Задание 8,

Фамилия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Любой случайный оракул стойкий к нахождению коллизий второго рода |  |
| b | Любая стойкая к коллизиям второго рода хэш-функция является стойкой к коллизиям первого рода |  |
| c | Любая стойкая к коллизиям второго рода хэш-функция является стойкой односторонней хэш-функцией |  |
| d | Любой стойкий MAC с фиксированным ключом и суперполиномиальной областью определения даёт стойкую к коллизиям хэш-функцию |  |
| e | На любую хэш-функцию на возможна теоретическая атака сложностью |  |
| f | Атака на стойкость хэш-функции в модели случайного оракула даёт атаку в модели одностороенней хэш-функции |  |
| g | Атака на стойкость к коллизиям второго рода для некоторой хэш-функции даёт атаку на случайный оракул для данной функции. |  |
| h | Отправка хэш-значения для некоторой величины по открытому каналу гарантирует, что противник не сможет восстановить данную величину. (используется хэш-функция, стойкая к коллизиям второго рода) |  |
|  | **Не заполнять!** | / 8 |

1. Рассмотри следующие функции сжатия

Задача – найти 4 различные пары :

Необходимо вывести в виде формулы получение этих пар и представить ответ в виде hex-строки.

|  |  |
| --- | --- |
|  | Ответ |
|  | Доп. Листы. |
| **Не заполнять!** | /4 |

1. Пусть и – стойкие к коллизиям хэш-функции на . Доказать, что – стойкая к коллизиям хэш-функция. Доказать от противного – предположить, что не стойкая к коллизиям.

|  |  |
| --- | --- |
|  | Ответ |
|  |  |
| **Не заполнять!** | /2 |

1. Пусть – стойкая к коллизиям хэш-функция. Какая их описанных хэш-функций является стойкой? Формально докажите или опровергните стойкость.

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |
| f |  |  |
| g |  |  |
| h |  |  |
|  | **Не заполнять!** | /8 |

1. Докажите утверждения ниже

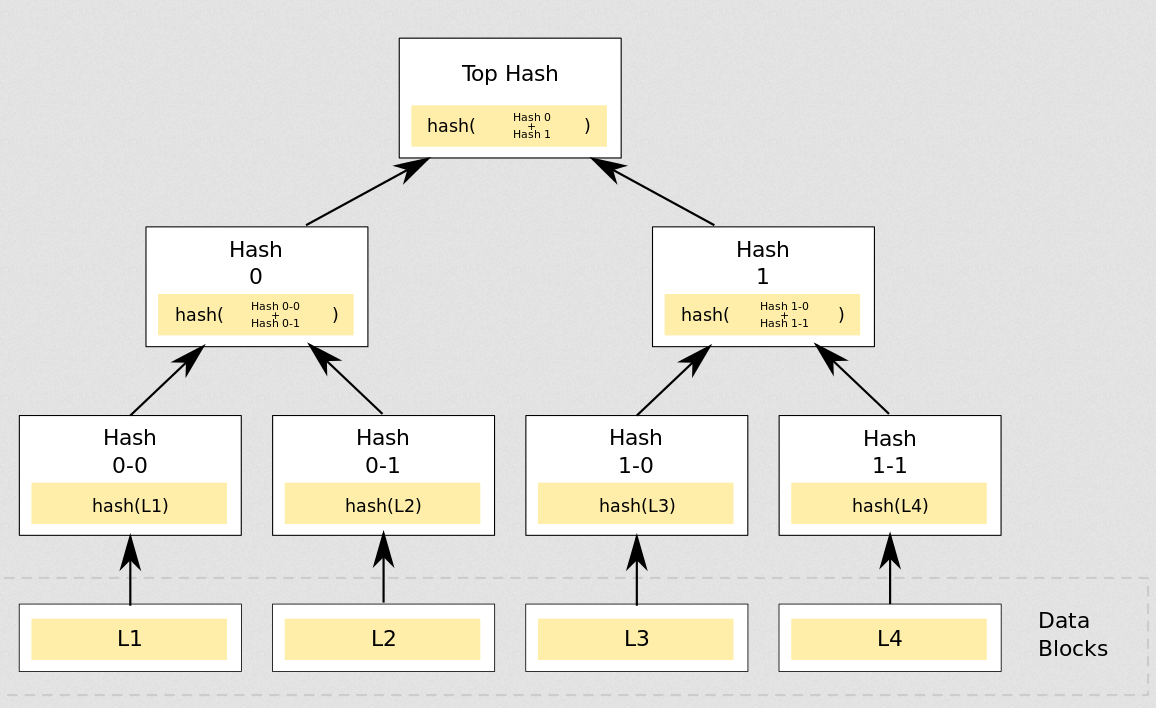
Пусть – случайный оракул. . Какова сложность нахождения тройной коллизии, т.е. трех различных величин ? (ответ + его вывод на доп. Листах)

|  |  |
| --- | --- |
|  | Ответ |
|  |  |
| **Не заполнять!** | /4 |

1. Почитать что такое дерево Меркла (Merkle tree), просмотра и осознание картинки на последней странице дз – достаточно.

Пусть имеется 647 различных файлов. Ответе на вопросы ниже.

