Randpay

Подсистема массовых микроплатежей криптовалюты Emercoin

В статье рассмотрена подсистема агрегации платежей на основе лотерейного протокола с нулевым доверием. Показана её применимость для действительно массовых взаиморасчётов (триллионы операций в год). В протоколе получатель платежа создаёт уникальный платёжный адрес и задаёт пространство возможных платёжных адресов, в которое этот адрес входит. Плательщик выбирает случайный адрес из заданного пространства и создаёт транзакцию платежа на выбранный адрес, после чего передаёт транзакцию получателю платежа. Получатель, в случае совпадения адреса из полученной транзакции с ранее созданным, подписывает транзакцию и публикует её в сеть, что приводит к перечислению платежа. В случае несовпадения адресов, транзакция является недействительной и игнорируется. Типичным сценарием Randpay будет малая вероятность осуществления реального платежа в каждом конкретном акте «игры в лотерею», что тем не менее при большом количестве таких платежных актов приводит к справедливым изменениям баланса для всех участников. В статье представлена математическая модель, доказывающая статистическую сходимость Randpay для всех участников к справедливым суммам, которые они бы имели отправленными или полученными без использования этой подсистемы. Кроме того, применение Randpay сокращает транзакционные издержки для плательщиков, что делает её применение экономически оправданной, особенно для микроплатежей.

Проблема масштабирования

Децентрализованные платёжные системы, построенные на технологии blockchain, приобретают всё большую популярность. Оптимисты и крипто-энтузиасты даже утверждают, что скоро все банковские системы станут ненужными, так как хозяйствующие субъекты будут использовать Биткоин или иные криптовалюты для прямых взаиморасчётов. Однако блокчейн-системы имеют изначальное ограничение масштабируемости, вытекающее из их архитектуры, а именно – блокчейн должен содержать все транзакции, которые когда-либо были сделаны. Если взять для примера индустрию телекоммуникаций, и представить, что каждый телефонный звонок завершается взаиморасчётом (settlement), то несложно подсчитать, что при размере транзакции 200 байт (обычно больше), 2 триллиона звонков мирового телекома в год потребуют ежегодного увеличения размера блокчейна на 400 терабайт, что сразу же делает такое решение неприемлемым. Да, читатели могут возразить, что есть механизмы уменьшения размера транзакций, тот же Segwit, уменьшающий размер почти вдвое. Но даже 200 терабайт в год вместо 400 – всё равно непреодолимый барьер для практического применения криптовалют в телекоме (если проводить платёж за каждый звонок), да и в других областях применения, требующих массовых платежей.

Уже сейчас, даже когда криптовалютные платежи ещё не вошли в массовый обиход, блокчейны основных криптовалют уже занимают сотни гигабайт, и держать полную ноду криптовалюты

становится весьма накладно. Несложно представить, что будет, когда криптовалютные платежи действительно станут массовыми.

Также следует заметить, что любой узел такой распределённой системы получает в режиме реального времени все транзакции всех участников сети, и при большом потоке транзакций могут возникнуть проблемы, связанные с ограничениями пропускной способности сети.

Нельзя сказать, что проблема не осознаётся разработчиками криптовалют. Время от времени появляются в крипто-прессе заявления о том, что «эта криптовалюта будет иметь пропускную способность, сравнимую с VISA» или идеи о «шардинге», то есть разделения базы блокчейна между участниками. Но пока что не видно ничего работающего, и даже на уровне прототипа ничего не представлено. Тем не менее, мы желаем этим разработчикам успехов, и чтобы заявленное стало реальностью.

По нашему мнению, действительно работающая система, решающая проблемы масштабирования, должна быть основана на идее агрегирования платежей, то есть замены большого количества микроплатежей на небольшое количество итоговых, которые и попадут в блокчейн. Также заметим, что при агрегации транзакционные платежи (tx fee) надо платить только за агрегирующие транзакции. Иными словами, посредством агрегации достигается не только масштабирование платёжной системы, но и снижение транзакционный издержек в расчёте на единичный платёж.

Ниже мы сосредоточимся на том, что уже реально существует, работает и может быть применено здесь и сейчас.

Lightning Network

Система агрегирования платежей Lightning Network представляет собой отдельную сеть агентов, связанных платёжными каналами. Два агента могут создать канал, связав в «транзакции открытия канала» свои монеты, потом вести взаимозачёт только друг с другом, и в конце концов закрыть канал, отправив в блокчейн «транзакцию закрытия канала». Таким образом, в блокчейне на жизненный цикл канала создаются только две транзакции (открытия и закрытия), а взаимозачёты происходят непосредственно между агентами, и на блокчейн-сеть не влияют. Такая система имеет следующие преимущества и недостатки:

Преимущества

- Высокая скорость транзакций – не требуется ждать подтверждений блокчейна.
- Абсолютная точность платежа – получатель в каждом акте платежа получает точную сумму, ему предназначенную.

Недостатки

- Требуется создание и обслуживание процессинговой инфраструктуры – сети операторов каналов, которые всегда online.
- Возможно, операторы каналов захотят получать плату за свои услуги по передаче денег внутри своих каналов, как это делают платёжные системы типа VISA/MC.
- Платежи возможны только в пределах сети каналов – невозможно отправить платёж «на любой адрес», как это сейчас происходит в криптовалютах.

