Задание 4,

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

1. Пусть – PRF на . Выберите верные утверждения, доказав или опровергнув их.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a | *; –* может быть стойкой |  | |
| b | *; –* **не** может быть стойкой |  | |
|  | **Не заполнять!** | / 2 | / 2 |

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Любой стойкий блочных шифр семантически стойкий для любых сообщений имеющих размер, кратный длине блока |  |
| b | Если блочный шифр имеет ключ длины 128 бит, его параметр стойкости не может превосходить 128 бит |  |
| c | Если блочный шифр имеет ключ длины 128 бит, его параметр стойкости не может быть ниже 64 бит |  |
| d | Возможно существование стойкого блочного шифра, не стойкого к восстановлению ключа |  |
| e | Стойкость блочного шифра можно свести к стойкости его функции зашифрования, как псевдослучайной подстановки |  |
| f | Блочной шифр в режиме ECB является шифром подстановки |  |
| g | Если стойкий блочный шифр имеет ключ длины 128 бит и размер блока 128 бит то он является абсолютно стойким. |  |
| h | Невозможно построить абсолютно стойкий шифр на основе блочного шифра с длинной ключа 128 бит, размером блока 64 бита для сообщений длины 128 бит. |  |
|  | **Не заполнять!** | / 8 |

1. Пусть – стойкая PRF, . Для некоторого параметра рассмотрим . Является ли - стойкой PRF? Докажите

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

1. Рассмотрим игру на семантическую стойкость для случайных сообщений: вместо выбора произвольных сообщений противник может выбрать сообщения только случайно из множества сообщений. В остальном игра идентично обычной игре на семантическую стойкость. Являются ли игры эквивалентными? (записать в ответ). Если нет – выясните какая из них является более строгой, докажите это сведением, продемонстрируйте пример шифра, стойкого в одной из моделей семантической стойкости, и не стойкого в другой. Если игры эквивалентны – формально докажите это.

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

1. Пусть – стойкая PRF, . Какие из следующих алгоритмов является стойкими PRF? Для каждого алгоритма предоставить доказательство стойкости или атаку.

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d | . |  |
| e |  |  |
| f |  |  |
| g | *.* |  |
| h |  |  |
| i |  |  |
|  | **Не заполнять!** | /9 |

1. Рассмотрим модифицированную игру на стойкость PRF. Назовём игру, описанную в лекции – адаптивной, в том смысле, что противник отправляет сообщения последовательно, после получения ответа на свое предыдущее сообщения от претендента. Т.е. при формировании сообщения противник может учитывать полученные от претендента результаты . Рассмотрим неадаптивную версию игры – противник отправляет сообщения **одновременно**, и получает результаты , где здесь и далее . Преимущество противника в неадаптивной игре описывается аналогично адаптивной версии.

Пусть стойкая PRF на - супер полиномиальная. Построим следующим образом: для некоторого элемента определим , для остальных .

Формально докажите или опровергните утверждения ниже.

Для заданий с и d определите аналогичную задачу для блочных шифров  и .

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a | – не стойкая PRF против адаптивных противников |  |  |
| b | *’* –стойкая PRF против неадаптивных противников |  |  |
| с | – не стойкий блочный шифр против адаптивных противников |  |  |
| d | *’* – стойкий блочный шифр против неадаптивных противников |  |  |
|  | **Не заполнять!** | /4 | /4 |

Задание 1.

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

1. Вычислить энтропию () следующих величин:

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

1. Рассмотрим игру с двумя экспериментами.
   1. В эксперименте 0 претендент подбрасывает монетку и возвращает **РЕШКА**, если выпалa решка, и **ОРЁЛ** если орёл.
   2. В эксперименте 1 претендент всегда возвращает **ОРЁЛ**.

Цель противника различить два эксперимента. Пусть событие того, что в эксперименте противник возвращает 1. Преимущество противника .

Вычислить для следующих алгоритмов:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c | , иначе 0 |  |
| d | , иначе 1 |  |
| e |  |  |
| f | , построить |  |
|  | **Не заполнять!** | / 12 |

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Абсолютно стойкий шифр всегда семантически стойкий |  |
| b | Любой шифр Шеннона является абсолютно стойким |  |
| c | Аддитивный одноразовый блокнот – семантически стойкий шифр |  |
| d | Аддитивный одноразовый блокнот переменной длины – семантически стойкий шифр |  |
| e | Если шифр имеет длины ключей больше длин шифртекстов то он абсолютно стойкий |  |
| f | Если шифр имеет энтропии и длины ключей больше энтропий и длин шифртекстов то он абсолютно стойкий |  |
| g | Если для всех пар сообщение – шифртекст () имеется одинаковое количество ключей , таких что , то шифр на - абсолютно стойкий |  |
| h | Для семантически стойкого шифра энтропия ключа всегда больше или равна энтропии открытого текста. |  |
|  | **Не заполнять!** | / 8 |

1. Пусть – одноразовый блокнот на (множество векторов длины , для которых чётные координаты равны 1).   
   Является ли семантически стойким шифром? Если нет продемонстрируйте атаку с преимуществом равным 1.