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Какова высота дерева меркла для вычисления хэш-значения, обеспечивающего целостность всех файлов. |  |
| b | Какое количество хэшей необходимо пересчитать, при замене одного из файлов? |  |
| c | Какое минимальное количество хэшей необходимо пересчитать при замене 4-х файлов? |  |
| d | Какое максимальное количество хэшей необходимо пересчитать при замене 4-х файлов? |  |
| e | Какое количество хэшей необходимо вычислить при построении дерева? |  |
| f | Сколько узлов хэш значений отвечает за целостность одного файла? |  |
| g | За целостность какого количество файлов отвечает корневой узел? |  |
| h | Предположим необходимо переслать один из файлов. Предполагая, что получатель знает только значение корня дерева (и может проверить только его) Меркла, какое минимальное количество узлов дерева необходимо переслать вместе с файлом, для осуществления проверки файла получателем? (authentication path) |  |
|  | **Не заполнять!** | / 8 |



Задание 9,

Фамилия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Любую схему стойкого аутентифицированного шифрования можно использовать в качестве стойкого кода аутентичности |  |
| b | При использовании схемы MAC-then-Encrypt необходимо использовать независимые ключи для MAC и шифрования |  |
| c | При использовании схемы EAX необходимо использовать независимые ключи для MAC и шифрования |  |
| d | CCA стойкость более сильно определение, чем CPA стойкость |  |
| e | Возможно построить CCA стойкий шифр, который не будет CPA стойким |  |
| f | Обеспечение целостности открытых текстов не может быть обеспечена через целостность соответствующих шифртекстов |  |
| g | Схема Encrypt-and-MAC в общем случае является не стойкой |  |
| h | CCA стойкости достаточно для защиты аутентичности от пассивных противников |  |
| i | Целостность шифртекстов более сильное определение, чем целостность открытых текстов, при передаче шифртекстов по каналу связи |  |
|  | **Не заполнять!** | / 9 |

1. Пусть – случайная величина, полученная с использованием **неравномерного** распределения, :

Иными словами, выбирается из подмножества векторов в , для которых первые 128 бит – нулевые.

Пусть – стойкая PRF, с множеством ключей . Какие из PRF ниже является стойкими PRF (в практическом смысле, минимальный параметр стойкость – 80 бит), но не является стойкими при выборе c использованием распределения, описанного выше?

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
|  | **Не заполнять!** | / 8 |

1. Пусть – схема стойкого аутентифицированного симметричного шифрования на . Какие из схем ниже являются стойкими схемами аутентифицированного шифрования (формально докажите или опровергните).

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |
| f |  |  |
|  | **Не заполнять!** | /12 |

1. Пусть – строго стойкий блочный шифр. Докажите, что шифр ниже стойкий CCA шифр, но не стойкий AE шифр.

|  |  |
| --- | --- |
|  | Ответ |
|  | Доп. Листы. |
| **Не заполнять!** | /2 |

Задание 6,

Фамилия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**В заданиях для простоты вычислений предполагать, что гага=гиби=gibi= , мега=меби =mebi= , число секунд в году .**

1. В некоторой криптосистеме используется блочный шифр в детерминированном режиме CTR.

Ответе на вопросы ниже

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a | Предполагая стойкость блочного шифра с функцией зашифрования , является ли описанная криптосистема стойкой при **одноразовом использовании ключа** в теоретическом (предельном) смысле? (записать в ответ да или нет) Почему? (на доп листах) |  | |
| b | Предполагая стойкость блочного шифра с функцией зашифрования , является ли описанная криптосистема стойкой при **многоразовом использовании ключа (ключ используется для шифрования нескольких сообщений)** в теоретическом (предельном) смысле? (записать в ответ да или нет) Почему? (на доп листах) |  | |
| c\* | Пусть в качестве используется PRP, с длинной ключа 128 бит, размер блока 128 бит, параметр стойкости принять равным 128 бит.  Пусть имеется защищенный канал связи с пропускной способностью 100 mebibit/s, в котором непрерывно шифруются сообщения. Оценить вероятность атаки на криптосистему в течении одного года, при условии, что симметричный ключ не меняется. |  | |
| d\* | Пусть в качестве используется PRP, с длинной ключа 128 бит, размер блока 64 бит, параметр стойкости принять равным 120 бит.  Пусть имеется защищенный канал связи с пропускной способностью 100 mebibit/s, в котором непрерывно шифруются сообщения. Оценить необходимую частоту смены симметричного ключа, при заданной вероятности атаки равной . |  | |
|  | **Не заполнять!** | / 8 | / 8 |

*\* при вычислениях полагать что шифруется единственное сообщение максимальной длинны, которое может быть передано в указанном канале за заданное время.*

1. После анализа симметричной криптосистемы была получена следующая оценка стойкости в сведении к псевдослучайной функции , где – функция зашифрования блочного шифра, – максимальное число обращений к криптосистеме при фиксированном ключе, – размер блока PRF, – размер выхода криптосистемы.

Ответе на вопросы ниже

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a | Предполагая стойкость блочного шифра с функцией зашифрования , является ли описанная криптосистема стойкой в теоретическом (предельном) смысле? (записать в ответ да или нет) Почему? (на доп листах) |  | |
|  | **Не заполнять!** | / 1 | / 1 |

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Любая PRP является PRF |  |
| b | Любая PRF является PRP |  |
| c | Любая стойкая PRF является PRP |  |
| d | Любая стойкая PRP является стойкой PRF |  |
| e | Любая стойкая PRP с суперполиномиальным образом является стойкой PRF |  |
| f | Любой стойкий блочный шифр является стойкой PRF |  |
| g | Любой семантически стойкий шифр (одноразовое использование ключа) должен быть детерминированным |  |
| h | Любой CPA стойкий шифр является семантически стойким при одноразовом использовании ключа. |  |
|  | **Не заполнять!** | / 8 |

1. Пусть шифр на .