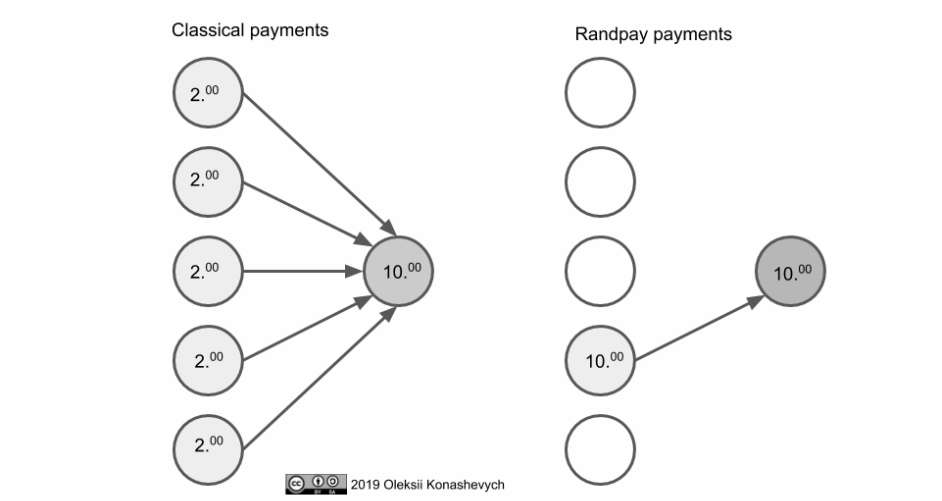
- В случае выхода из строя оператора канала, или же преднамеренного отказа в обслуживании – платёж становится невозможным.
- При создании канала необходимо связать в нём какую-то сумму монет, и в случае некооперативного поведения контрагента или отказа в обслуживании – эта сумма может оказаться заблокирована на долгий срок (возможно, годы).
- Возможен отказ в обслуживании вследствие исчерпания денег в канале. Например, при распродажах или появлении модного товара, который многие захотят быстро купить.
- Lightning – протокол с состоянием. То есть канал надо создать, и помнить где-то его последнее состояние, и забыть о нём можно только после закрытия. В случае же потери состояния обоими контрагентами – связанные в канале деньги могут стать утерянными.

Таким образом, Lightning Network по целям и структуре подобна системе процессинга платежей типа VISA/MC, которая связывает вместо банков простых обладателей криптокошельков.

Randpay

Система Randpay для агрегации платежей использует вероятностный подход, и основана на идее лотерейного платежа, предложенной Роном Райвестом, автором широко известных криптосистем RC4, RSA и других.

Идея



Суть идеи состоит в том, чтобы оплату проводить не платёжной транзакцией которая со 100% вероятностью попадает в блокчейн, а лотерейным билетом, который попадёт в блокчейн только в случае “срабатывания” - то есть проигрыша для плательщика и выигрыша для получателя платежа. В последнем случае только выигравшие лотерейные билеты попадают в блокчейн в виде транзакций. Пустые билеты останутся неиспользованными, и в блокчейн не попадут, и их можно не хранить далее. При этом для каждого акта платежа лотерейным билетом математическое ожидание переведённой суммы равно сумме, которая была бы отправлена обычной платёжной

транзакцией. При таком подходе к агрегации, суммы платежей (выигрышных билетов) будут достаточно большими, но сами платежи вносятся в блокчейн намного реже.

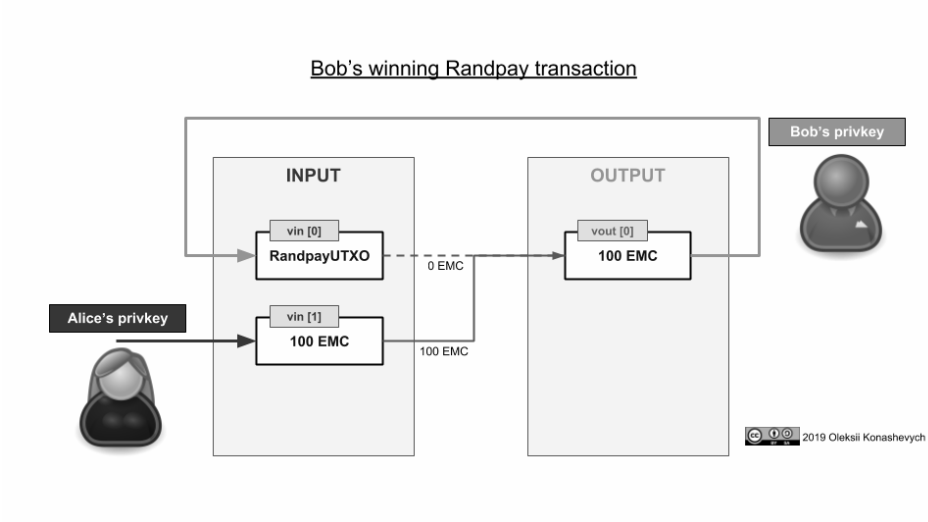
Пример

Пусть надо платить за телеком-сервис $0.01 в минуту. Но вследствие транзакционных издержек и ограничений криптовалюты такие платежи нерентабельны, а в большом количестве — невозможны. При использовании лотерейного метода, клиент (плательщик) платит сумму гораздо большую - скажем, $100, но с вероятностью выигрыша 1/10000. Таким образом, в среднем, за один акт платежа производится усреднённый платёж суммы $0.01 в минуту.