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

1. Пусть – подстановочный шифр на (множество подстановок на ).   
   Является ли семантически стойким шифром? Если нет продемонстрируйте атаку с преимуществом равным 1.

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

1. Пусть – семантически стойкий шифр на . Какие из следующих алгоритмов является семантически стойкими? Для каждого алгоритма предоставить доказательство стойкости или атаку.

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |
| f |  |  |
| g |  |  |
| h |  |  |
| i |  |  |
|  | **Не заполнять!** | /18 |

1. – семантически стойкий шифр на . Пусть – функция сжатия без потерь. Заметим, что демонстрирует разный уровень сжатия для различных сообщений.
   1. Пусть в игре на семантическую стойкость Претендент сжимает сообщения перед зашифрованием, т.е. . Является ли данная схема семантически стойкой? Если да – доказать, иначе – продемонстрировать атаку. Имеет ли данная схема смысл для уменьшения размера шифрткеста? Почему?
   2. Пусть в игре на семантическую стойкость Претендент сжимает шифртекст после зашифрования, т.е. . Является ли данная схема семантически стойкой? Если да – доказать, иначе – продемонстрировать атаку.

Имеет ли данная схема смысл для уменьшения размера шифрткеста? Почему?

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a | . |  |  |
| b |  |  |  |
|  | **Не заполнять!** | /2 | /2 |

1. Пусть – семантически стойкий шифр на . Банковская организация желает разделить секретный ключ **на две части** и , так, что **обе** необходимы для расшифрования.

Банк генерирует случайное число и вычисляет . Тогда .

Аналогичная задача для трех сторон: разделяя ключ на **три** части можно получить ключ по **любым двум** из ним: банк генерирует пары и , такие что . Как следует разделить части пар между сторонами?

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |
|  | **Не заполнять!** | /3 |

Задание 2.

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

1. Среди указанных ниже величин найдите пренебрежимо малые (negl), супер-полиномиальные (sup) и полиномиально-ограниченные (poly-b) (в теоретическом смысле):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Задание | Ответ | | |
|  |  | negl | sup | poly-b |
| a |  |  |  |  |
| b |  |  |  |  |
| c |  |  |  |  |
| d |  |  |  |  |
| e |  |  |  |  |
| f |  |  |  |  |
| g |  |  |  |  |
| h |  |  |  |  |
| i |  |  |  |  |
|  | **Не заполнять!** | / 2 | / 2 | / 2 |

1. Пусть – эффективный алгоритм, позволяющий пересказывать следующий бит по битам для некоторого генератора . Т.е. величина не пренебрежимо малая. Определим игру на предсказание предыдущего бита: имя биты ] предсказать бит . (определяется аналогично игре на определение следующего бита). Постройте эффективный алгоритм , позволяющий выиграть игру на предсказание прошлого бита, используя алгоритм . Найдите – преимущество алгоритма в игре на предсказание прошлого бита (определяется аналогично ).

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a |  |  | |
|  | **Не заполнять!** | / 2 | / 2 |

1. Выберите верные утверждения:

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a | Если алгоритм противника в некоторой игре против эффективный, то величина – пренебрежимо малая |  |
| b | Любая пренебрежимо малая – полиномиально ограниченная |  |
| c | Любая полиномиально ограниченная – пренебрежимо малая |  |
| d | Аддитивный одноразовый блокнот переменной длины – семантически стойкий шифр |  |
| e | Пусть алгоритм от параметра . На вход алгоритма подали вход длинной , он детерминированно выполнился за время , не являющимся полиномиально ограниченным от . – точно не эффективный. |  |
| f | Пусть алгоритм от параметра . На вход алгоритма подали вход длинной , он детерминированно выполнился за время , – эффективный. |  |
| g | Любой эффективный алгоритм – полиномиально ограничен памятью |  |
| h | Если и на стойкие PRG, то для (последовательности статистически неразличимы). |  |
|  | **Не заполнять!** | / 8 |

1. Пусть – стойкий PRG. Пусть , где - побитовый AND. Рассмотрим следующий статистический тест на : , где   
    - получает последний бит вектора .

Каково преимущество алгоритма ? ( - ?)

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

1. Пусть – абсолютно стойкий шифр на

Является ли абсолютно стойким шифром? Если нет продемонстрируйте атаку.

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

1. Пусть – стойкий PRG. Какие из следующих алгоритмов является стойкими PRG? Для каждого алгоритма предоставить доказательство стойкости или атаку.

|  |  |  |
| --- | --- | --- |
| № | Задание | Ответ |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |
| f | *-* побитовый OR |  |
| g |  |  |
| h |  |  |
| i |  |  |
|  | **Не заполнять!** | /18 |

1. –шифр на . Пусть имеется возможность случайно выбирать шифртекст равновероятно из . Рассмотрим игру: Противник посылает сообщение претенденту. Претендент вычисляет  и отправляет противнику, который затем вычисляет бит , являющегося результатом игры. Определим . Определим – стойкий шифр с псевдослучайными шифртекстами (pseudo-random ciphertext secure), если для любых противников величина – пренебрежимо малая.

