Лабораторная работа № 1

Режимы шифрования

Задания лабораторной работы

Задание 1. Реализация дополнения блока до нужной длины по стандарту PKCS7.

По стандарту PKCS7 каждый блок открытого текста дополняется байтами, значением которых является количество недостающих байт до длины блока. Пусть N=20 – длина блока. Тогда при открытом тексте

```
YELLOW SUBMARINE
```

длинной L = 16, количество недостающих байт равно N – L = 20 – 16 = 4. Следовательно, необходимо дополнить строку байтами со значением 04. Итоговая строка:

```
YELLOW SUBMARINE\x04\x04\x04\x04
```

где “\x04” – символическое обозначение байта со значением 04. *То есть, это не строка “\x04”, а байт со значением 04.* Реализуйте функцию, принимающую 2 параметра – строку и размер блока, выполняющую дополнение блока по стандарту PKCS7.

Задание 2. Проверка дополнения PKCS7

Необходимо написать функцию, которая проверяет правильным ли образом составлено PKCS7 дополнение из задания 1 и если правильно, отрезает из строки лишние байты. Например, строка:

```
ICE ICE BABY\x04\x04\x04\x04
```

имеет правильное дополнение, и результат работы функции должен выглядеть как

```
ICE ICE BABY
```

А строки

```
ICE ICE BABY\x05\x05\x05\x05
```

и

```
ICE ICE BABY\x01\x02\x03\x04
```

имеют неправильные дополнения, следовательно, в такой ситуации реализованная функция должна выдавать исключение.

Задание 3. Атака на CBC режим. Bit-flipping

1. Сгенерируем случайный ключ для AES. Далее мы будем шифровать в режиме CBC с этим ключом.
2. Реализуем функцию (Функция 1), которая будет принимать на вход строку и добавлять в

начало строку

```
comment1=cooking%20MCs;userdata=
```

и в конец строку

```
comment2=%20like%20a%20pound%20of%20bacon
```

Так же должна присутствовать фильтрация входной строки на символы *“=” “;”*.
Далее функция полностью шифрует в режиме AES-128-CBC достроенную строку.

3. Задача – необходимо изменить шифротекст (результат Функции 1), не зная ключа, таким образом, чтобы при расшифровке там находилась подстрока

```
admin=true
```

Реализуем функцию, которая будет расшифровывать получившуюся строку и искать в ней подстроку

```
admin=true
```

CBC имеет следующие особенности:
1. При изменении 1 бита в текущем блоке, расшифрованный блок сильно изменяется.
2. В следующем блоке расшифрованного текста возникает ошибка в 1 бит на соответствующей позиции.

Дополнительную информацию по этой теме можно найти в Приложении 1.

Задание 4. Функция детектирования режима ECB (функция оракул).
Как вы знаете, при режиме ECB каждый блок открытого текста шифруется отдельно от остальных. То есть одинаковые блоки открытого текста перейдут в одинаковые блоки шифротекста, независимо от номера блока. В других режимах шифрования (например в CBC) ситуация другая. Каждый следующий блок шифротекста зависит от результата шифрования предыдущего. Таким образом, можно с легкостью определить, какой режим шифрования используется в данном случае.

1. Реализуйте функцию, которая шифрует данные на случайном ключе, то есть функция генерирует случайные 16 байт ключа и шифрует на них, используя AES-128 в режимах ECB, CBC.
Выглядеть результат будет как-то так:

```
encryption_oracle(inputStr) => [MEANINGLESS JIBBER JABBER]
```

2. Сделайте так, чтобы функция шифровала входную строку с вероятностью 50% в режиме ECB, и с вероятностью 50 % в режиме CBC.

3. В теле функции перед шифрованием добавьте к inputStr случайное количество (5-10) байт в начало строки и в конец.

4. Реализуйте функцию, которая будет детектировать в каком режиме было зашифровано

ваше сообщение.

Задание 5. Побайтовое дешифрование режима ECB или почему не стоит пользоваться ECB. В данном задании будет показано почему никогда не стоит пользоваться режимом ECB. Дело в том, что имея доступ к функции шифрования, и даже не имея ключа можно достаточно просто получить шифрованный текст простым перебором в блоке всех возможных вариантов открытых текстов. Далее будет дано объяснение как это сделать.

1. Реализуйте функцию, которая шифрует строку в режиме ECB, используя случайный ключ. При этом ключ генерируется 1 раз. То есть, при последующих вызовах этой функции ключ не должен изменяться.

