Схема получения криптокошельков

Цель:

Отправить от пользователя серверу шифрованное сообщение бакета с адрессами кошельков, что бы получить от сервера информаию о своих кошельках, так что бы сервер сам незнал, какие кошельки из его ответ принадлежат пользователю.

Важно

- Кошельки соответствующие адрессам, запрашиваемым пользователем, должны присутствовать в ответе сервера.
- Сервер не должен знать какие именно кошельки из ответа принадлежат пользователю, запрашивающему кошельки. Поэтому он подмешивает в ответ кошельки других пользователей.
- Подмешиваемые кошельки должны должны быть не изменяемыми при неизменяймом запросе. То есть если пользователь 2 раза запришивает одни и теже аддреса, то он получает 2 одинаквых набора кошельков.

Идея:

Пользователь и сервер синхранизирует набор из k хеш функций. Сервер хранит блум фильтры полученные хешированием адресов с помощью синхронизированных хеш функций. Каждому адресу соответствует свой блум фильтр.
Пользователь хочет запросить сразу пачку из r адресов. Он логически суммирует все фильтры блума соответствующие этим адресам(операция OR), после чего подмешивает туда еще случайно l еденицы(случайность подрузумевается в том смысле, что он делает один раз случайно, а позже запоминает растоновку случайно добавленных еденицы и всегда использует их).Пользователь для одного бакета лишь раз считает l и фиксирует l еденицы, далее(при последующих запросах) он использует уже насчитанные значения l и позиции еденицы. Помимо того, l псевдо случайно добавленных еденицы могут накладыаться на уже стоящие еденице в блуме, мы ничего с этим не делаем, просто оставляем уже стоящую еденицу, и перехождим к следующей псевдо случайной еденицы. По итогу у пользователя сформируется битовая маска сопоставленная каждому из его бакетов с адресами(каждая маска формируется в момент первого запроса этого бакета к бд). Именно через эти маскм пользователь будет общаться с сервером.
Когда пользоателю нужно получить информацию по адресам в бакете, он сначала запрашивает текущий размер БД(кол-во токенов в бд/кол-во адресов по всем пользователям), с помощью него он считает параметр l, далее он формирует битовую маску и отправляет ее серверу. Сервер в свою очередь сперва отправляет пользователю размер БД, а после того как он получат битовую маску. Он с помощью индексов ополучает из БД маски всех аддресов которые являются подмножеством маски пользователя и возврощает ему часть этого, а именно столько сколько возвращал в превый раз(так как после увеличения БД мы кол-во масок которые являются подмножеством нашей растет.)

Подсчет параметров

n	кол-во токенов, которые я хочу получить от сервера.
N	кол-во токенов в базе данных на момент запроса.
m	размер блум фильтра(битовой маски).
p	вероятность ложно положительного срабатывания (имеется ввиду вероятность, хешированный адрес какого-то токен будет лежать в подмножестве нашей маски)

Тогда верна следующая формула: $p = (1 - (1 - \frac{1}{m})^{rk+l})^k$, а также мы хотим, чтобы в момент когда в ДБ находилось N элементов, $p = \frac{n}{N}$ из этого следует, что: $1 - (\frac{n}{N})^{\frac{1}{k}} = (1 - \frac{1}{m})^{rk+l} \Rightarrow ln(1 - (\frac{n}{N})^{\frac{1}{k}}) = (rk + l)ln(1 - \frac{1}{m}) \Rightarrow l = \frac{ln(1-(\frac{n}{N})^{\frac{1}{k}})}{ln(1-\frac{1}{m})} - rk(*)$

Тем самым когда пользователь подготавливает маску он запрашивает N и на основе нее подсчитывает параметр l (по формуле $(*)$) после чего к своим просуммированным хешированным адресам в бакете, он рандомно генерирует индексы из $uniform(0, m-1)$ и добавляет на эти места 1, если их там еще не было. Так он делает l раз. Тогда матож возвращаемых из бд результатов будет n элементов.
Важно, пользователь запоминает количество елементов которые сервер вернул при первом запросе этого бакета и направляет это количество серверу вместе с маской, сервер в свою очередь находит те маски, которые являются подмножеством нашей и возвращает только верхнюю часть найденных, размер которой равен тому самому числу(ответу на первый запрос).

Заметим что

Заметим, что l - убывающая функция с ростом N тем самым, так как это кол-во еденицы которые мы добавляем, максимальный размер БД который мы можем зацепить достигается при $l = 0 \Rightarrow ln(1 - (\frac{n}{N})^{\frac{1}{k}}) = r * k * ln(1 - \frac{1}{m}) \Rightarrow (1 - (\frac{n}{N})^{\frac{1}{k}}) = (1 - \frac{1}{m})^{rk} \Rightarrow (1 - (1 - \frac{1}{m})^{rk})^k = \frac{n}{N} \Rightarrow N = \frac{n}{(1-(1-\frac{1}{m})^{rk})^k}$ Осталось подобрать параметры k и m для максимизации размера допустимой БД, или просто посмотреть, чтобы достигалось какое-то недостигаемое в реальности значение N, например $1e10$

Статистика

Посчитаем параметры для некоторых значениях параметров m, n, k, r
Например если взять $n = 1000$, $k = 22$, $m = 5000$, $r = 100$ тогда $l \approx 600$ даже при $N \approx 1e11$.
в колонке l мы указываем значение l при $N \approx 1e10$

n	k	m	r	l
1’000	22	5’000	100	1e3
1’000	10	5’000	100	1e2
1’000	9	5’000	100	13
1’000	<9	5’000	100	отрицательное число
1’000	50	10’000	100	≈ 7000
1’000	90	10’000	100	≈ 9000

Тесты

Я реализовал небольшие тесты на python и проверил матожидание ответа. Статистика появится позже. Результаты можно найти в репозитории:
`https://github.com/Vladimir119/critpo_wallet`