Формально докажите или опровергните утверждения ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Ответ | |
| a | Если – стойкий шифр с псевдослучайными шифртекстами, то он всегда семантически стойкий |  |  |
| b | Одноразовый блокнот - стойкий шифр с псевдослучайными шифртекстами |  |  |
| с | Невозможно построить шифр, который будет семантически стойким, но не стойким с псевдослучайными шифртекстами. |  |  |
|  | **Не заполнять!** | /3 | /3 |

Стойкость симметричный криптопримитивов с ключом основывается на предположении о случайности данного ключа. При этом предполагается, что ключ был получен случайно из равномерно распределённого множества ключей достаточной размерности.

Иными словами, стойкость симметричной криптосистемы требует максимальной энтропии ключа.

В реальности, однако, большинство источников случайности (энтропии) не обладают равномерным распределением, и ключевой материал, полученный из данных источников, не может быть использованы напрямую в качестве симметричных ключей.

Если же необходимо получить симметричный ключ на основе такого ключевого материала используют так называемые Функции Выработки Ключа (Key Derivation Functions, KDF).

В данной работе рассматривается применение KDF к неравномерно распределенному источнику энтропии с целью выработки равномерно распределённых симметричных ключей.

HKDF

KDF состоит их двух подфункций: извлечения (extract) и расширения (expand).

img

img

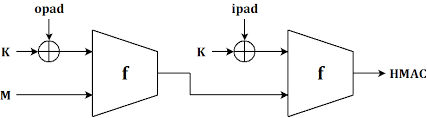
Функция извлечения получает равномерно распределённый случайный ключ, используя неравномерно распределённый ключевой материал.

Функция расширения формирует последовательность ключей на основе одного случайного равномерно распределённого ключа.

HKDF - KDF на основе кода аутентичности HMAC.

Извлечение: PRK <- HMAC(XTS, SKM)  
Расширение: K\_i <- HMAC(PRK, К\_{i-1} CTX, i), если i=1, K\_0 - пустая строка

1. На основе файла [weather.json](https://github.com/CryptoCourse/CryptoLabs/blob/master/Impl/weather.json) построить гистограммы температуры, влажности, скорости ветра, облачности и озонового слоя (выбрать данные по часам). "Толшину" столбца гистограммы выбрать так, что бы было наглядно неравномерное распределение величин. Выбрать одну из указанных величин (или комбинацию величин) в качестве ключевого материала.
2. Реализовать [HMAC](https://en.wikipedia.org/wiki/HMAC) на основе хэш функции SHA-256. В качестве SHA-256 использовать криптографически стойкую реализацию из общераспространённой библиотеке на вашем языке.



img

HMAC(K,C) = H((K + opad) || H((K + ipad) || m))

opad = 0x5c, ..., 0x5c

ipad = 0x36, ..., 0x36

1. Интерфейс функции: byte[] HmacSha256(byte[] key, byte[] data)
2. Реализовать функцию HkdfExtract, которая на основе HMAC, в качестве псевдослучайной функции, соли XTS и ключевого материала SKM) получает ключ PRK для псевдослучайной функции.

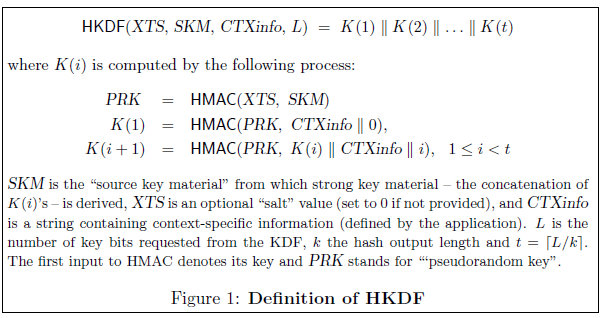
Интерфейс функции: byte[] HkdfExtract(byte[] XTS, byte[] SKM)

1. Реализовать функцию HkdfExpand, которая на основе псевдослучайной функции HMAC, её ключа PRK, контекста CTX, прошлого ключа lastKey и счетчика i получает i-й симметричный ключ. Если прошлого ключа нет передаётся значение null.

Интерфейс функции: byte[] HkdfExpand(byte[] PRK, byte[] lastKey, byte[] CTX, int i)

1. Для i =1..1000 получить 1000 симметричных ключей длины 256 бит на основе HKDF, где data - выбранные данные на шаге 1. В качестве соли использовать случайную равномерно распределённую величину, полученную и использованием криптографического Г(П)СЧ, реализованного в вашем языке. Длина соли XTS - 256 бит.

