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

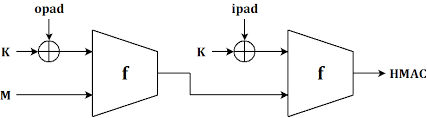
Функция извлечения получает равномерно распределённый случайный ключ, используя неравномерно распределённый ключевой материал.

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.

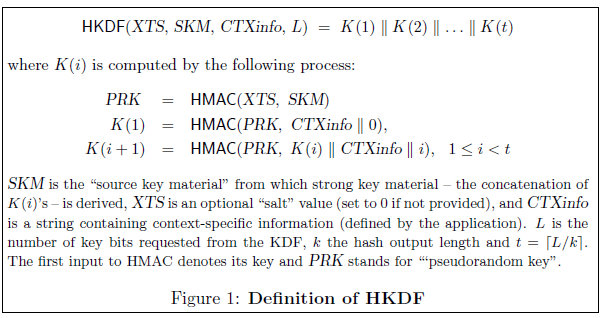
Интерфейс функции: byte[] HkdfExtract(byte[] XTS, byte[] SKM)

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)



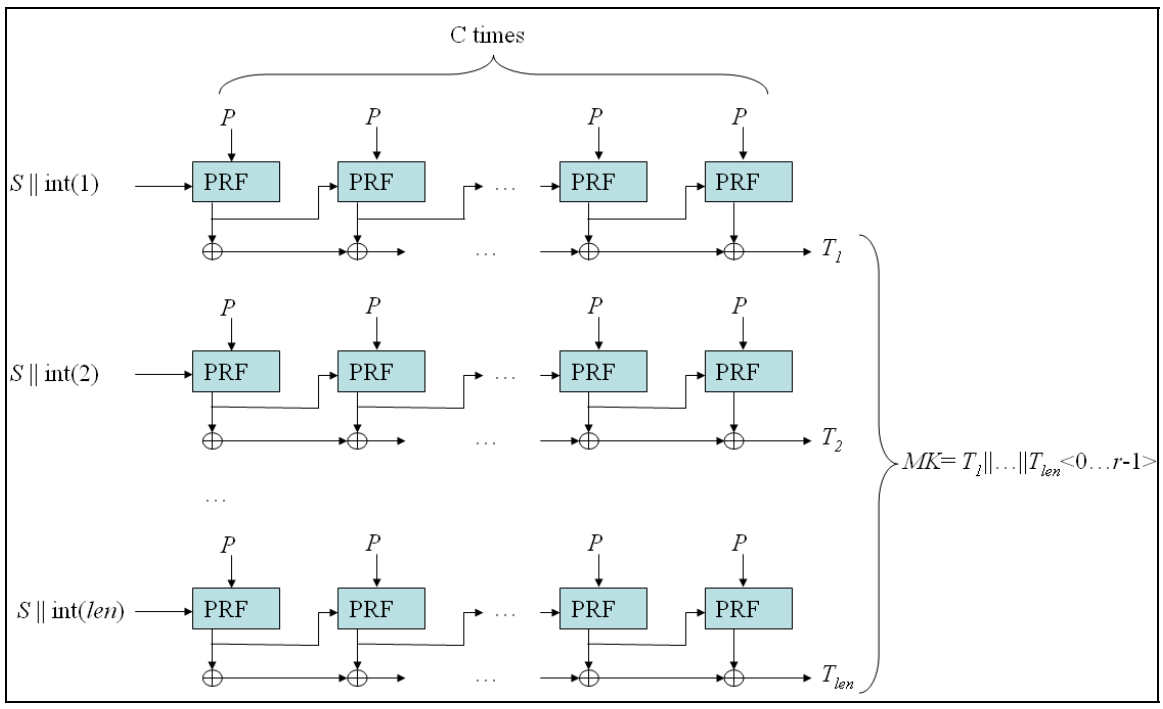
img

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