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Пусть длина сообщений и длины соответствующих шифртекстов совпадают для всех ключей. Показать, что – не CPA стойкий. |  |
| b | Пусть длина шифртекстов больше длины соответствующих открытых текстов на бит. Показать, что существует атака на CPA стойкость сложностью с преимуществом . |  |
|  | **Не заполнять!** | / 6 |

1. Рассмотрим следующую игру. Пусть шифр определён на , где множество сообщений такое, что можно эффективно выбрать случайное сообщение с равномерным распределением. Показать, что если CPA стойкий, тогда невозможно выиграть игру на генерацию двух одинаковых шифртекстов. Оценить преимущества в игре на генерацию одинаковых шифртекстов для CPA стойкого шифра. Игра на генерацию выглядит следующим образом – претендент генерирует случайный ключ, противник отправляет откртых текстов, получая шифртекстов на ключе претендента. Если хотя бы одна пара шифртекстов совпадает – противник выигрывает игру.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Не заполнять!** | / 4 | / 4 |

*n. Hard mode on.* ***Опционально (т.е. можно не делать).***

*Решить задачу 4.2. на странице 165 книги A Graduate Course in Applied Cryptography v0.4*

*+ 10 к итоговой оценке за семестр.*

Задание 7,

Фамилия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Любая стойкая PRF даёт стойкий MAC |  |
| b | Любая стойкая PRF, с суперполиномиальной областью значений даёт стойкий MAC |  |
| c | Любая стойкая PRP, с суперполиномиальной областью значений даёт стойкий MAC |  |
| d | Стойкая PRF является более сильным определением, чем стойкий MAC |  |
| e | На любой MAC на возможна теоретическая атака сложностью |  |
| f | На любой CBC-MAC на возможна теоретическая атака сложностью |  |
| g | CMAC требует использования трех независимых случайных ключей |  |
| h | Любое беспрификсное кодирование увеличивает длину сообщения |  |
| i | MAC обеспечивает целостность сообщений при передаче |  |
| j | MAC обеспечивает аутентичность источника информации (т.е. гарантирует, что только имеющий секретный ключ мог отправить это сообщение) |  |
| k | Добавление длины сообщения в конец сообщения является беспрификсным кодированием |  |
|  | **Не заполнять!** | / 10 |

1. Рассмотрим ECBC MAC. Вместо использования нулевого IV будем использовать случайный IV для каждого сообщения и включать его в состав итоговой метки. Т.е. . Данная система не является стойким MAC. Задача – от имени противника получить верный MAC для сообщения , где – размер блока PRF. Является ли данный MAC стойкой **беспрификсным** PRF?

|  |  |
| --- | --- |
|  | Ответ |
|  |  |
| **Не заполнять!** | /4 |

1. Alice отправляет данные 6 получателям . Задача – обеспечить целостность. Alice использует MAC. Использование одного ключа для всех получателей не обеспечивает целостность, так как если противником является один из получателей он может подделать MAC для любого сообщения и рассылать пакета от имени Alice. Вместо этого Alice использует 4 секретных ключа . Alice пересылает по защищенному каналу некое подмножество каждому получателю . Пересылая затем каждое сообщение, она включает также 4 кода аутентичности для каждого сообщения, выработанных на этих ключах. Каждый пользователь считает пакет целостным, если для всех его ключей совпали коды аутентичности (те кода, которые не соответствуют ключам пользователя им игнорируются).

Как Alice должна распределить ключи между пользователями?

|  |  |
| --- | --- |
|  | Ответ |
|  |  |
| **Не заполнять!** | /4 |

1. Пусть – стойкий MAC на . Какой их описанных MAC является стойким? Формально докажите или опровергните стойкость. Если явно не указан алгоритм проверки – считать MAC детерминированным.

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |
| f |  |  |
| g |  |  |
| h |  |  |
| i |  |  |
| j |  |  |
| k |  |  |
|  | **Не заполнять!** | /22 |

1. Докажите утверждения ниже

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Пусть – MAC.  Пусть , .  Докажите, что – стойкий, если хотя бы один из – стойкий MAC | (доп листы) |
| b | Пусть – **детерминированные** MAC.  Пусть .  Докажите, что – стойкий, если хотя бы один из – стойкий MAC | (доп листы) |
|  | **Не заполнять!** | /4 |

## Задание

Дана REST служба с API указанным ниже. Для аутентификации в службе пользователь должен передать её токен через метод ValidateRawToken.

Получить зашифрованный токен можно используя метод GetEncryptedToken.

Проверить корректность зашифрованного токена можно используя метод ValidateEncryptedToken, расшифровывающий токен, и проверяющий его корректность.

Проверить корректность незашифрованного ("сырого") токена можно используя метод ValidateRawToken, проверяющий корректность переданного токена.

Задача - вызвав метод ValidateRawToken, получить сообщение "Raw Token decoded and validated. Wellcome to secretNet!".

Шифрование токенов производится в режиме Mac-Then-Encrypt, CBC-AES, HMAC.