CTX <- "Ваше имя"  
XTS <- Crypto.Random(256)  
PRK <- HkdfExtract(XTS, data)  
K\_i = HkdfExpand(PRK, K\_{i-1} CTX, i)



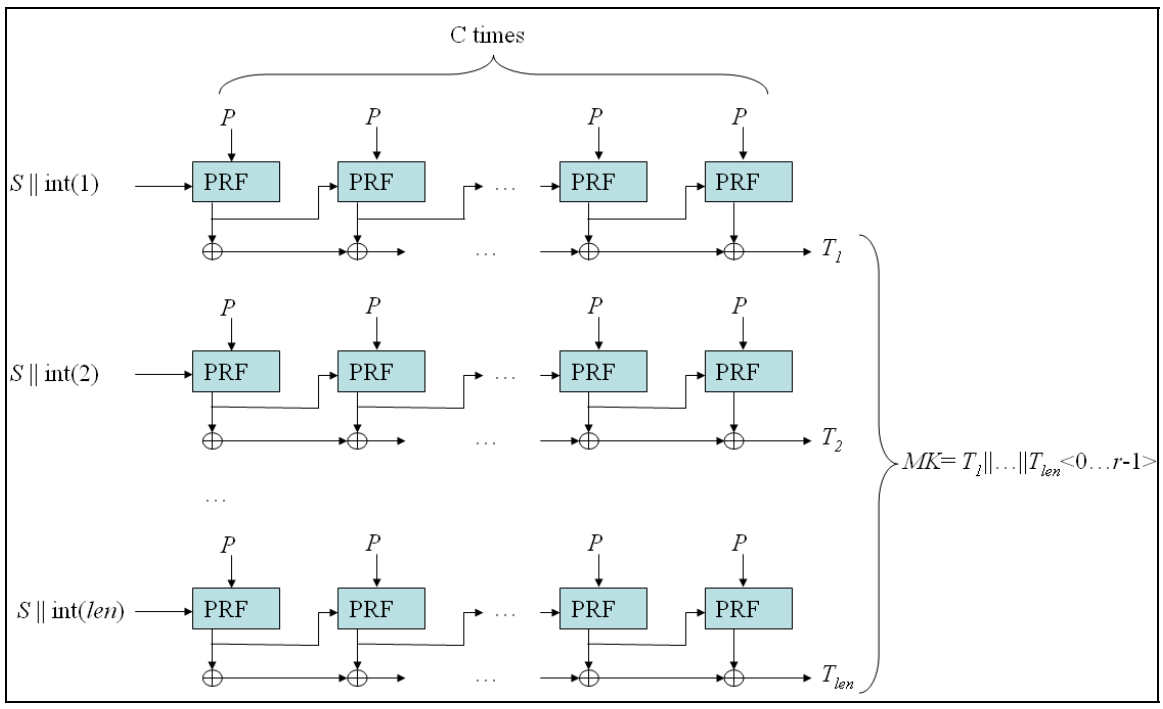
img

1. Убедиться в равномерной распределённости первых 5 бит ключей, построив гистограмму.

PBKDF2

HKDF позволяет получить равномерные данные из неравномерно распределённого источника энтропии. Однако если источник обладает низкой энтропией (например - пароли) необходимо использовать PBKDF2, основным изменением которого является медленное хэширование, необходимое для увеличения сложности перебора ключевого материала. P - пароль пользователя (ключевой материал), S - соль, len = |K|/|HMAC|, |K| - размер ключа, |HMAC| - размер выхода HMAC.

U\_1 = HMAC(P, S||i)  
U\_c = HMAC(P, U\_{c-1})  
F(P,S, c, i) = U\_1 + .... + U\_c, '+' = XOR  
T\_i = F(P, S, c, i)  
K = T\_1 || T\_len



img

1. На основе файла [passwords.json](https://github.com/CryptoCourse/CryptoLabs/blob/master/Impl/passwords.json) построить гистограмму распределения первых 5 бит паролей (кодировка ASCII).
2. Реализовать PBKDF2 с использованием HMAC в качестве PRF, с использованием случайного S. Число итераций 10000.
3. Получить симметричный ключ для каждого пароля длины 512 бит.
4. Убедиться в равномерной распределённости первых 5 бит ключей, построив гистограмму.

Результат работы

**HKDF** Гистограммы (любой из: jpg, pdf, png, xlsx), обоснование выбора данных для ключевого материала, код.

**PBKDF** Гистограмма (любой из: jpg, pdf, png, xlsx), код.

Дополнительные ссылки

https://github.com/CryptoCourse/CryptoLectures/blob/master/Lectures/Lecture11.pdf (стр 33-46)

https://github.com/CryptoCourse/CryptoLectures/blob/master/Lectures/Lecture3.pdf (стр 33-35)

https://en.wikipedia.org/wiki/HMAC

https://tools.ietf.org/html/rfc2104

https://en.wikipedia.org/wiki/PBKDF2

https://tools.ietf.org/html/rfc2898#section-5.2

https://eprint.iacr.org/2010/264.pdf

Задание

Дана REST служба с API указанным ниже.