То есть, клиент (плательщик) как бы играет в "русскую рулетку" -- в револьвере 9999 гнёзд барабана пусты, и одно гнездо содержит платёж. В акте платежа, если платёж не проходит (вероятность 9999/10000), то клиент выигрывает и получает минуту сервиса бесплатно. Если же платёж проходит (с вероятностью 1/10000), то выигрывает сервер (тот, кто предоставляет товар или услугу), и транзакция на $100 записывается в blockchain.

Таким образом, возможны ситуации, когда клиент получает полностью бесплатную сессию. Или же - когда за 1 минуту переплачивает много денег. Но в среднем, при регулярном использовании сервиса (не обязательно от одного и того же поставщика), значение платежей стремится к справедливой сумме, которая получалась бы при реальной поминутной оплате. При этом, понятно, количество реальных коин-транзакций также уменьшается в 10000 раз (для данного примера), и пропорционально снижаются транзакционные издержки.

Таким образом, если использовать Randpay в том же телекоме с указанными в данном примере параметрами, то settlement всемирного телекома за 2 триллиона звонков в год при использовании Randpay добавит к блокчейну только 40Gb, что хоть и немало, но вполне приемлемо для практического использования. А повысив коэффициент агрегации (например, при снижении поминутной ставки ниже $0.01), можно снизить инфляцию блокчейна ещё больше. Тем более, что и другие способы борьбы с раздуванием блокчейна, такие как Segwit и оптимизатор транзакций никто не отменял.



Неточность

Понятно, что в силу случайности процесса платежа Randpay, реальная сумма, собранная получателем платежа есть сумма выигравших лотерейных билетов. И она будет отличаться от справедливой суммы, которая была бы, если бы реальные платежи шли поминутно. Но по мере предоставления сервиса, полученная сумма будет расти линейно по мере предоставления сервиса, тогда как отклонение – будет нарастать как квадратный корень. То есть, по мере активного использования сервиса относительная ошибка будет стремиться к нулю. Ну а в бухгалтерских документах, если надо, эту неточность можно вписывать в графу наподобие «операционные сюрпризы». Или просто писать реальную сумму.

Рассмотрим преимущества и недостатки системы Randpay по сравнению с Lightning Network:

Преимущества

- Не требуется создание и обслуживание отдельной платёжной сети операторов каналов. Для работы с Randpay нужен только обычный кошелёк Emercoin, и ничего более.
- Соответственно, нет платежей операторам такой сети.
- Соответственно, нет элементов ненадёжности, связанных с функционированием таких операторов.
- Транзакции типа «каждый с каждым», без ограничений – как в криптовалюте сделано изначально.
- Актуальная (усреднённая) сумма платежа в каждом акте может быть ниже минимальной денежной единицы криптовалюты (Сатоши).
- Деньги не связываются в каналах на месяцы и годы. Обычно в течение минуты становится ясно, билет выиграл или нет, и плательщик может переиспользовать деньги одного несработавшего билета в другом.
- Отсутствует проблема отказа в обслуживании вследствие исчерпания денег в канале.
- По сравнению с двумя транзакциями открытия и закрытия канала в Lightning, Randpay отправляет в блокчейн единственную транзакцию, что делает систему вдвое эффективнее при том же коэффициенте агрегации.
- Randpay – протокол без состояния. То есть не надо устанавливать канал или другие финансовые взаимоотношения с контрагентом, помнить состояние канала, и корректно его закрывать. То есть «заплатил и забыл».
- Специального завершения платежа протокол также не требует, что особенно полезно при ненадёжных линиях связи. Любой контрагент может в любой момент отключиться, без ущерба для платежей.

Недостатки

- Для надёжности платежа, требуется ждать подтверждений транзакции закрытием блоков, как и в обычных платежах криптовалютой. Работа без подтверждений возможна, но требует дополнительных мероприятий для защиты от мошенничества. См раздел «Безопасность» ниже.
- Реальная сумма платежа будет отличаться от «справедливой», но по мере использования сервиса – стремиться к ней.

Идея протокола и наивный алгоритм

Протокол Randpay, хотя и является оригинальной разработкой, основан на идеях двух ранее известных криптографических протоколов:

- Эзотерический протокол «Подбрасывание монеты по телефону».
- Протокол аутентификации CHAP.

В протоколе Randpay сервер (получатель платежа) задаёт клиенту «загадку» (challenge), а тот должен предоставить ответ-решение. Протокол не основывается на каком-либо доверии между клиентом и сервером, а загадка состоит в запросе "выбери адрес". Алгоритм протокола следующий:

1. Получатель платежа создаёт новый платёжный адрес P2PKH и соответствующий ему приватный ключ.
2. Далее получатель формирует требование лотерейного билета (загадку адреса addrchap) - пространство возможных адресов, один из которых – адрес [1], для которого он имеет приватный ключ. Этот адрес - выигрышный для получателя (и проигрышный для плательщика). Получатель передаёт сформированное требование плательщику напрямую, то есть без участия криптовалюты или блокчейна.
3. Плательщик формирует случайный платёжный адрес в заданном пространстве, и создаёт транзакцию-платёж на этот адрес (лотерейный билет), после чего отправляет транзакцию (ответ) получателю платежа в частном порядке.
4. Получатель платежа, получив транзакцию, валидирует её, и в случае корректности транзакции - предоставляет клиенту товар или услугу.
5. Если плательщик не угадал адрес, то есть сформировал платёж на адрес, для которого у получателя платежа нет приватного ключа, то деньги для получателя недоступны, и он вынужден отвергнуть транзакцию-платёж, то есть плательщик получает свой товар или сервис бесплатно. Если же плательщик угадал адрес - то получатель публикует транзакцию в сеть, тем самым забирая деньги плательщика.

