KDF

Макаров Артём

МИФИ 2021

HMAC

Уберём независимость ключей k_1 и k_2:

- $k_1 \leftarrow k \oplus ipad$
- $k_2 \leftarrow k \oplus opad$

Где

- $ipad = (0\text{x}36, 0\text{x}36, \ldots, 0\text{x}36)$
- $opad = (0\text{x}5\text{c}, 0\text{x}5\text{c}, \ldots, 0\text{x}5\text{c})$

Итого:

$$HMAC(k, m) = H(k \oplus opad || H(k \oplus ipad || m)$$

Стойкость как PRF показана в модели стойкой PRF h при связанных ключах (related key attack PRF, RKA-PRF), что в свою очередь может быть доказано в модели идеального шифра (не вводили в данном курсе).

HMAC

- Де-факто интернет стандарт
- Не требует блочного шифра для реализации, основан на хэш-функции
- Используется во множестве протоколов
- Самый распространённых MAC
- Может быть построен с использованием произвольной хэш-функции (включая ГОСТ)
- В настоящий момент используется HMAC-SHA-256
- Лучше избегать использование HMAC-SHA-1, хотя в настоящий момент не известны практические атаки, существенно лучше перебора

Получение ключей

Пусть имеется единственный ключ SK, полученный из:

- Аппаратного датчика случайных чисел
- Протокола распределения ключей (передача или согласование ключа)
- Пароль пользователя

Хотим получить сессионные ключи из данного ключа.

Используются KDF – key derivation functions

Если источник ключей имеет равномерное распределение

$F: K \times X \rightarrow \{0,1\}^n$ - PRF

KDF(SK, CTX, L) :=

F(SK, (**CTX || 0**)) || F(SK, (**CTX || 1**)) || ⋯ || F(SK, (**CTX || L**))

CTX: контекст, строка, уникально представляющая приложение или назначению ключей (для независимой генерации различных ключей для различных приложений).

Если источник не имеет равномерное распределение

Напомним – PRF стойкая, только если ключи – случайные и равномерно распроданные.

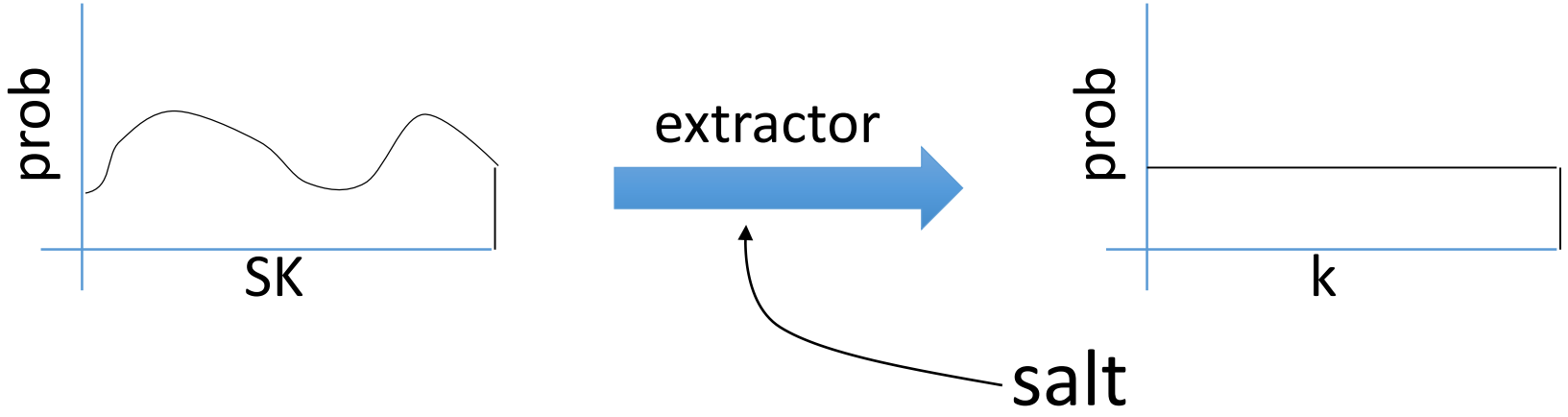
SK не равномерно распределён ⇒ PRF может и не давать случайно выглядящий выход

Примеры неравномерного распределения:

- Протоколы обмена ключей: ключ может быть равномерно распределён только в некотором подмножестве ключей
- Аппаратный PRG: возможен смещённый выход
- Пароли: …. без комментариев