## Ход работы.

### Тестирование

<userId> = имя аккаунта на GitHub (или фамилия студента)

<challengeId> = debug

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/IvIsTime
2. Получить зашифрованный дебаговый токен с помощью метода GET <host>/api/PaddingOracle/<userId>/debug/GetEncryptedToken
3. Восстановить дебаговый токен используя вызовы метода GET <host>/api/PaddingOracle/<userId>/debug//ValidateEncryptedToken/{encryptedToken}
4. Убедиться, что расшифрованный дебаговый токен равен следующей последовательности байт:  
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32

<userId> = имя аккаунта на GitHub (или фамилия студента)

<challengeId> = 1

1. Получить зашифрованный токен токен с помощью метода GET <host>/api/PaddingOracle/<userId>/<challengeId>/GetEncryptedToken
2. Восстановить токен используя вызовы метода GET <host>/api/PaddingOracle/<userId>/<challengeId>/ValidateEncryptedToken/{encryptedToken}
3. Аутентифицироваться с использованием метода GET <host>/api/PaddingOracle/<userId>/<challengeId>/ValidateRawToken/<{rawToken}>, получив сообщение "Raw Token decoded and validated. Wellcome to secretNet!"

### Сдача лр

шаги 5 - 7 этапа тестирования аналогично для 10 различных <challengeId>.

## Описание API

Rest запросы, в заголовке выставлен Content-Type: application/json; charset=utf-8.

### Описание методов

## GET <host>/api/PaddingOracle

Проверка работоспособности контроллера. Возвращает operating. Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |

## GET <host>/api/PaddingOracle/{userId}/{challengeId}/GetEncryptedToken

Генерирует, подписывает с помощью MAC, и зашифровывает токен в режиме CBC на фиксированном для задания для задания ключе.  
Ответ не кодируется в BASE64.  
Ответ представляет собой hex строку.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

## GET <host>/api/PaddingOracle/{userId}/{challengeId}/ValidateEncryptedToken/{encryptedToken}

Расшифровывает указанный токен и проверяет его валидность.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |
| <encryptedToken> | зашифрованный токен, в виде hex строки |

## GET <host>/api/PaddingOracle/{userId}/{challengeId}/ValidateRawToken/{rawToken}

Проверяет переданный "сырой" (незашифрованный) токен и аутентифицирует пользователя.  
Возвращает строку Raw Token decoded and validated. Wellcome to secretNet! в случае корректности токена.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |
| <rawToken> | "сырой" (незашифрованный) токен в виде hex строки |

### Реально возможный use-case

Пусть аутентификация на сайте осуществляется с помощью cookie. Часто cookie зашифрованы, и не прозрачны для пользователя, но могут нести  
конфиденциальную информацию. Например, cookie могут содержать персональные данные пользователя. При этом аутентификация производится по зашифрованной cookie.

Если злоумышленник перехватил простроченную cookie пользователя, он не сможет использовать её после аутентификации. Но если время действия хранится внутри cookie  
(как оно часто и бывает), сервер всё равно будет вынужден расшифровать cookie для её проверки. Таким образом противник получает оракул расшифрования, аналогичный методу ValidateEncryptedToken,  
который он может использовать для извлечения информации из cookie. По "сырой" coookie обычно не аутентифицируют, так что метод ValidateRawToken служит в лр  
для валидации результата расшифрования. Иными словами ральная атака происходит не на аутентификацию, а на конфиденциальность информации в cookie.

Дана REST служба с API указанным ниже.

В качестве MAC используется следующая функция: MAC(k,x) = SHA-1(k || x)

Задача - получить ответ от службы с текстом "Wellcome to secretNet!"

**ВАЖНО!** Все передаваемые сообщения должны быть предварительно закодированы с помощью BASE64. Полученные сообщения так же должны быть декодированы из BASE64 в указанный тип.

**ВАЖНО!** Все строки кодируется с использование кодировки ASCII.

т.е. передача строки для лр выглядит следующим образом:  
строка -> (asсii) -> массив байт -> (base64) -> строка -> json

## Ход работы.

### Тестирование

<userId> = имя аккаунта на GitHub (или фамилия студента)

<challengeId> = 1

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/Sha1Mac
2. Получить зашифрованное MAC для строки "user=<userId>; с помощью метода GET <host>/api/Sha1Mac/<userId>/<challengeId>/mac
3. Получить сообщение содержащее текст ;admin=true и MAC для него
4. Получить ответ от метода POST <host>/api/Sha1Mac/<userId>/<challengeId>/<mac>/verify, убедиться что текст соотвествует строке "Wellcome to secretNet!".

### Сдача ЛР

шаги 1 - 4 этапа тестирования аналогично для 5 различных <challengeId>

## Описание API

## GET <host>/api/Sha1Mac

Проверка работоспособности контроллера. Возвращает operating. Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |

## GET <host>/api/Sha1Mac/<userId>/<challengeId>/mac

Возаращает строковое hex представление MAC для строки $"user=<userId>;" Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

## POST <host>/api/Sha1Mac/<userId>/<challengeId>/<mac>/verify

Проверяет корректность MAC для указанных в теле запроса данных. В случае если в теле указана строка, содержащая ;admin=true  
возвращает строку Wellcome to secret net!, в случае корректного MAC. Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |
| <mac> | hex представление MAC для проверки |

В рамках данной работы необходимо реализовать протокол sigma, используя в качестве примитивов схемы ECDSA в качестве схемы формирования электронных подписей, ECDH - в качестве схемы формирования общего секрета, AES-CTR - в качестве алгоритма шифрования, HMAC-SHA256 в качестве PRF и MAC.

Пусть ECDSA состоит из тройки алгоритмов - (GEN, SIGN\_X(m), VER\_X(m, s)), где

GEN - функция генерации ключевой пары

SIGN\_X(m) - функция формирования подписи для сообщения m с использованием закрытого ключа пользователя X

VER\_X(m, s) - функция проверки подписи s для сообщения m с использованием открытого ключа пользователя X.