Интересное отличие Randpay от других CHAP-протоколов состоит в том, что плательщику выгодно выбрать адрес, которым не владеет получатель платежа, и тем самым сохранить свои деньги (то есть «не угадать»), тогда как в классических CHAP-системах клиент стремиться предоставить верный ответ («угадать»).

Исходя из эгоистических соображений, получатель должен обеспечить равномерное распределение выигрыша в пространстве адресов, а плательщик – равновероятный случайный выбор одного из возможных адресов предложенного пространства. Именно в этом случае средняя сумма выплаты соответствует математическому ожиданию, то есть является справедливой. Попытка же любого контрагента как-либо манипулировать распределением может привести к тому, что другой контрагент разработает стратегию, эксплуатирующую данное отклонение в свою пользу. Например, если получатель платежа всегда создаёт чётные адреса в фазе [1], то плательщик, узнав (или предположив) такое положение дел, может всегда выбирать нечётные в фаза [3], и тем самым никогда не платить. Аналогичные рассуждения можно сделать и для случая, когда плательщик пытается как-то «оптимизировать» своё угадывание. В этом случае

получатель платежа сможет специально задавать такую загадку, чтоб плательщик угадывал как можно чаще. Таким образом, равновесие Нэша достигается именно при соблюдении заданного в протоколе равномерного распределения обеими сторонами, а попытки манипуляции распределением любой из сторон не только не приносят пользы, но могут ещё и принести ущерб манипулирующему.

Другие же варианты мошенничества, как-то меньшая сумма в транзакции, неподписанная транзакция, использование уже потраченных UTXO или использование адреса вне заданного пространства – легко детектируются, и получатель платежа может прекратить взаимодействие с клиентом-мошенником.

Рассмотренный здесь наивный алгоритм работоспособен, но имеет существенный недостаток, который делает его мало применимым для практического использования. Недостаток заключается в том, что получатель платежа может вести себя некооперативно, а именно – в фазе [5] всегда отправлять транзакцию в сеть, вне зависимости от того, угадал плательщик адрес, или нет. Да, при этом получатель не сможет получить доступ к деньгам из транзакции, ибо у него (да и ни у кого другого) нет приватного ключа для траты этих денег. В этом случае деньги оказываются потерянными навсегда. Иными словами, некооперативный получатель деньги себе забрать не может, но может сделать вред плательщику, уничтожив его деньги, пусть и без пользы для себя.

Реальный алгоритм

Для предотвращения рассмотренного выше некооперативного поведения получателя, реальный алгоритм в целом похож на наивный, но содержит механизм, блокирующий такое деструктивное поведение. Идея механизма состоит в том, что получатель платежа при публикации транзакции в сеть вынужден доказывать, что он владеет приватным ключом от адреса, на который направлен платёж, то есть плательщик угадал реально существующий адрес из загадки. Вот как это устроено:

В сети (в каждом кошельке), в множестве не-потраченных выходов (UTXO set) имеется специальный фиктивный выход номер 0 от несуществующей транзакции с TXID=ECECECECECECECECECECECECECECECECECECECECECECECECECECECECECECECECECECECEC.

Этот специальный выход, далее называемый RandpayUTXO, обладает следующими свойствами:

- Его можно тратить многократно, то есть после траты он не отмечается как потраченный (прямо неразменный пятак из «Понедельника»).
- С него можно потратить только 0 монет, то есть ничего (эх, фальшивый пятак).
- Этот UTXO во входах транзакции может присутствовать только единожды. Далее назовём такой вход randpay-in.
- Этот randpay-in подписывается приватным ключом от адреса vout[0] текущей транзакции. То есть актуальный адрес платежа для scriptPubKey для RandpayUTXO при проверке и для формирования подписи берётся из скрипта vout[0] текущей транзакции.

Когда плательщик создаёт Randpay транзакцию в фазе [3], он в массив входов vin вносит ещё одну трату с фиктивного RandpayUTXO, то есть добавляет вход randpay-in. При этом подписывает все входы транзакции, кроме этого. Его потом подпишет получатель, если сможет. Угадываемый адрес платежа плательщик помещает в vout[0] формируемой транзакции.

Получив выигрышную транзакцию (плательщик угадал адрес), перед отправкой в сеть в фазе [5] получатель должен подписать её вход randpay-in приватным ключом от адреса, на который он собрался получать деньги, то есть от адреса, специфицированного в vout[0]. Таким образом, получатель доказывает, что он владеет приватным ключом от адреса vout[0], и имеет право получить платёж, то есть - публиковать транзакцию в сеть.