Парадигма извлечения и расширения

Step 1: **извлечь** псевдослучайный ключ k из ключа SK



salt: некоторая величина

step 2: **расширить** k используя в качестве ключа PRF (как указано ранее)

HKDF: KDF from HMAC

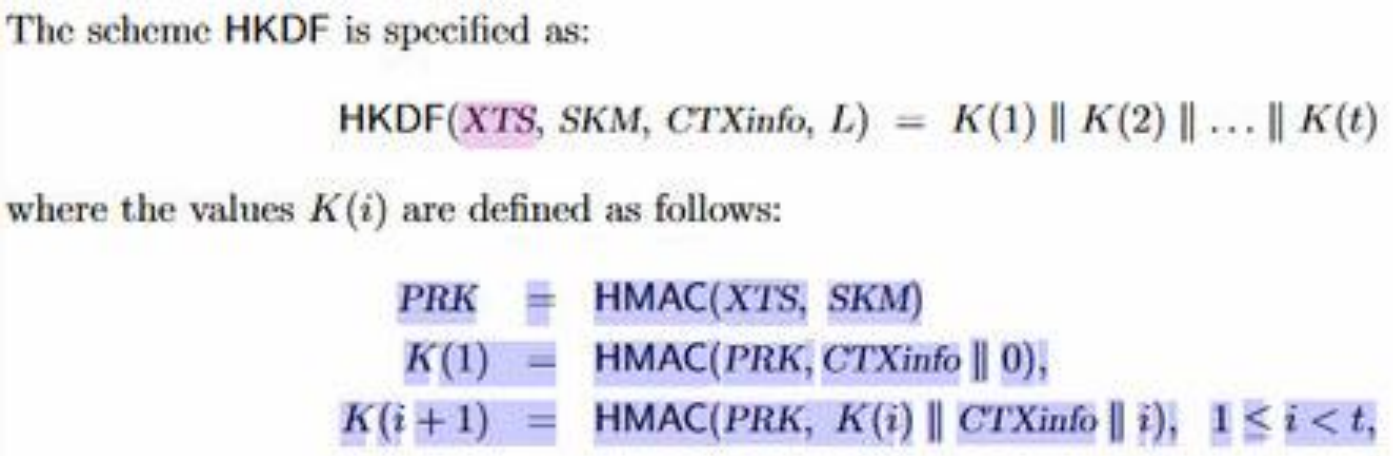
Реализует Парадигма извлечения и расширения с помощью HMAC :

- извлечение: use **k ⟵ HMAC(salt, SK)**

- Затем расширить используя HMAC в качестве PRF с ключом **k**

HKDF

Более формально



The scheme HKDF is specified as:

$$\mathsf{HKDF}(XTS, SKM, CTXinfo, L) = K(1) \parallel K(2) \parallel \ldots \parallel K(t)$$

where the values $K(i)$ are defined as follows:

$$\begin{aligned} PRK &= \mathsf{HMAC}(XTS, SKM) \\ K(1) &= \mathsf{HMAC}(PRK, CTXinfo \parallel 0), \\ K(i+1) &= \mathsf{HMAC}(PRK, K(i) \parallel CTXinfo \parallel i), \quad 1 \leq i < t, \end{aligned}$$

XTS – соль, SKM – ключевой материал, $K(i)$ – выходы

HKDF

The scheme HKDF is specified as:

$$\mathrm{HKDF}(XTS, SKM, CTXinfo, L) = K(1) \parallel K(2) \parallel \ldots \parallel K(t)$$

where the values $K(i)$ are defined as follows:

$$\begin{aligned} PRK &= \mathrm{HMAC}(XTS, SKM) \\ K(1) &= \mathrm{HMAC}(PRK, CTXinfo \parallel 0), \\ K(i+1) &= \mathrm{HMAC}(PRK, K(i) \parallel CTXinfo \parallel i), \quad 1 \leq i < t, \end{aligned}$$

XTS – случайная => PRK – стойкая PRF (т.к. HMAC – стойкая PRF), требование на хэш-функцию – необходимые требования для стойкости MAC

XTS – константа 0. PRK – детерминированная хэш-функция, на основе хэш-функции, использованной в HMAC. Для получения псевдослучайных выходов $K(i)$ необходимо использовать модель случайного оракула для хэш-функции.

Промежуточные решения – использование различных констант, счётчиков, nonce, дают промежуточные результыты.