Пусть ECDH состоит из пары алгоритмов - (GEN, GET(X\_SK, Y\_PK)), где

GEN - функция генерации ключевой пары

GET(X\_SK, Y\_PK) - функция выработки общего секрета из открытого ключа Y и закрытого ключа X

0. Предварительный обмен ключами (выполняется вне протокола)

A: (ECDSA\_PK\_A, ECDSA\_SK\_A) <- ECDSA.GEN; ECDSA\_PK\_A -> B  
B: (ECDSA\_PK\_B, ECDSA\_SK\_B) <- ECDSA.GEN; ECDSA\_PK\_B -> A

1. Описание протокола Sigma.

A:   
(DH\_PK\_A, DH\_SK\_A) <- ECDH.GEN;   
r\_A <-^r {0,1}^R;   
(DH\_PK\_A, r\_A) -> B  
  
B:   
(DH\_PK\_B, DH\_SK\_B) <- ECDH.GEN;   
r\_B <-^r {0,1}^R;   
(k\_m, k\_e) <- PRF\_(r\_A||r\_B)(ECDH.GET(DH\_SK\_B, DH\_PK\_A));   
(DH\_PK\_B, r\_B, B, ECDSA.SIGN\_B(DH\_PK\_A, DH\_PK\_B), MAC\_k\_m(B)) -> A  
  
A:   
(k\_m, k\_e) <- PRF\_(r\_A||r\_B)(ECDH.GET(DH\_SK\_A, DH\_PK\_B));   
(A, ECDSA.SIGN\_A(DH\_PK\_B, DH\_PK\_A), MAC\_k\_m(A)) -> B

где <-^r Q - выбор случайного элемента из Q (равномерное распределение);

R - параметр, принять равным 128.

**ВАЖНО - Неявно предполагается проверка всех полученных подписей и кодов аутентичности. Процедуры проверки опущены в описании для простоты. При реализации необходимо реализовать все проверки, и в случае нарушения аутентичности выдавать соответствующее исключение.**

2. Отправка зашифрованных данных

A: c <- AES\_k\_e(m); (c, MAC\_k\_m(c)) -> B

Описание лабораторной работы

1. Реализовать протокол Sigma, используя встроенные средства языка, получив общие ключи для аутентификации и шифрования.
2. Использовать полученные ключи для аутентифицированного шифрования AES-CRT-Then-Hmac для обеспечения аутентичности и секретности данных.
3. Проверить аутентичность полученных данных на стороне получателя и расшифровать сообщение.

Дополнительные ссылки

https://webee.technion.ac.il/~hugo/sigma-pdf.pdf (стр 17-18)

Дана REST служба с API указанным ниже.

Задача - построить эффективный распознаватель режимов шифрования (ECB, CBC).

**ВАЖНО!** Все передаваемые сообщения должны быть предварительно закодированы с помощью BASE64.

Полученные сообщения так же должны быть декодированы из BASE64 в указанный тип.

**ВАЖНО!** Все строки кодируется с использование кодировки ASCII.

т.е. передача строки для лр выглядит следующим образом:  
строка -> (asсii) -> массив байт -> (base64) -> строка -> json

## Ход работы.

### Тестирование

<userId> = имя аккаунта на GitHub (или фамилия студента)

<challengeId> = 1

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/EncryptionModeOracle
2. Получить зашифрованное сообщение с помощью метода POST <host>/api/EncryptionModeOracle/<userId>/<challengeId>/noentropy
3. Определить режим шифрования
4. Проверить верность ответа использовав метод GET <host>/api/EncryptionModeOracle/<userId>/<challengeId>/verify
5. Проверить корреиность программы для <challengeId> = 1..10

### Сдача лр

шаги 1 - 3 этапа тестирования аналогично, но использование метода POST <host>/api/EncryptionModeOracle/<userId>/<challengeId> на шаге 2.  
для 50 различных <challengeId>.

## Описание API

Rest запросы, в заголовке выстален Content-Type: application/json; charset=utf-8.

### Описание методов

## GET <host>/api/EncryptionModeOracle

Проверка работоспособности контроллера. Возвращает operating. Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |

## POST <host>/api/EncryptionModeOracle/<userId>/<challengeId>

Дополняет полученные данные случайными байтами вначале и в конце.

После чего зашифровывает данные на случайном ключе, либо в режиме CBC, либо в ECB.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

## POST <host>/api/EncryptionModeOracle/<userId>/<challengeId>/noentropy

Дополняет полученные данные случайными байтами вначале и в конце.

После чего зашифровывает данные на фиксированном для задания ключе, либо в режиме CBC, либо в ECB.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

## GET <host>/api/EncryptionModeOracle/<userId>/<challengeId>/verify

Возвращает, какой режим использует метод POST <host>/api/EncryptionModeOracle/<userId>/<challengeId>/noentropy  
в указанном задании. Ответ не кодируется в BASE64. Возвращает "ECB" или "CBC"

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

Периодически в приложениях необходимо сравнивать две величины. Рассмотрим частный случай - сравнение массивов (или строк) применительно к криптографии.

