










# Схема получения криптокошельков

## Описание сущностей:





### User:

-  Каждый пользователь имеет свой уникальный **приватный** мастер ключ. На основе этого ключа пользователь может генерировать бесконечное количество приватных ключей и к ним в пару столько же публичных ключей - своих уникальных адресов(уникальных среди всех пользователей и среди адресов самого пользователя)
-  Каждый адрес привязывается к своему токену, либо может быть не привязан вообще ни к чему(null address). Но на каждый токен указывает ровно один адрес.
-  Адреса группируются в бакеты по  $r$  штук, причем бакеты статические(они один раз формируются из  $r$  адресов и потом их наполнение не изменяется).
-  Заполнение бакета из адресов происходит сверху вниз, то есть в бакете поддерживается нумерация  $1, 2, \dots, r$  и если адрес с номером  $s$  указывает на токен, то  $\forall i \in \{1, \dots, s - 1\}$  адрес с номером  $i$  также подвязан к токену. Но может быть, что адреса с большими номерами - null address.
-  Взаимодействию пользователя и сервера может происходить через разные браузеры и приложение поэтому пользователь не может запоминать никакой информации о своих прошлых запросах.
-  Пользователю запрещено использовать криптографический хеш

### Server:

-  Сервер хранит БД с мапингом адресов и токенов. И может еще хранить какую-либо вспомогательную информацию. Но не хранит никаких данных о пользователях.
-  Сервер может синхронизировать какую-либо информацию с пользователями, например хеш-функции, их количество, какие-либо константы и проч и проч.
-  Сервер получает все обновления о новых токенах и привязанных адресах.

### Attacker:

-  Злоумышленник пытается узнать какие токены принадлежат пользователю. Его цель сопоставить адреса токенов и конкретных пользователей, которым они принадлежат.
-  Канал связи сервера и пользователя не безопасен вся информация переданная от пользователя серверу и наоборот от сервера пользователю может быть перехвачена злоумышленником.
-  Злоумышленник имеет полный доступ к базе данных сервера и может просматривать(но не менять) любую информацию от туда.
-  Злоумышленник может однозначно определить от какого из пользователей исходил конкретный запрос к серверу и наоборот какому пользователю сервер направил ответ.

## Постановка задачи:

Пользователь имеет бакет адресов и хочет выяснить какие из адресов в этом бакете все еще являются свободными(null address), а какие занятые. Для этого он должен направить информацию о бакете с адресами

на сервер и получить от сервера какую-то часть БД, которая однозначно содержит строки: адресс-токен для каждого используемого адреса из бакета пользователя.

### і Замечания:

- Так как пользователь, на свой запрос, получает от сервера набор адрессов и токенов, но личность пользователя нельзя однозначно ассоциировать с адрессами, сервер подмешивает туда мусорные строки адреса-токены. Но ни сам сервер не злоумышленник не должны знать какие из этих строк принадлежат пользователю, а какие нет.
- Также, если пользователь направил серверу запрос с адрессами конкретного бакета и получил ответ содержащий строки  $(a_1, t_1), (a_2, t_2), \dots, (a_c, t_c)$ , где  $a_i$  - адрес,  $t_i$  - токен связанный с адресом  $a_i$ , то при следующих запросах от пользователя этого же бакета ответ сервера обязан содержать (в качестве подмножества)  $(a_1, t_1), (a_2, t_2), \dots, (a_c, t_c)$ . Иначе злоумышленник по анализу нескольких запросов пользователя сможет сузить выборку адрессов принадлежащих пользователю.
- По информации которой пользователь направляет серверу должно быть **невозможно** определить что именно шифровал пользователь (односторонняя функция).
- При запросе бакета состоящего из  $r$  строк -  $d$  связанных с токеном адресов и  $o = r - d$  (null address), пользователь должен получать ответ где математическое ожидание процента адресов пользователя равно  $p$  %, то есть математическое ожидание размера ответа сервера должно быть  $\frac{100}{p} * d$  элементов.