HKDF

- Позволяет извлекать энтропию из неравномерно распределённого источника для получения равномерно распределённой последовательности
- Контекст используется для изоляции ключей между приложениями или применениями
- Соль может быть константной или отсутствовать, но случайная соль даёт лучшую стойкость. Если не получается использовать случайную – лучше использовать хотя бы счётчик
- Де-факто интернет стандарт
- Не использовать MD-5 и SHA-1

KDF для паролей (PBKDF)

- Не использовать HKDF: у паролей удивительно малая энтропия
- Полученные ключи могут быть уязвимы к перебору по словарю исходного материала

PBKDF: **salt** и **медленное хэширование**

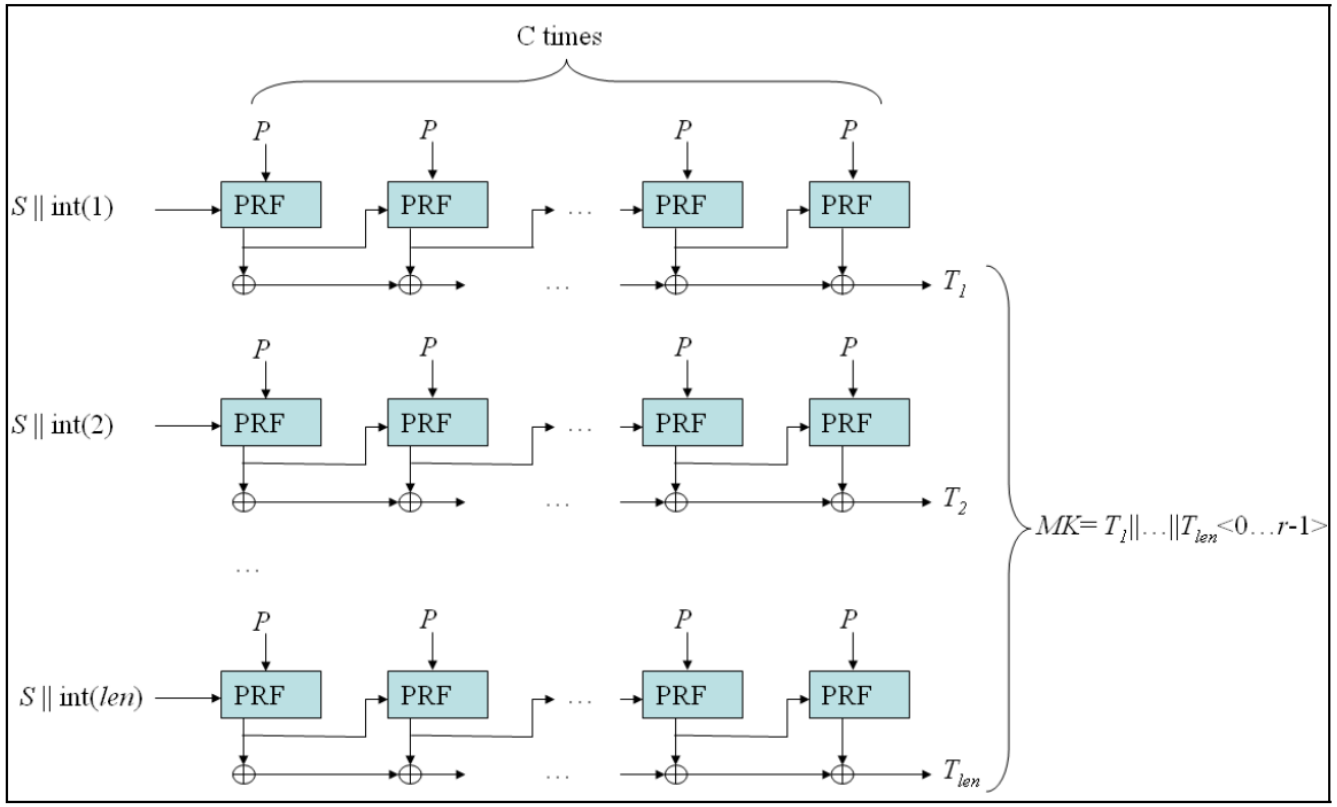
Пример: **PKCS#5** (PBKDF1)