Задача - детерминировано дешифровать пин-код из шифртекста.

Пин-код длины не более 5, состоит из цифр. Строковое представление пин-кода кодируется с помощью ASCII, дополняется до  
длины 16 байт нулями и шифруется.

**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/IvIsTime
2. Получить зашифрованный пин-код с помощью метода GET <host>/api/EncryptionModeOracle/<userId>/<challengeId>/encryptedpin
3. Получить пин-код, используя метод POST <host>/api/EncryptionModeOracle/<userId>/<challengeId>/noentropy
4. Проверить пин-код, используя метод GET <host>/api/IvIsTime/<userId>/<challengeId>/validate

### 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/IvIsTime

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

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

POST <host>/api/IvIsTime/<userId>/<challengeId>/noentropy

Зашифровывает данные в режиме CBC на фиксированном для задания для задания ключе.

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

GET <host>/api/IvIsTime/<userId>/<challengeId>/encryptedpin

Возвращает зашифрованный пин код, на фиксированном для задания ключе

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

GET <host>/api/IvIsTime/time

текущее время в формате unix time (секунды), используется в качестве IV.

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

GET <host>/api/IvIsTime/<userId>/<challengeId>/validate

Возвращает пин-код, используемый в задании.

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

Здесь и далее используется AES с длиной ключа 128 бит.

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 PKCS5.

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

Аналогично реализовать функцию расшифрования AesBlockDecrypt.

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 режимы ECB, CBC, CFB, OFB, CTR с использованием функции AesBlockEncrypt.

Реализовать интерфейс  
byte[] AesEncrypt(byte[] key, byte[] data, string mode, byte[] iv = null), где

key - байтовое представление ключа блочного шифра.

data - массив байт для шифрования

mode - режим шфирования, допустимые значения ECB, CBC, CFB, OFB, CTR. Может быть задан через Enum на c#.

iv - вектор инициализации или начальное заполнение счётчика в указанном режиме. Если значение не передано, или пуредано значение null (или пустой массив), но режим треюует использования IV или счётчика - значение должно быть сгенерировано (через отдельный метод, с использованием системного криптографически стойкого генератора).

Для генерации ключей и IV использовать стойкие стандартные генераторы. В качестве дополнения использовать PKCS5 padding (дополнение, см [википедия](https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS%235_and_PKCS%237))

**IV должен быть случайным для режимов CBC, CFB, OFB.**

**Выбор начального заполнения счётчика для режима CTR должен быть согласован с** [**rfc3686**](https://tools.ietf.org/html/rfc3686#page-7)**.**

При использовании IV (или счётчика) шифртекст дополняется им с начала сообщения.

c = IV || E(k, m)

2.5 Использовать реализации режима CBC в вашем языке программирвоания для валидации вашей реализации режима CBC.

3. Расшифровать следующие шифртексты:

Режим CBC и CTR. IV = 16 байт. IV добавлен к зашифрованному тексту в начале. PKCS5 padding

CBC key: 140b41b22a29beb4061bda66b6747e14  
CBC Ciphertext 1: 4ca00ff4c898d61e1edbf1800618fb2828a226d160dad07883d04e008a7897ee2e4b7465d5290d0c0e6c6822236e1daafb94ffe0c5da05d9476be028ad7c1d81  
  
CBC key: 140b41b22a29beb4061bda66b6747e14  
CBC Ciphertext 2: 5b68629feb8606f9a6667670b75b38a5b4832d0f26e1ab7da33249de7d4afc48e713ac646ace36e872ad5fb8a512428a6e21364b0c374df45503473c5242a253  
  
CTR key: 36f18357be4dbd77f050515c73fcf9f2  
CTR Ciphertext 1: 69dda8455c7dd4254bf353b773304eec0ec7702330098ce7f7520d1cbbb20fc388d1b0adb5054dbd7370849dbf0b88d393f252e764f1f5f7ad97ef79d59ce29f5f51eeca32eabedd9afa9329  
  
CTR key: 36f18357be4dbd77f050515c73fcf9f2  
CTR Ciphertext 2: 770b80259ec33beb2561358a9f2dc617e46218c0a53cbeca695ae45faa8952aa0e311bde9d4e01726d3184c34451

4. Для кажого режима шифрования зашифровать и расшифровать произвольный текст длины 2,5 блока.

Стойкость MAC к коллизиям определяется исключительно в модели, при секретном ключе.  
При этом стойкость к коллизиям при известном ключе не гарантируется.

В данной лабораторной работе демонстрируется, что использование MAC с фиксированным ключом в качестве  
стойкой к коллизиям хэш функции - плохая идея.

В данной работе рассматривается truncated cbc mac, устроенный следующим образом - данные шифруются с использованием режима CBC,  
в качесте MAC используется половина последнего блока полученного шифртекста. Заметим, что в отличии от raw-CBC конструкции данная схема является стойким MAC. Используется PKCS7 дополнение до размера блока.