Если же плательщик не угадал адрес, то получатель не может подписать randpay-in, так как не имеет приватного ключа от vout[0]. И сеть просто не примет неподписанную транзакцию. В результате, получатель вынужден просто проигнорировать транзакцию, и вреда плательщику нанести не сможет.

Математика

В отличие от Lightning и других платёжных систем, протокол Randpay имеет дополнительную степень свободы - коэффициент агрегации. Он определяет, сколько платежей «сворачиваются» в реальную платёжную транзакцию, отправленную в блокчейн. В примере выше коэффициент агрегации 10000. Иными словами, 10000 платежей по $0.01 в среднем сворачиваются в одну транзакцию на $100 (далее эту сумму обозначим *amount*). Соответственно, во столько же раз растёт и сумма этой реальной транзакции. Мы назвали этот коэффициент агрегации словом ***risk***, так как он мультиплицирует сумму, которой рискует плательщик в акте платежа Randpay. Так, в нашем примере, плательщик рискует суммой в $100, которую у него изымут с вероятностью, обратной риску:

$$p = \frac{1}{risk}$$

Соответственно, математическое ожидание суммы для единичного платежа:

$$E = p \cdot amount$$

Упрощая модель предположим, что все транзакции в серии имеют одинаковые параметры *(p, amount)*. Упрощение корректно, так как платежи Randpay взаимно независимы, и их можно сгруппировать по данным параметрам, после чего применить нижеследующие рассуждения к каждой группе отдельно.

Так как статистически последовательность платежей Randpay подчиняется биноминальному распределению, то статистически значимое количество платежей **n** серии можно аппроксимировать нормальным распределением:

$$Bin(n, p) \approx N(np, npq)$$

С математическим ожиданием:

$$E = np$$

И дисперсией:

$$\sigma^2 = npq$$

Где $q = 1 - p$

Тогда среднеквадратичное отклонение (ошибка) реально уплаченной или полученной суммы от справедливой:

$$\sigma = \sqrt{npq}$$

А относительная ошибка (отношение отклонения к сумме, проплаченной за n платежей):

$$E_{error} = \frac{\sigma}{E} = \frac{\sqrt{npq}}{np} = \sqrt{\frac{npq}{n^2p^2}} = \sqrt{\frac{q}{np}}$$

Здесь легко заметить, что при ***n*** стремящемся к бесконечности, относительная ошибка стремится к нулю:

$$\lim_{n\to\infty} \sqrt{\frac{q}{np}} = 0$$

Иными словами, при постоянном использовании Randpay, средние суммы множества платежей будут стремиться к тем же значениям, которые имели бы место при классических платежах, когда в блокчейн вносится каждая транзакция. Но так как в блокчейн попадает далеко не каждая транзакция, то соответственно и он растёт меньше в ***risk*** раз, и соответственно снижаются актуальные транзакционные издержки:

$$blockchain\_inflation = \frac{tx\_size}{risk}$$

$$avg_{fee} = \frac{fee}{risk}$$

Казалось бы, увеличивая значение параметра ***risk***, легко повысить эффективность системы чуть ли не до бесконечности, так как с ростом риска снижается как инфляция блокчейна, так и транзакционные издержки. Но тут возникает ограничение для увеличения ***risk***, так как оно ведёт к автоматическому увеличению суммы транзакции ***amount***, и снижению количества транзакций. И могут возникнуть проблемы:

- У плательщика просто может не оказаться достаточно большой суммы, чтоб создать транзакцию лотерейного платежа.
- Увеличение суммы создаёт стимул плательщику отозвать платёж мошенническими методами, например, создав двойную трату (double spend) в случае выигрышного билета.
- Плательщику может быть некомфортно, пусть и с низкой вероятностью, но сильно переплатить. Например, заплатить $10000 за услугу с ценой $0.01.
- В системе должно пройти несколько реальных платежей за отчётный период. Ибо если из-за большого значения ***risk*** компания-продавец будет получать в среднем один платёж в 10 лет, это скорее всего приведёт к краху её бизнеса.

Иными словами, система ограничивает увеличение риска «вверх», параметром ***amount*** – суммой, которой готов рискнуть плательщик. Но интересно то, что система не ограничивает движения

«вниз», то есть позволяет увеличивать практически неограниченно параметр ***risk*** для сверх-дешёвых платежей. Например, можно представить бизнес, который продаёт информацию о текущих ценах акций на бирже (stock quotes) поштучно, скажем по цене $0.00000001 за квоту. Или же – ответы системы ИИ на какие-либо вопросы. Или – статьи с новостных сайтов, выборки из каких-либо баз данных, и тп. Это делает рентабельными бизнес-модели, которые в других платёжных системах были бы просто невозможны вследствие высоких транзакционных издержек по сравнению с уплачиваемой суммой.

Протокол взаимодействия агентов

Рассматриваемый здесь протокол общего вида является рекомендацией взаимодействия между плательщиком и получателем. Реализация остаётся на усмотрение разработчиков системы, использующей этот или подобный протокол. Подчеркнём – эти рекомендации относятся именно к взаимодействию агентов между собою, а не между агентом и узлом Emer. Последнее будет рассмотрено ниже, в спецификации Randpay API.