В большинстве языков программирования, реализована операция (или функция) сравнения массивов (или строк) обозначаемая '==' или методом *.equals(*)

В подавляющем большинстве случаев, данная операция выполняется по следующему алгориму:

bool areEqual(int arr1[], int arr2[], int n, int m)   
{   
 // If lengths of array are not equal means   
 // array are not equal   
 if (n != m)   
 return false;   
   
 // Linearly compare elements   
 for (int i=0; i<n; i++)   
 if (arr1[i] != arr2[i])   
 return false;   
   
 // If all elements were same.   
 return true;   
}

Как видно из кода, в целях оптимизации метод завершает свою работу при первом несовпадении. Таким образом, время выполнения алгоритма зависит от подаваемых на вход величин: массивы, отличающиеся в последнем байте, будут сравниваться дольше, чем массивы, отличающиеся в первом.

Для большинства приложений, такое поведение метода сравнения является корректным и приемлемым, так как позволяет экономить до n/2 операций при сравнении случайных массивов.

Для криптографии - не всё так радужно, так как использование данного метода позволяет реализовать так называемую атаку по времени, являющуюся частным случаем атаки по побочным каналам.

Пусть имеется сервер, проверяющий HMAC, присланный пользователем. Для этого вычисляется целевой HMAC от некоторых данных, а затем сравнивается с HMAC'ом, присланным пользователем, используя стандартный метод '==', описанный выше.

Очевидно, что время ответа сервера зависит от того, насколько MAC пользователя совпадает с целевым. При этом время проверки линейно зависит от числа символов в совпавшем префиксе.

Противник может осуществить атаку - имея корректный префикс длины i-1, он перебирает i-й байт в последовательности, замеряя время ответа. Если время ответа для некоторого символа больше, чем для остальных - этот символ считается угаданным, он добавляется в префикс и перебирается i+1 символ.

## Задание

Дана REST служба с API указанным ниже.

Задача - получить от севера ответ Wellcome to SecretNet!

NB - для простоты лр и уменьшения времени перебора размер MAC был ограничен 8 байтами. Для "боевого" исполнения размер в 32 байта не является проблематичной для данного типа атак, но время атаки возрастёт линейно.

## Ход работы.

### Тестирование

<userId> = имя аккаунта на GitHub (или фамилия студента)

<challengeId> = 1

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/HmacTiming
2. Получить ответ Wellcome to SecretNet!
3. Найти минимальное значение параметра d=<delay>, при котором код перестаёт работать
4. Написать лр (не удаля старый код!!!) который выполнит лр для значения d/2
5. Найти по крайней мере два алгоритма, с помощью которых можно реализовать сравнение за константное время.
6. Найти, если ли в вашем языке программирования встроенный метод, для сравнения за константное время (может быть методом у хэш функций).

### Сдача лр

шаги 1 - 3 этапа тестирования аналогично для 20 различных <challengeId>.

## Описание API

Rest запросы, в заголовке выстален Content-Type: application/json; charset=utf-8.

### Описание методов

## GET <host>/api/HmacTiming

Проверка работоспособности контроллера. Возвращает operating. Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |

## GET <host>/api/HmacTiming/<userId>/<challengeId>/data=<data>&mac=<mac>&delay=<delay>

Проверяет аутентичность пользовательских данных. Для сравнения MAC используется сравнение, с непостоянным временем исполнения.  
Для простоты лр дополнительную задержку можно указывать в параметре <delay>

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |
| <data> | данные, на которых вычисляется MAC, hex строка |
| <mac> | MAC для проверки, hex строка |
| <delay> | задержка в мс на один символ в операции сравнения, принимает значения 1...1000 |

Одноразовый блокнот на фиксированном ключе. Кодировка ASCII.

Расшифрусте последний шифртекст. Шифртексты представлены в hex.

Подсказка - что будет, когда пробел ксортися с буквами из [A-Za-z]?

()

f9e4d9bff5bac624e2ce1bb3c4cce85336d650082caa9037eb48f6a5aedf9ff6f8f4daa2a3e358a783321dd0d09ffa57f9528f4551c65a726620117c81ff867786c50dfaed6056c670a8103b4dab6fe7cfae8bfb71803aa04023e0271b2bc012e10e09901d58c2d654d34004109da5ec79e55f8b42c624ce6c6befeb44edb8f36229c19e32fe40e7596bca

ebf495b9e6f4912df6c10bfcc69ef3593294520461ad9b2fa400e3f1b1d79fbeb6ead8e3a5fe5eead73917d2d28fe312e944c20a6bc60931666e52729fe28c25c0d71ae1ec7756dd7eed532741b679ed9da39a8f3f993cba516eee720c64c85cb5391d95164f818500c808031e9daba1

fae99cfcfabd8527a3d907b5d88bbc5a31994b0961839020b215acf4f9df98beb6ead8e3baf20da9d12802c9cf8dfc53fb49c41775831970696e556f84fa8c77818402e1f63219d438ab513b5bbf2de1dcfcc9ee7cd717b54623c3681121cf

fae99cfcf7bd962ae6df1bb9ce98bc4b21995a0822ad9165a900e3e4f9c18effb3a5caadaee554bad7381dd3808be255e453c8116ece5a7d68215a6ecded9a7787cb01eaa27305927ba4403d5db479e7c5fa9abe239837a14b66e5271d3d8753b50f1c8b1c53c1d645d20314089fbaa562e30d8f4b813fd5687af4a649e0cacc6825df8362967bfc5863d6825c5365e3

f7ee8cfcf0bb8865f78d18bdd898bc4f3cd65c0838e89465b81cb7a5b6d0cbfdb9f78fa8a8ee5eeac5231dd0808bae55fe5881126ecc5a62772b52748ce0802d85d74ee7ec3205c67dac5c3c56a12de1dcfcc9ee7cd71eb55a60a1551030c25cf7191a9e535ec99b4dd90e121881a9ec62e30dad4b8f20d7647c