**NB**: хотя truncated cbc mac на AES теоретически стойкий, очевидно, что его размер составляет 64 бита, т.е. общая атака даёт лишь  
32 бита стойкости. Данный MAC используется в данной работе для простоты (в идеале нужно использовать другой стойкий MAC, с более сложной  
консрукцией или блочный шифр с большим размером блока). В рамках работы предположить, что 2^32 является  
"неперебираемым" для нас значением. Т.е. не нужно закладываться на то, что мы можем перебрать MAC. Атака должна работать и для  
блочных шифров с большим размером блока.

В качестве блочного шифра используется AES. Ключ в коде аутентичности фиксированный для задания.

В реальной жизни хэш функции часто используются для проверки целостности кода. В частности, в данном задании дана служба, которая  
проверяет целостность данных. Хэш функция должна гарантировать, что если данные были изменены, то изменится и хэш.  
Т.е. (например) имея доступ к возможности модификации исходного кода его нельзя изменить, не изменив хэш. Т.е. контроль хэша обеспечивает  
контроль целостности строк исходного кода. Покажем, что использовать MAC с фиксированным ключом в качестве хэша плохо, так как  
это позволяет модифицировать код, не изменяя при этом результирующий хэш (т.е. изменения невозможно отследить полученным хэшом  
=> это плохая хэш конструкция, как как получили коллизию).

Задание

Дана REST служба с API указанным ниже. Дана строка кода на языке python

self.crypto.security\_parameter = configuration.settings.client.get\_security\_parameter()

Задача - получить строку кода, в которой переменной self.crypto.security\_parameter присваивается значение 32, и при этом  
имеющую такое же хэш значение (в качестве хэша используется truncated cbc mac с фиксированным ключом).

**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.

Ход работы.

### Тестирование

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

<challengeId> = 1

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/CbcMacFixedKey
2. Получить текущий ключ с использованием метода GET <host>/api/CbcMacFixedKey/{userId}/{challengeId}/Key
3. Иодифицировать строку кода как указано в задании (см "Задача" выше), так, чтобы хэш новой строки остался равен хэшу изначальной строки.
4. Проверить, полученный результат с использованием метода POST <host>/api/CbcMacFixedKey/{userId}/{challengeId}/{mac}  
   следующим образом - для исходного и полученного сообщения получить результат "have valid mac" для одного и того же хэш значения.

### Сдача лр

шаги 1 - 4 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/CbcMacFixedKey

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

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

POST <host>/api/CbcMacFixedKey/{userId}/{challengeId}/{mac}

Вычисляет хэш от полученных данных.  
Возвращает строку, содержащую информацию о том, совпадает ли хэш данных с предоставленным хэш значением.

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

GET <host>/api/CbcMacFixedKey/{userId}/{challengeId}/Key

Получает ключ от текущего задания, который используется для вычисления хэш функции.

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

**ВАЖНО!** Перед данной работы рекомендуется выполнить  
[лабораторную на распознание режимов шифрования.](https://github.com/CryptoCourse/CryptoLabs/blob/master/docs/labEncryptionModeDist.md)

Дана REST служба с API указанным ниже.

Служба шифрует указанный пользователем открытый текст, следующим образом:

E(k, random\_padding || user\_data || target\_data||)

random\_padding - случайное дополнение, фиксированное для пары <userId>, <challengeId>

user\_data - данные пользователя (из post запроса)

target\_data - целевые данные, фиксированные для пары <userId>, <challengeId>.

Задача - расшифровать target\_data.

**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, 2, 3, 4, 5

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/EcbDecryption
2. Получить зашифрованное сообщение с помощью метода POST <host>/api/EcbDecryption/<userId>/<challengeId>/noentropy
3. Определить режим шифрования  
   3.1 Если режим шифрования CBC, перейти к следующему <challengeId>.
4. Используя метод POST <host>/api/EcbDecryption/<userId>/<challengeId>/noentropy получить необходимые шифртексты.
5. Расшифровать target\_data
6. Проверить верность ответа использовав метод GET <host>/api/EcbDecryption/<userId>/<challengeId>/verify
7. Проверить корректность программы для <challengeId> = 1..10

### Сдача лр

шаги 1 - 7 этапа тестирования аналогично, но для 15 различных <challengeId>.

Описание API

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

### Описание методов

GET <host>/api/EcbDecryption

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

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

POST <host>/api/EcbDecryption/<userId>/<challengeId>/noentropy

Дополняет полученные данные случайными байтами в начале и в конце. Способ дополнения описан в начале лабораторной.

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 target\_data для указанного задания

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

Здесь и далее в качестве шифра используется AES-128 в режиме CTR. В качестве кода аутентичности HMAC-SHA-256.

Ожидаемый формат шифртекста: CTR(IV) || ENCRYPTED\_DATA || MAC, где MAC вычисляется от (CTR(IV) || ENCRYPTED\_DATA) на ключе, независимом от ключа шифрования.