2. Теперь сделайте так, чтобы при передаче в функцию, описанную выше, к входной строке прибавлялась строка

```
unknownStrBase64="Um9sbGluJyBpbiBteSA1LjAKV2l0aCBteSByYWctdG9wIGRvd24gc28gbXkga
GFpciBjYW4gYmxvdwpUaGUgZ2lybGllcyBvbiBzdGFuZGJ5IHdhdmluZyBqdXN0IHRvIHNheSBoaQp
EaWQgeW91IHN0b3A/IE5vLCBJIGp1c3QgZHJvdmUgYnkK"
```

Это строка в Base64. Предварительно нужно автоматически декодировать эту строку. **НЕ СМОТРИТЕ РЕЗУЛЬТАТ ДЕКОДИРОВАНИЯ.** Идея в том, что вы не должны знать ее содержание.

3. Итак, у Вас есть функция следующего вида

```
AES_128_ECB(your_string || decodeBase64(unknownStrBase64), random-key)
```

Теперь нужно написать функцию которая с помощью вашей входной строки (your_string) расшифровывает неизвестную строку (unknownStrBase64)

4. Как же это сделать...

а) Сначала необходимо детектировать длину блока, хотя мы это и знаем, все равно это будет полезно: Для этого в your_string передается строка состоящая из одинаковых байт: “A” “AA” “AAA” и так далее... пока не поймем какой будет размер блока (2 блока шифротекста будут одинаковыми).

б) Теперь, когда мы знаем размер блока, необходимо подать в переменную your_string строку на один байт(символ) короче, чем блок. Например, если длина блока 8 байт, то мы подаем строку “AAAAAAA” длиной 7. Как вы должны догадаться, на последнее место встанет первый символ неизвестной строки (unknownStr).

в) Теперь нужно перебрать все возможные варианты этого последнего байта и сравнить с выходным результатом из пункта б). Для этого нужно осуществить перебор по всем возможным символам по размеру блока. Пусть размер блока 8, тогда входная строка должна быть “AAAAAAAA” “AAAAAAAB” “AAAAAAAC” и так далее...

г) Итак, мы получили первый байт (символ) неизвестной строки, теперь необходимо сделать тоже самое для остальной строки. Пусть первый символ будет “E”
Так для второй строки, при размере блока 8, перебор всех возможных значений последнего байта будет осуществляется по строке “AAAAAAE*”, то есть

“AAAAAAEA” “AAAAAAEB” и так далее.

д) Оцените трудоемкость дешифрования сообщения.

Задание 6. Подмена шифротекста, зашифрованного в режиме ECB

1. Первое, что нужно сделать, это написать функцию, которая будет переводить строку в объект, пусть дана строка в формате

```
foo=bar&baz=qux&zap=zazzle
```

Данную строку необходимо преобразовать в объект, то есть на выходе получить объект:

```
{
  foo: 'bar',
  baz: 'qux',
  zap: 'zazzle'
 }
```

Можно использовать JSON объект.

2. Далее напишем функцию profile_for(email), которая:

а) принимает на вход строку с почтой;

б) генерирует объект профиля;

в) кодирует его в строку и возвращает пользователю.

Например:

```
profile_for("foo@bar.com")
{
  email: 'foo@bar.com',
  uid: 10,
  role: 'user'
 }
```

кодируем его в строку:

```
email=foo@bar.com&uid=10&role=user
```

Заметим, что в строку email нельзя передавать символы & = и прочие символы, которые не содержатся в почтовом адресе.

3. Теперь зашифруем получившуюся строку в режиме ECB с неизвестным, далее неизменным, ключом.

4. Теперь получившийся шифротекст нужно изменить так, чтобы при расшифровке получилось

```
{
  email: ' foo@bar.com,
  uid: 10,
  role: 'admin'
 }
```

то есть по сути получить роль админа. При этом мы имеем доступ к функции *profile_for*, можем передавать ей все что угодно и имеем выходной шифротекст.

Задание 7.

Усложним задачу. Модифицируем функцию из задания 3, пусть теперь в начало строки добавляется случайное количество байт. Таким образом, функция выглядит так:

```
AES_128_ECB(random_prefix || your_string || decodebase64(unknowStrBase64), random_key)
```

где *random_prefix* – случайное количество байт.

Задача – расшифровать unknowStrBase64.