1. **WANTPAY(double amount, uint32_t risk)** [Этот шаг опционален, но весьма желателен]
   Этим запросом клиент (плательщик) сообщает серверу (получателю платежа), что он хотел бы сделать randpay-платёж на сумму ***amount***, с вероятностью срабатывания ***p = 1 / risk***. То есть актуальная усреднённая сумма однократного платежа ***amount / risk***. Например, при WANTPAY(100, 100000) клиент сообщает, что готов заплатить 100EMC с вероятностью 1/100000, то есть актуальный платёж будет на 0.001EMC. Клиент формирует сумму и вероятность исходя из содержимого его кошелька и готовности нести соответствующий риск от единовременной выплаты большой суммы. В среднем, клиенту выгодно увеличивать параметр risk, так как доля TX FEE в платеже от этого снижается. С другой стороны - он ограничен суммой, которую необходимо блокировать на время "лотерейной игры" и риском выплаты этой суммы.

2. **NEEDPAY(double amount, uint32_t risk, char [] addrchap)**
   Этим запросом, в ответ на WANTPAY, сервер сообщает, что хотел бы получить платёж на сумму ***amount***, с вероятностью ***p = 1 / risk***, для чего предлагает клиенту угадать адрес из пространства адресов, формируемых парой (risk, addrchap). Параметры (amount, risk), передаваемые сервером клиенту, могут отличаться от предложенных клиентом в WANTPAY. Это возможно потому, что сервер решил, что сумму ему слишком большую предлагают при низкой вероятности (высокий риск), или же его не устраивает актуальная сумма платежа, или ещё какие-то причины.

3. **PAYMENT(uint32_t risk, char[] rawtx)**
   Клиент посылает серверу Randpay-транзакцию, соответствующую условиям NEEDPAY. Сервер её отправляет на проверку в кошелёк, и на основании ответа кошелька (если не находит мошенничества, сумма достаточна), то предоставляет товар или услугу клиенту. В случае, если клиент угадал адрес (с вероятностью ***p = 1 / risk***), кошелёк автоматически подписывает транзакцию и публикует её в блокчейн Emercoin.

Randpay API

Данный API реализован как расширение JSON HTTP Emercoin API, и обеспечивает генерацию и обработку Randpay-транзакций узлом Emer. С помощью этого API пользовательская программа взаимодействует с узлом (нодой) Emer, и производит или получает платёж. В случае корректной работы функций API, их возвращаемые значения указаны в спецификации ниже. В случае некорректных входных данных, попытки мошенничества или ошибки исполнения – возвращается стандартный код ошибки с сопутствующим текстовым сообщением.

randpay_createaddrchap (uint32_t risk, int timeout)

Эта функция ноды Emer, вызываемая получателем платежа, создаёт пространство возможных адресов для плательщика размером ***risk***. Результат используется для этапа протокола NEEDPAY.

Для указанного параметра ***risk*** создаёт пару приватный-публичный ключ, и из публичного ключа формирует addrchap – пространство «сырых» адресов для плательщика:

$$addrchap = \frac{hash160(Pubkey)}{risk}$$

Ничего не вносит ни в кошелёк (wallet.dat), ни в блокчейн, но сохраняет в ОЗУ пару:

addrchap -> Privkey

на интервал времени ***timeout*** секунд. Эта пара пригодится получателю при приёме Randpay-платежа, в вызове randpay_submittx.

Возвращаемое значение:

addrchap - 160 бит в hex

Получившийся addrchap получатель передаёт плательщику на этапе NEEDPAY.

randpay_createtx (double amount, char[] addrchap, uint32_t risk, int timeout, bool naive=false)

Эта функция, вызываемая плательщиком, создаёт транзакцию (лотерейный билет) на сумму ***amount*** на случайный адрес получателя, который формирует из ***addrchap***. Использованные в транзакции входы блокируются на интервал времени ***timeout***, порядка 30..60 секунд. Блокировка подразумевает, что в течение ***timeout*** те же самые UTXO не будут использованы в другом платеже из этого же кошелька. За это время сервер должен либо провести платёж и забрать деньги, либо же деньги станут доступны для другого платежа. Опциональный параметр naive позволяет создать наивную randpay-транзакцию, как рассмотрено выше. Тогда транзакция будет короче на один вход. Наивная randpay-транзакция может иметь смысл при отладке, или же если плательщик точно уверен, что получатель будет вести себя кооперативно.

Алгоритм этой функции:

- Распаковываем addrchap из hex в бинарный вид.
- Формируем адрес (сырой) из (risk, addrchap):

  *uint160 pay_addr = risk * addrchap + GetRand(risk);*

- Создаём транзакцию-платёж на адрес rand_addr. При этом обязательно платёж на созданый выше адрес помещаем в vout[0].
- Использованные входы временно блокируем, чтобы случайно не переиспользовать в другом платеже.
- Подписываем все входы транзакции.
- Если транзакция полноценная (не наивная), то создаём вход randpay-in, внеся в vin[0] трату с RandPayUTXO: EC...EC:0

Возвращаемое значение:

- Randpay_tx_hex - hex raw transaction

Результат этой функции – платёжная транзакция Randpay (лотерейный билет). Плательщик её передаёт получателю на этапе протокола PAYMENT.

randpay_submittx(char[] Randpay_tx_hex, uint32_t risk)

Здесь нода получателя платежа вначале верифицирует Randpay-транзакцию, после чего, используя сохранённую ранее пару addrchap -> Privkey, сравнивает, угадал ли плательщик адрес. Если адрес совпал (плательщику не повезло), то функция:

- Переносит Privkey в кошелёк, чем обеспечивает получателю возможность дальнейшей траты полученных денег.
- Подписывает randpay-in (для не-наивной транзакции) и отправляет транзакцию в сеть.