1. Используя средства языка (реализации AES-CTR и HMAC-SHA-256) реализать класс, реализующий аутентифицированное шифрование в режиме Encrypt-Then-Mac, со следующим интерфейсом:

1.0 Конструктор AuthenticEnryptor(Mode mode)

* инициализирует объект для шифрования или расшифрования, определяемый значением переменной mode.

1.1 void SetKey (byte[] key)

* инициализирует объект шифрования ключом key.

1.2 byte[] AddBlock(byte[] dataBlock, bool isFinal)

* добавляет блок данных для аутентифицированного зашифрования или расшифрования. В случае передачи флага isFinal должен вычислятьсярезультирующий код аутентичности (в зависимости от режима mode). При расшифровании, в случае неуспешной проверки кода аутентичности должна выводиться ошибка. В случае зашифрования данная функция зашифровывает блок данных, и добавляет блок шифртекста для вычисления кода аутентичности, если блок последний - дополняет возвращаемый шифртекст результирующим кодом аутентичности. В случае расшифрования добавляет шифртекст для вычисления кода аутентичности, расшифровывает его, если блок последний - проверяет код аутентичности, содержащийся в блоке с полученным, на основе переданного шифртекста.

1.3 byte[] ProcessData(byte[] data)

Производит шифрованиепереданных данных. При расшифровании, в случае неуспешной проверки кода аутентичности должна выводиться ошибка.

2. Зашифровать и проверить произвольный блок данных, размера 100 MB.

### Коллизии хэш-функций

Стойкость современных хэш-функций часто основывается на предположении об их стойкости к нахождению коллизий.

Пусть H: {0,1}^\* -> {0,1}^n функция, переводящая множество двоичных векторов произвольной длины в множество векторов фиксированной длины n. n называется размеров выхода хэш-функции H.

Хэш-функция H называется стойкой к коллизиям второго рода, если вычислительно сложно найти такую пару (x,y) \in {0,1}^\* : x != y, H(x) = H(y).

Очевидно, что даже для "идеальной" хэш-функции (т.е. хэш-функции, на которую не существуют атак лучше перебора) существуют коллизии, из-за разности мощностей множества прообразов и множества образов хэш-функции.

### Атака на основе парадокса дней рождений

Рассмотрим атаку на основе так называемого парадокса дней рождений.

0. S <- пустое множество.  
1. Выбрать произвольный элемент x из {0,1}^\*  
2. Вычислить значение h\_x = H(x)  
3. Проверить, есть ли в множестве S элемент (y, h\_x). Если есть => goto DONE  
4. Добавить элемент (x, h\_x) в множество S  
5. goto 1  
DONE:  
6. Вернуть пару (x, y) как коллизию хэш-функции.

Данная атака на некоторую стойкую к коллизиям хэш-функцию H: {0,1}^\* -> {0,1}^n имеет вычислительную сложность O(2^(n/2)) и требует O(2^(n/2)) памяти.

Объяснение "на пальцах" почему откуда взялся корень.

**NB!** Это только наглядное объяснение, а не доказательство! Доказательство вводится формально на лекции.

При выборе q-го элемента число пар составляет q(q-1) ~ q^2. При этом вероятность коллизии в каждой паре - 1/2^n (совпадение двух случайных векторов). Хотим вероятность коллизии порядка 1, следовательно q^2/2^n ~ 1, q=sqrt(n).

### Атака на основе метода Полларда

Рассмотрим атаку на основе метода Полларда. Основная идея - уменьшить требования к памяти, используя параллельные вычисления.

Пусть имеется m потоков выполнения программы. Пусть Pi : {0,1}^a -> {0,1}^(a+k) - инъективная функция: x |-> x || 0^k, где 0^k - нулевой двоичный вектор длины k. q=b/2 - log\_2(m). H: {0,1}^\* -> {0,1}^b

Отличительной точкой (distinguished point) назовём такой вектор y, у которого первые q бит - нулевые.

Каждый поток инициализируется уникальным начальным значением y\_0. Начальные значения сохранены в памяти.

Основная идея - каждый поток вычисляет последовательно цепочку хэшей, с применением поверх хэша инъективной функции.

y\_i = Pi(H(y\_{i-1})

Номер текущей итерации для каждого потока (i) также сохранён в памяти.

После чего производится проверка, является ли получено значение отличительной точкой. Если является - проверяем, встречали ли мы такую точку ранее. Если нет - добавляем в множество найденных точек и продолжаем алгоритм.

Если данная точка уже встречалась - значит где-то в построенных цепочках встречались 2 различных прообраза. Для найденной отличительной точки имеем 2 номера итерации i и j. Пусть i>j Находим разность между ними d=i-j.

Применяем d итераций к начальной точке y, соответствующей цепочке большей длины. Далее синхронно итерируем цепочки, пока не найдём коллизию.