fae99caef1f48730e68d1babd9cce84223934d5d2eaed526b900b3f1b6d199ffa8edd6e3e0b759a2c22552cac883ed5aab56c8096a831174623e116e88ef9b3294d74efde37413927ebf5f3818bf62f7cfaed6a725833fb10870e8740b21d51eb51d069d5349ce97549c170e188ca6ec7ae44182078d35c2712eefae0abf8fe8736cc08b74d301f34761ded0485d7eff397146ba3e57082d2bf3973555225dd56c959411d999165fd45718ae

fae99caef1f48730e68d1babd9cce84223934d5d2eaed526b209b7eabec48aeeb0fc95e3a2f948ead73913c9808be25ee456d24572cb1f31402147789fe284328ed04efaed3203c17ded52274db268a2dbe1c8ad34d727bb0861f3621e2f8746fd19489a1c59c3da00dd0e025180a0a92df9458f53c622c2707bf5b90cbecae8682993ad7dc044e75b63d69e45127fe239635aa97b4714353af8c37317707cc239829e11e8881b50da1e04b373

f9e4d9bff5bac631e6c84fa8de89bc4b3c9f500961bf9d20b91ce3f1b1d3cbfdb0ecdfe3a4e40dbfcd3913cdd093ae5bed01c04571d1157f606e537499ac8024c0d70be0f63217dc7ced533a56b578efd8fd9aa33e8536f4586cf6620d64c140fa11488d1b5886934eca09141e81a3a963f90dc307a734ce215df4aa04a498

efa1d1ace6bd9023f7c842b7d395b51b7393501e33b18531a216ada5aad583fbb5e08fb0b9f659afd071419dc186e95df948d50d6bd05631692f5c7881f5c936c0d41ce1e17712c76aa8103357b42de5d8e0dfbc30833aba4f23ea6206378b12f45c188b1c5ec39255ce05461780bcec68e34e9c5e9624ce6f69b0eb08a38ebc616cc3987dd544f1407cd6d0575d79ad7d734abe2255122920facd2a

fae99cfcd7bb8821eade0afcf994fa5421927a1422bc9c2aa518b1fcf99ed9aee8b386e3a9f24ba3cd34019dc398f742ff4e810475830e79626e506f99ac8631c08419fceb661fdc7fed5f754ae67eedd1f8d3a036d730bb4c66f229

fae99cfce7b18530e6d94fb1d39fef5a34931e1432f2d512a31cada5acc582f0bfa5cee3bee35fafc23c52dec99ae657f90d810b63d51f63273b4278cdf88132c0cf0bf7a27f19c07ded443d59a82dedd3eddf

Дана REST служба с API указанным ниже.

Задача - получить ответ от службы с текстом "Wellcome to secretNet!"

В качестве шифра используется поточный шифр. Первые 16 байт шифртекста являются значением счётчика. Изменять нужно лишь последующие байты шифртекста, оставив первые байты без изменений.

**ВАЖНО!** Все передаваемые сообщения должны быть предварительно закодированы с помощью BASE64. Полученные сообщения так же должны быть декодированы из BASE64 в указанный тип.

**ВАЖНО!** Все строки кодируется с использование кодировки ASCII.

т.е. передача строки для лр выглядит следующим образом:  
строка -> (asсii) -> массив байт -> (base64) -> строка -> json

## Ход работы.

### Тестирование

<userId> = имя аккаунта на GitHub (или фамилия студента)

<challengeId> = 1

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/StreamIntegrity
2. Получить зашифрованное сообщение с помощью метода GET <host>/api/StreamIntegrity/<userId>/<challengeId>/noentropy
3. Декодировать сообщение из BASE64, получив шифртекст
4. Получить модифицированное сообщение
5. Закодировать модифицированное сообщение с помощью BASE64 и отправить с использованием метода POST <host>/api/StreamIntegrity/<userId>/<challengeId>
6. Получить ответ от сервера, декодировать из BASE64, убедиться, что текст соотвествует строке "Wellcome to secretNet!".

### Сдача лр

шаги 1 - 6 этапа тестирования аналогично, но использование метода GET <host>/api/StreamIntegrity/<userId>/<challengeId> на шаге 2.

## Описание API

Rest запросы, в заголовке выстален Content-Type: application/json; charset=utf-8.

### Описание методов

## GET <host>/api/StreamIntegrity

Проверка работоспособности контроллера. Возвращает operating. Ответ не кодируется в BASE64.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |

## GET <host>/api/StreamIntegrity/<userId>/<challengeId>

Возаращает зашифрованную строку "Here is some data to encrypt for you" на фиксированном для пары (userId, challengeId) ключе со случайным счётчиком.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

## GET <host>/api/StreamIntegrity/<userId>/<challengeId>/noentropy

Возаращает зашированную строку "Here is some data to encrypt for you" на фиксированном для пары (userId, challengeId) ключе с фиксированным счётчиком.