Функция предусматривает специальное значение параметра ***risk = 0***. Это специальный случай для транзакций от лёгкого клиента. Он подразумевает, что требуется валидировать транзакцию, и если randpay-in подписан – отправить её в сеть. При этом Emer-нода не использует пары addrchap -> Privkey, ничего не подписывает и не переносит в свой кошелёк никакого Privkey, так как предполагается, что эти операции лёгкий кошелёк будет делать сам.

Возвращаемые значения:

- amount – сумма из vout[0]
- won – булевый флаг, сработал ли лотерейный билет

Randpay URI интерфейс

Кроме рассмотренного выше интерфейса API, кошелёк Emercoin содержит механизм отправки Randpay-платежей посредством вызова его через URI внешним приложением (например браузером), подобный механизму Bitcoin BIP21, который, к слову, Emercoin также поддерживает. Этот механизм позволяет интернет-сайтам через браузер пользователя запрашивать микроплатежи в Emercoin, сформировав на странице соответствующий URI. Пользователь кликает линк, и кошелёк пользователя (после подтверждения, конечно) тут же производит оплату сайту через механизм Randpay.

Этот интерфейс может быть применён сайтами для продажи доступа к статьям или медиа-контенту (видео, музыкальным произведениям, и тп). Например, сайт может запрашивать 0.001

EMC за доступ к статье, создав Randpay-запрос с параметрами amount=10&risk=10000. Пример URI для такого запроса:

emercoin:randpay?amount=10.0&chap=00deadbeef&risk=10000&submit=http%3A%2F%2Frandpay.news.com%3Fid=777%26article=666

Рассмотрим структуру Randpay URI подробнее. Его общий вид:

emercoin:[//]randpay?amount=DOUBLE&chap=chap_hex&risk=INT[&timeout=INT]&submit=CALLBACK_URI

Здесь в квадратных скобках указаны элементы URI, которые сайт может не указывать. Параметры URI этого интерфейса (**amount, chap, risk, timeout**) соответствуют таковым в вызове randpay_createtx().

- **amount** — сумма платёжной транзакции. Её заплатит плательщик, если билет выиграет.
- **chap** — сформированное получателем (сайтом) требование лотерейного билета в hex-представлении.
- **risk** — величина, обратная вероятности выигрыша.
- **timeout** — интервал времени в секундах, на который блокируются выходы UTXO, участвующие в транзакции.

Параметр **submit** содержит callback-URI. Туда кошелёк пользователя отправит сформированную Randpay-транзакцию (лотерейный билет) в hex-кодировке посредством HTTP POST. Содержащиеся в этом параметре специальные символы, зарезервированные в rfc3986 (например «/:&»), необходимо перекодировать в «процентное представление», для избежания конфликта параметров оригинального URI и callback-URI.

Пользователь кошелька может указать в файле **emercoin.conf** параметры, управляющие поведением этого интерфейса (после знака «=» приведены дефолтные параметры):

- **rp_max_amount**=0 - максимальная сумма в транзакции, которую можем слать без подтверждения, EMC.
- **rp_max_payment**=0 - максимальная сумма платежа (матожидание **rp_max_amount / risk**), которую можем слать без подтверждения, EMC.
- **rp_timeout**=30 — время блокировки входов, использованных в randpay-транзакции, сек.
- **rp_submit**=false — автоматический выбор кнопки «Submit» в окне подтверждения платежа. Дефолтный выбор — CANCEL, то есть отмена платежа.

Безопасность

Ниже рассмотрены возможные сценарии мошенничества плательщика при использовании Randpay, и применённые механизмы борьбы с ними. Даны рекомендации получателю платежа для минимизации возможных потерь от мошенничества.

Атаки на подсистему Randpay можно разделить на две группы – неверное формирование транзакций с целью заплатить меньше или не заплатить вообще, либо же атака на консенсус блокчейн-сети путём отмены транзакции через двойную трату (далее по тексту DS, double spend).

Атаки на транзакцию:

Рассмотренные здесь сценарии атак неуспешны при использовании Emer Randpay, так как текущий код содержит механизмы предотвращения таких атак. Тем не менее, они здесь рассмотрены в образовательных целях.

1. Попытка сформировать транзакцию не с тем ***risk***, который указан в сопровождении транзакции.
   Приводит к неверной распаковке addrchap в *submittx*, детектируется, возвращается код ошибки. Вероятность же попадания распакованного addrchap на чужой такой же - пренебрежимо мала, и её можно не принимать во внимание. И даже если это вдруг и произойдёт - получатель получит деньги, что тоже неплохо.
2. Попытка передать меньшую сумму, чем просили.
   Обнаруживается при анализе ***amount*** в возврате *submittx*.
3. Попытка заплатить в транзакции несуществующими или уже потраченными монетами.
   Обнаруживается на этапе валидации в *submittx*. Возвращается код ошибки.
4. Попытка предъявить в качестве входов адреса, принадлежащие получателю, и через сдачу украсть деньги получателя.
   Обнаруживается на этапе валидации в *submittx*. Возвращается код ошибки.
5. Попытка переиспользовать чужую подпись для RandpayUTXO, взятую из ранее опубликованной кем-то транзакции.
   Бессмысленна, так как транзакцию и адрес отправки создаёт плательщик, и ему просто невыгодно использовать randpay-in с найденной где-то подписью. С тем же успехом он может создать наивную транзакцию без randpay-in, и избавить получателя от доказательства владения адресом.