Сложность нахождения отличительной точки на одном потоке - O(2^{b/2}), на m потоках - O(2^{b/2}/m) Потребление памяти порядка O(m)

0. S <- пустое множество. Общий доступ для всех потоков.  
0.

Каждый поток выполняет следующий алгоритм:

1. y\_i = Pi(H(y\_{i-1})  
2. Если y\_i != 0^q || yr\_i: goto 1  
3. Если (z\_j, j) : z\_j == y\_i, для любого j находится в S: goto FOUND  
4. Добавить (y\_i, i) в S  
5. goto 1  
FOUND:   
6. Пусть i>j. Вычислить d = i-j  
7. Вычислить y = ((Pi\*H)^d)(y\_0), z = z\_0; где (Pi\*H)(u)=Pi(H(u))  
8. Если H(y) = H(z): вернуть (y,z) как коллизию хэш-функции.  
9. (y, z) = (Pi(H(y)), Pi(H(z)))  
10. goto 8

### Выполнение лабораторной

1. Реализовать "усеченные" хэш функции sha-XX, путём взятия первый XX бит функции sha-256. XX = {15-20}
2. Найти коллизию используя парадокс дней рождений для хэш функций sha-XX.
3. Найти коллизию, используя Ро-метод Полладра для хэш функций sha-XX на двух потоках.

Для пунктов 2 и 3 оценить используемую память для каждого метода. Для каждого из пунктов найти не именее 100 (больше - лучше) коллизий для каждой хэш-функциий. Замерить время выполнения. Построить график зависимости среднего времени выполнения от размера выхода хэш-функции и средних затрат памяти памяти от размера выхода хэш-функции для обоих методов нахождения коллизий. Затраты по памяти можно для простоты оценить приблизительно как количество хранимых хэшей, умноженное на размер хэша + размер прочих хранимых данных в байтах.

### Результат работы:

код, коллизии для хэш-функций (список hex значений), графики.

Доп литература:

https://github.com/CryptoCourse/CryptoLectures/blob/master/Lectures/Lecture11.pdf (стр 7-8, 26-29, 43-46)

https://iamaaditya.github.io/2012/07/the-birthday-paradox-proof/

https://crypto.stackexchange.com/questions/52231/what-is-the-best-and-fastest-algorithm-to-generate-a-hash-collision

https://cr.yp.to/hash/collisioncost-20090517.pdf (стр 9-10)

## Задание

Дана REST служба с API указанным ниже. Аналог службы из реальной жизни - служба, в которой для аутентификации пользователй  
используются зашифрованные токены (cookie).

Токены получаются из имени пользователя по известному детерминированному алгоритму  
(эмулируется методом GET <host>/api/IvIsKey/{userId}/{challengeId}/User/Token/).

Токены администраторов  
получаются по другому известному алгоритму (эмулируется методом GET <host>/api/IvIsKey/{userId}/{challengeId}/Admin/Token/).  
После получения токенов они зашифровываются на симметричном ключе в режиме CBC с  
IV = key. Зашифрованный токен передаётся пользователю.

Используя зашифрованный токен пользователь может аутентифицироваться  
через метод GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/user/encryptedtoken={encryptedToken}, передавая  
токен. Для аутентификации администраторов служит метод  
GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/admin/encryptedtoken={encryptedToken}.

Задача - получить от метода GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/admin/encryptedtoken={encryptedToken}  
строку "Wellcome to secretNet!", т.е. успешно пройти аутентификацию как администратор.

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

## Ход работы.

### Тестирование

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

<challengeId> = 1

1. Проверить работоспособность контроллера с помощью метода GET <host>/api/IvIsKey
2. Используя методы API, собственный код и помощь тёмных богов получить зашифрованный токен администратора.
3. Аутентифицироваться как администратор, используя метод  
   GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/admin/encryptedtoken={encryptedToken}

### 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/IvIsKey

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

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

## GET <host>/api/IvIsKey/{userId}/{challengeId}/User/Token/

Возвращает сырой (незашифрованный) токен для пользователя в виде строки. Ответ не кодируется в BASE64.

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

## GET <host>/api/IvIsKey/{userId}/{challengeId}/Admin/Token/

Возвращает сырой (незашифрованный) токен для администратора в виде строки. Ответ не кодируется в BASE64.

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

## GET <host>/api/IvIsKey/{userId}/{challengeId}/User/encryptedToken/hex

Возвращает зашифрованный токен для пользователя в виде hex строки. Может быть использован для аутентификации в  
методе GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/user/encryptedtoken={encryptedToken}.  
Ответ не кодируется в BASE64.

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

## GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/user/encryptedtoken={encryptedToken}

Аутентифицирует пользователя по зашифрованному токену. В случае некорретности токена возвращает ошибку.  
Ответ не кодируется в BASE64.

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

## GET <host>/api/IvIsKey/{userId}/{challengeId}/authenticate/admin/encryptedtoken={encryptedToken}

Аутентифицирует администратора по зашифрованному токену. В случае некорретности токена возвращает ошибку.  
Ответ не кодируется в BASE64.

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