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

## POST <host>/api/StreamIntegrity/<userId>/<challengeId>

Расшифровывает строку на фиксированном для пары (userId, challengeId) ключе.  
Если строка расширована как "Token: 8ce08ad2d48d7d356db43", возвращает строку "Wellcome to secretNet!".

|  |  |
| --- | --- |
| Параметр | Описание |
| <host> | имя хоста веб службы |
| <userId> | идентификатор студента |
| <challengeId> | идентификатор задания |

### Пример составления тестового post запроса на python

import json  
import requests  
  
r = requests.post("http://192.168.130.133/api/values", data=json.dumps("HereIsSomeStringData"), headers = {'Content-Type': 'application/json'})  
print(r.status\_code, r.reason)  
print(r.text)

### Пример post запроса

POST http://192.168.130.133/api/values HTTP/1.1  
Connection: Keep-Alive  
Content-Type: application/json; charset=utf-8  
Content-Length: 22  
Host: 192.168.130.133  
  
"HereIsSomeStringData"

Здесь и далее используется AES с длиной ключа 128 бит. В качестве хэш-функции используется SHA-256.

## 1. Реализовать функцию на вашем языке программирования со следующим интерфесом:

byte[] AesBlockEncrypt(byte[] key, byte[] data, bool isFinalBLock, string padding), где

key - байтовое представление ключа блочного шифра

data - блок для шифрования

isFinalBLock - флаг того, что передан последний блок шифруемого открытого текста

padding - вид дополнения, принимает значение MAC\_padding (10..00000) для OMAC, 'PKCS5' для truncated-MAC, 'NONE' для HMAC.

В ходе реализации необходимо пользоваться стандартной или общеизвестной реализацией алгоритма AES. Если библиотека не поддерживает одноблочное шифрование AES, необходимо воспользоваться режимом ECB.

Пример одноблочного шифрования на C#.

namespace AesExample  
{  
 using System;  
 using System.Security.Cryptography;  
 using System.Text;  
  
 internal class Program  
 {  
 private static void Main(string[] args)  
 {  
 // Your Key here, 16 bytes  
 byte[] key =  
 { 0x73, 0x69, 0x78, 0x74, 0x65, 0x65, 0x6e, 0x2d,  
 0x62, 0x79, 0x74, 0x65, 0x2d, 0x6b, 0x65, 0x79 };  
  
 // Plaintext, 16 bytes string, utf-8  
 string stringPt = "sixteen-byte-msg";  
  
 // Raw Plaintext, bytes  
 byte[] pt = Encoding.UTF8.GetBytes(stringPt);  
  
 // Resulted Ciphertext will be here  
 byte[] ct = new byte[16];  
  
 // Create new AES instance  
 using(Aes aes = new AesCryptoServiceProvider())  
 {  
 // Select Encryption mode  
 aes.Mode = CipherMode.ECB;  
  
 // Create encryptor with your key and zero IV  
 using (var aesEncryptor = aes.CreateEncryptor(key, new byte[16]))  
 {  
 // Transform one block  
 aesEncryptor.TransformBlock(pt, 0, 16, ct, 0);  
 }  
  
 // Get hex-string representation of Ciphertext  
 string hex = BitConverter.ToString(ct);  
 Console.WriteLine(hex.Replace("-", ""));  
 }  
 }  
 }  
}

c++

#include <stdio.h>   
#include <openssl/aes.h>   
  
static const unsigned char key[] = {  
 0x73, 0x69, 0x78, 0x74,   
 0x65, 0x65, 0x6e, 0x2d,  
 0x62, 0x79, 0x74, 0x65,  
 0x2d, 0x6b, 0x65, 0x79  
};  
  
int main()  
{  
 unsigned char text[]="sixteen-byte-msg";  
 unsigned char enc\_out[80];  
 unsigned char dec\_out[80];  
  
 AES\_KEY enc\_key;  
  
 AES\_set\_encrypt\_key(key, 128, &enc\_key);  
 AES\_encrypt(text, enc\_out, &enc\_key);   
  
 int i;  
  
 for(i=0;\*(enc\_out+i)!=0x00;i++)  
 {  
 printf("%X ",\*(enc\_out+i));  
 }  
 printf("\n");  
  
 return 0;  
}

python

from Crypto.Cipher import AES  
cipher = AES.new(b'sixteen-byte-key',AES.MODE\_ECB)  
cipher.encrypt(b'sixteen-byte-msg').hex()

## 2. Реализовать OMAC, truncated-MAC (64 бита) с использованием функции AesBlockEncrypt, HMAC на основе SHA-256.

2.0 Реализовать интерфейс void SetKey (byte[] key)

* инициализирует объект MAC ключом key.

2.1 Реализовать интерфейс void MacAddBlock(byte[] dataBlock)

* добавляющий блок данных для вычисления кода аутентичности, с использованием AesBlockEncrypt или SHA-256.

2.2 Реализовать интерфейс byte[] MacFinalize()

* возвращающий результат вычисления кода аутентичности, для всех запрошенных блоков.

2.3 Реализовать интерфейс byte[] ComputeMac(byte[] data)

* вычисляющий код аутентичности для прозвольных данных, используя метод MacAddBlock

2.4 Реализовать интерфейс bool VerifyMac(byte[] data, byte[] tag)

* проверяющий код аутентичности для прозвольных данных, используя метод ComputeMac

2.5 (Опционально) Использовать реализации реализованных алгоритмов в вашем языке программирования для валидации вашей реализации.

## 3. Для каждого алгоритма выработки кода аутентичности вычислить и проверить для произвольного текст длины 2,5 блока.

3.1 Поменять один бит в блоке - убедиться, что проверка кода аутентичности не выполняется, для модифицированных данных.

## 4. Замерить производительность OMAC и HMAC для сообщений, длины 0.1, 1, 10, 1024 KB (не менее 1000 сообщений).

Построить график зависимости среднего времени выполнения от размера сообщений (2 графика).

Доп материалы

Ссылка OMAC (CMAC) https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf (пункт 6)