### ☛ Идея:

1. До начала работы сервер и пользователь синхронизируют  $k$  хеш функций, каждая из которых выдает число в диапазоне от  $0 \dots m - 1$ .
2. Основной идеей шифрования бакета адрессов будет - битовая маска состоящая из  $m$  ячеек. По каждому из  $r$  адресов из бакета будет вычисляться  $k$  хеш функций. Далее генерируется число  $l = l(N)$ , и потом  $l$  псевдо случайных числе из того же диапазона. (в данном случае псевдослучайность означает лишь, то что от одного бакета можно сгенерировать много\* числе из данного диапазона причем, эти числа генерируются по заранее заданному алгоритму зависящему от бакета, то есть зная бакета нельзя угадать последовательность чисел). По номеру каждого из вышеперечисленных чисел ставится 1 в битовой маске и 0 иначе.
3. Эту битовую маску пользователь отправляет серверу. Сервер пробегается по массиву всех адресов и вычисляет свою битовую маску для каждого из адресов, все теми же хеш функциями. После чего сервер проверяет является ли эта маска подмножеством пользовательской по единицам (то есть стоят ли в маске пользователя 1 на месте где стоят они в маске текущего адреса), и если является возвращает этот адрес пользователю.
4. Пользователь получает множество адресов от сервера из которых лишь какой-то процент  $p$  его, а остальные попали случайно. И цель дальнейших рассуждений добиться фиксированного матожидания  $p$  для любого запроса любого пользователя.

### ⚡ Алгоритм

#### Приведем следующий алгоритм запроса:

Бакет пользователя состоит из массива  $address = \{\#a_1, \dots, \#a_r\}$  - адресов

До первого запроса  $lmin = 0$

1. Пользователь запрашивает размер БД- $N$  и  $lmin$
2. Считает  $l = find\_l(N, n)$ , где  $find\_l(N, n)$  - функция которая вычисляет кол-во псевдо случайных единиц, которые нужно добавить в битовую маску, для достижения нужного процента выдачи ложноположительных адрессов. А именно  $find\_l(N, n) = \frac{\ln(1 - (\frac{n}{N})^{\frac{1}{k}})}{\ln(1 - \frac{1}{m})} - rk$ , где  $n$  число адрессов, которые мы хотим получить от сервера на запрос.

3.  $l = \max(l, \text{min})$

4. Построим блум из бакета:

```
for a in address:
    for i in range(k):
        hash = hash[i](a) % m
        bloom[hash] = 1
    g = genaraoor(address)
    for i in range(l):
        bloom[g.next() % m] = 1
```

5. Отправить серверу *bloom*

6. Получить от сервера ответ содержащий  $n_0$  твоих и  $\sim n$  всего адресов. Обновить *lmin* на сервере:

```
temp = find_l(N, n_0)
if lmin != 0:
    lmin = min(lmin, temp)
else:
    lmin = temp
```

## ✂ Корректность

Сначала найдем `find_l`:

Верна следующая формула:  $p = (1 - (1 - \frac{1}{m})^{rk+l})^k$ , а также мы хотим, чтобы в момент когда в БД находилось  $N$  элементов,  $p = \frac{n}{N}$  из этого следует, что:  $1 - (\frac{n}{N})^{\frac{1}{k}} = (1 - \frac{1}{m})^{rk+l} \Rightarrow \ln(1 - (\frac{n}{N})^{\frac{1}{k}}) = (rk+l)\ln(1 - \frac{1}{m}) \Rightarrow l = \frac{\ln(1 - (\frac{n}{N})^{\frac{1}{k}})}{\ln(1 - \frac{1}{m})} - rk$  - а это и есть `find_l(N)`

Первый запрос мы делаем с параметром  $\text{lmin} = 0$ , а сатло быть мы просто запрашиваем от сервера  $\sim n$  адресов. Далее если  $n_0 = r$  тогда мы направляем на сервер запрос, что бакет заполнен и направляем размер ответа который нам пришел от сервера и при всех следующих запросах мы отсекаем от ответа сервера лишь верхушку равную по размеру этому числу. Тем самым мы ограничиваем рост лишних адресов. Если же  $n_0 < r$  тогда в силу выбора *lmin* и того, что он не уменьшается, вне зависимости от следующих запросов мы будем вместе с первыми  $n_0$  адресами бакет получать фиксированный набор адресов в несколько раз больших самого  $n_0$  и тем самым мы защищены от пересечений ответа.

## ■ Требования:

1. Синхронизация хеш функций пользователем и сервером до начала общения.
2. Хранение на сервере/либо у пользователя данных о каждом бакете, а именно *lmin*, *t* - максимальный размер ответа, который пользователь хочет получить от сервера.

## Реализация

🔗 [https://github.com/Vladimir119/critpo\\_wallet](https://github.com/Vladimir119/critpo_wallet)