- $H^{(c)}$(pwd ll salt): вычисляем хэш-функцию с раз, подмешивая соль

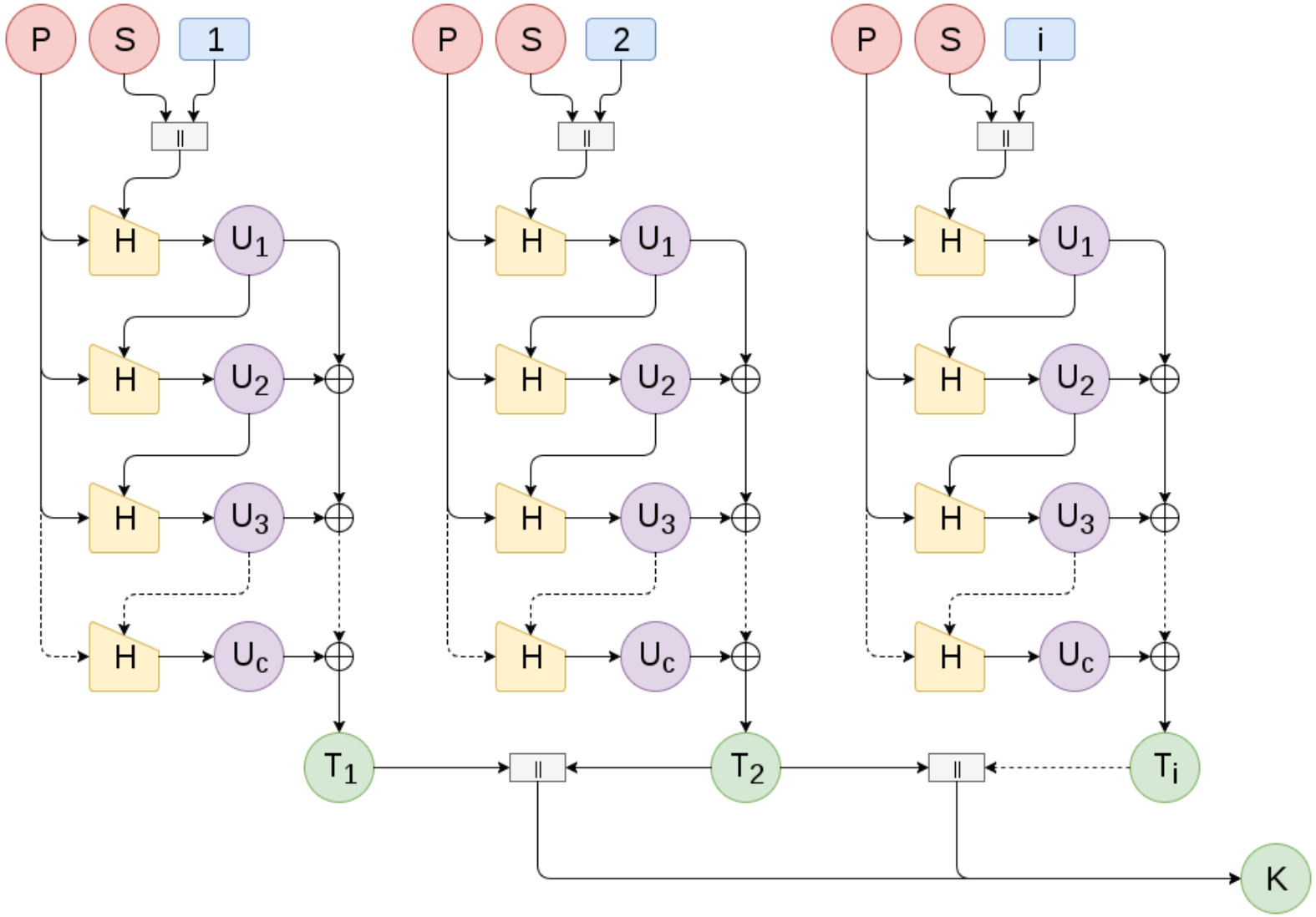
Пример: PBKDF2

- P – пароль, S – seed, c – число итераций, n – число T блоков для генерации одного ключа

- $U_1 = PRF(P, S||i)$
- $U_c = PRF(P, U_{c-1})$
- $F(P, S, c, i) = U_1 \oplus \cdots \oplus U_c$
- $T_i = F(P, S, c, i)$
- $DK = T_1|| \ldots ||T_n$ (MK на картинке)



PBKDF2, ещё одна картинка



PBKDF2, порядок перебора

	Master Password guessing times with hashcat's 4 GPU system		
	10000 PBKDF2 iterations (minimum for new keychains)	25000 PBKDF2 iterations (typical for new keychains)	45000 PBKDF2 iterations (high end)
	300,000 guesses/sec	120,000 guesses/sec	66,667 guesses/sec
Password Strength Entropy (in bits)			
39	9 days	23 days	41 days
52	193 years	482 years	867 years
65	1,498,426 years	3,746,064 years	6,742,915 years
78	12 billion years	29,129 billion years	52,433 billion years
90	91 trillion years	227 trillion years	408 trillion years