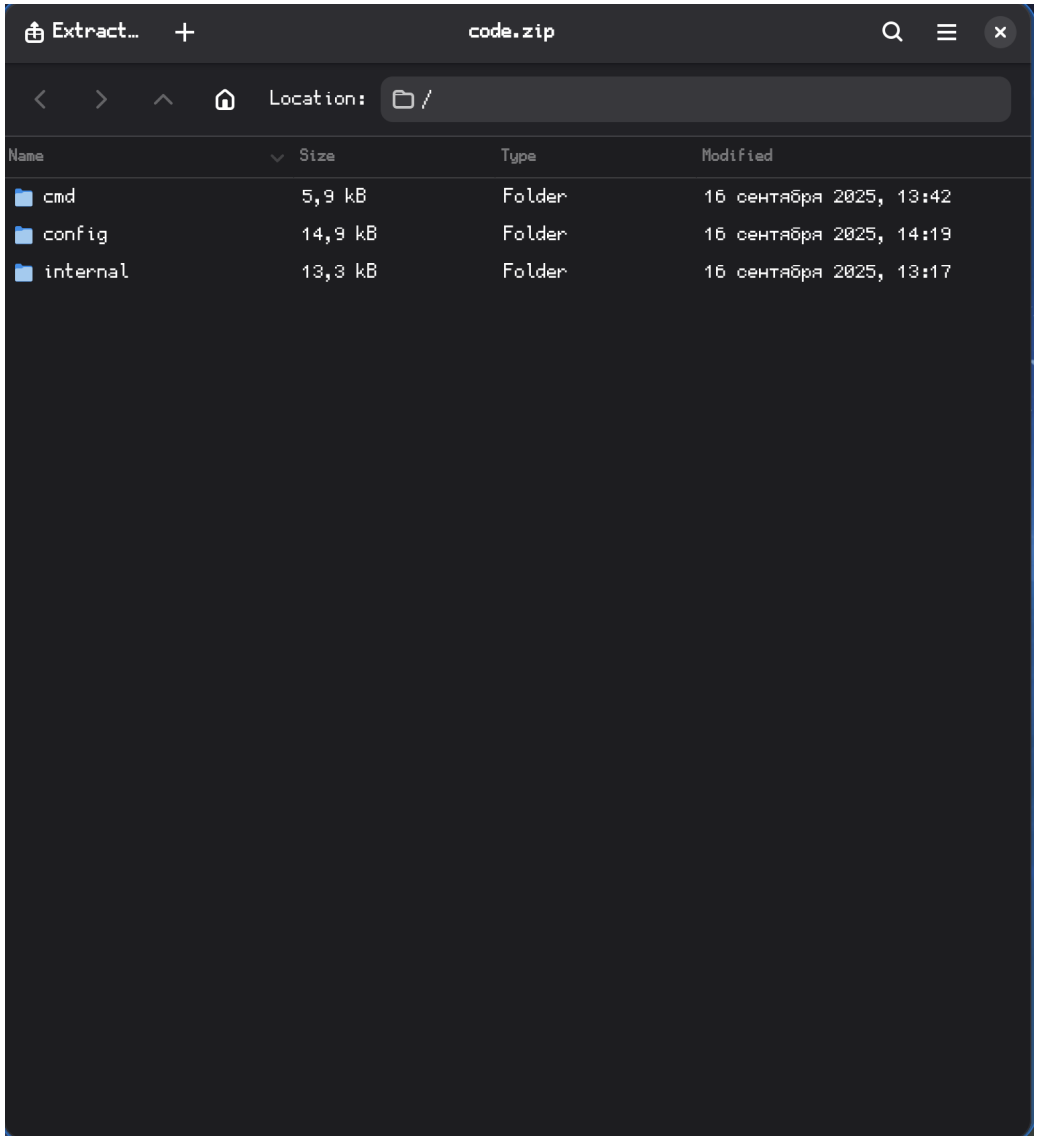

Рисунок 2.5 — Исходный архивный файл

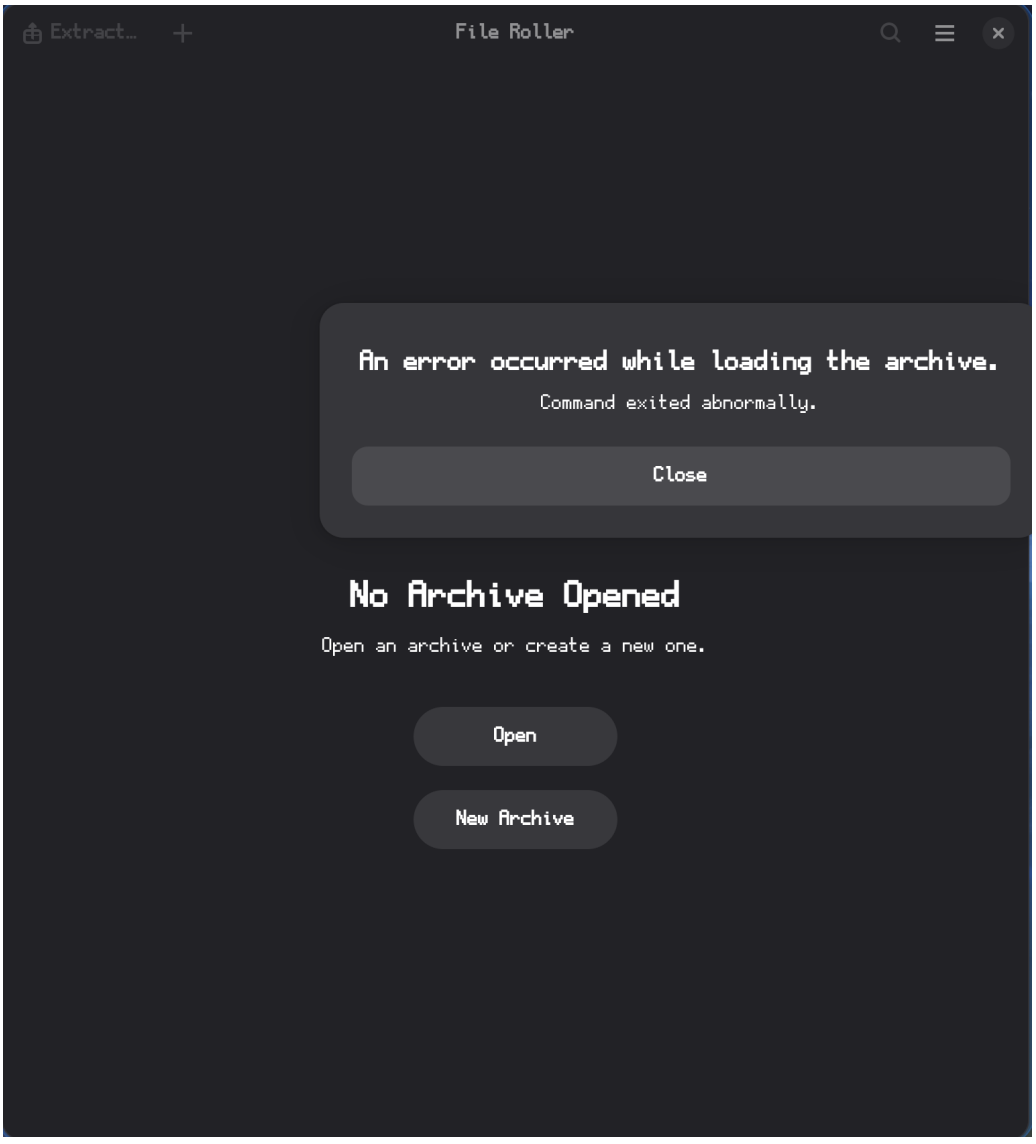

Рисунок 2.6 — Зашифрованный архивный файл

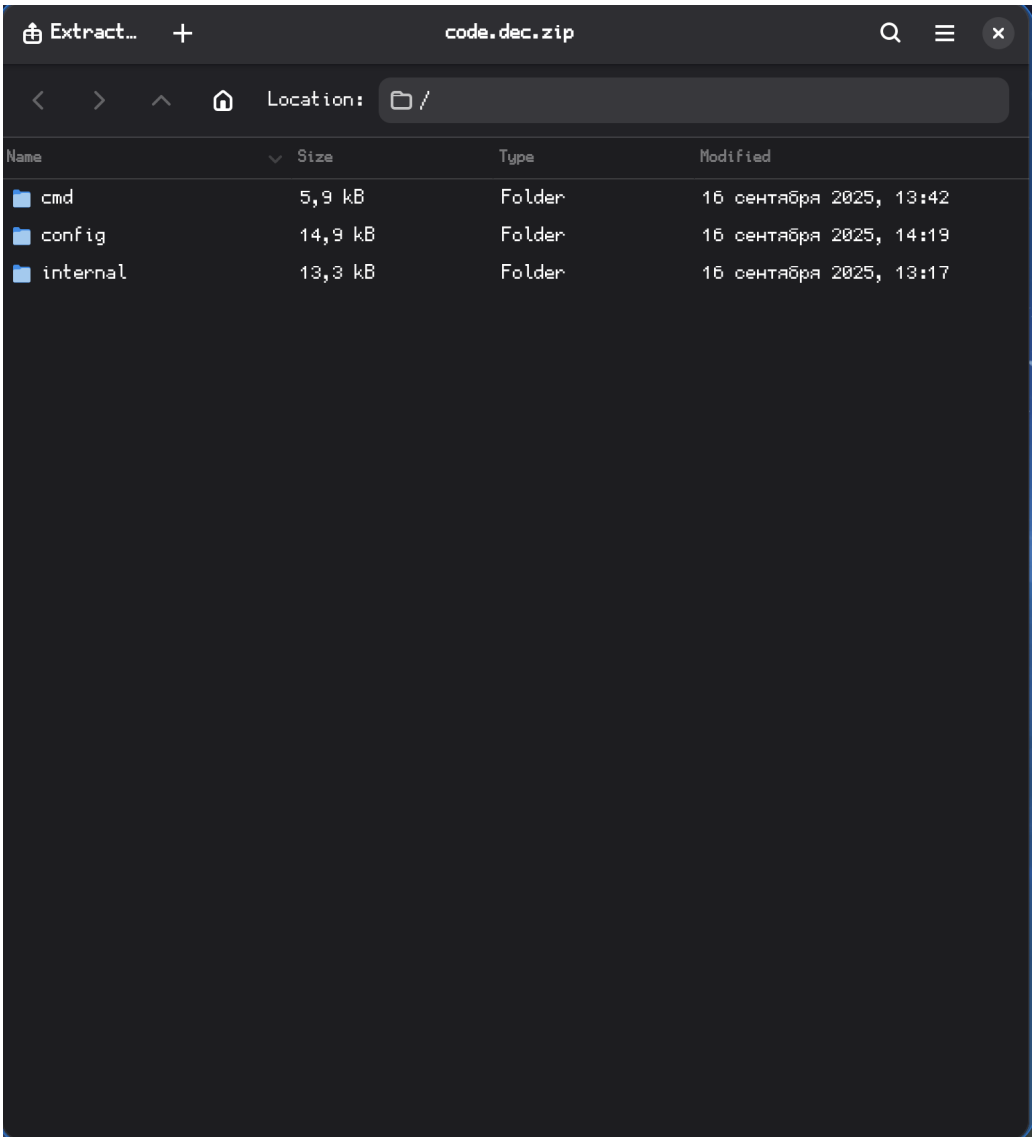

Рисунок 2.7 — Расшифрованный архивный файл

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы была разработана программная реализация алгоритма RSA.

Были выполнены следующие задачи:

1) проанализирован принцип работы алгоритма RSA;
2) описаны алгоритмы создания ключей, шифровки и расшифровки с помощью ключей;
3) реализованы описанные алгоритмы шифрования и расшифровки;
4) проведено шифрование и расшифровка архивного файла;

Все поставленные цели и задачи были выполнены.