Атаки на консенсус:

Транзакция подсистемы Randpay подвержена такой же уязвимости к атаке на консенсус посредством двойной траты (double spend, далее DS), как и обычная платёжная транзакция. В этом смысле она не лучше и не хуже обычной. И наиболее эффективный способ бороться с атаками на консенсус – ждать достаточного числа подтверждений блокчейна, как изначально и предусмотрено в любой блокчейн-системе. Но в некоторых случаях (например, при продаже минут разговора) нет времени ждать подтверждений, и приходится работать без них. В таких ситуациях Randpay немного лучше классического механизма, так как решение о выигрыше и публикации транзакции в сеть принимает получатель платежа, то есть он имеет некую фору. Ниже мы рассмотрим несколько сценариев атаки на консенсус посредством DS на Randpay-транзакции без подтверждений.

1. Плательщик формирует DS, отправляет Randpay-платёж, и тут же отправляет DS в сеть. В результате возникает логическая гонка, что произойдёт раньше - получатель получит DS или проведёт *submittx*. Если получатель первым получит DS, то он сможет отследить это мошенничество. Учитывая, что проверка происходит всегда, вне зависимости, был ли билет выигрышным, мошенническое поведение будет обнаружено до первого реального платежа. В результате, возникает вероятность, что плательщик может обмануть получателя на актуальную сумму платежа, ***amount / risk***.
   Причём чем раньше мошенник запустил DS, тем выше как вероятность успешности атаки (DS попадёт в блокчейн), так и детектирования мошенничества получателем.

Но, учитывая то, что Randpay-транзакция обычно не срабатывает, мошенническое поведение клиента будет с высокой вероятностью обнаружено на "не-выигрышном билете". Тем не менее, при неизвестной топологии сети, получатель в среднем подвергается риску потерять половину актуальной суммы платежа, то есть сумму

amount / risk / 2.

Стратегия получателя платежа для борьбы с этим мошенничеством - перед отправкой товара или предоставлением услуги подождать случайный интервал времени, примерно 0-10 сек в наблюдении - появится ли в сети DS-транзакция, которая проявит мошенническую сущность недобросовестного плательщика. Если появилась - то не отправлять товар, не глядя на то, какая из транзакций выиграет в логической гонке.

2. Плательщик формирует DS, но отправляет её в сеть только тогда, когда к нему в mempool приходит сработавшая Randpay-транзакция, которой он заплатил. То есть, попытку атаковать он делает только тогда, когда видит, что получатель Randpay выиграл у него деньги, и пытается их выдернуть обратно. В этой ситуации у плательщика не так много шансов получить их обратно, так как он получает свою копию выигравшей транзакции, которая уже разошлась по сети, и DS сетью скорее всего будет отвергнута.
   И так как в Emercoin стоит жёсткий приоритет транзакции по времени приёма, и нет приоритетов, зависящих от fee, то мошенник не может повышенным fee поднять приоритет своей DS-транзакции, выдавив из mempool оригинальную транзакцию.
   Для минимизации таких шансов, сервер получателя должен иметь соединения при помощи конфигурационного параметра ***connect*** только с несколькими заранее известными надёжными peer-ами, чтобы исключить прямую доставку выигравшей randpay-транзакции мошеннику, и дать ей время беспрепятственно разойтись по сети. Ведь чем позже мошенник получит копию randpay через блокчейн, тем меньше шансов у его DS выиграть гонку. Также полезна задержка передачи товара плательщику, с удержанием суммы, если получатель выиграл.
   Для новых неизвестных плательщиков имеет смысл ограничить ***risk***, чтоб они не взвинтили сумму выигрыша (заметим, что если атака успешна, он выигрывает полную сумму ***amount***, а не ***amount / risk***).

3. Мошенник - минтер POS.
   Находит kernel-транзакцию, решающую блок. Тут же к ней прикрепляет блок, содержащий DS. Но блок не отправляет в сеть, а делает покупку с использованием Randpay. Получив товар - публикует блок в сеть, тем самым отбирая деньги назад - ведь DS-транзакция иметь будет одно подтверждение, а оригинальная randpay-транзакция - ни одного! Но заметим, что пока плательщик удерживает блок от публикации, есть шанс, что кто-то другой закроет блок, и тогда плательщик потеряет минтерскую награду за это решение. Здесь возникают два варианта, подобные вышеописанным:
   - Мошенник всегда публикует задержанный блок, и тем самым информирует получателя о том, что он мошенник, даже если randpay не сработал.
   - Мошенник публикует задержанный блок только после появления транзакции в сети, чем вычёркивает её из mempool-а.

В обоих случаях, минтерская награда за блок подвергается риску не-получения вследствие задержки, ведь кто-то другой может закрыть блок и получить награду. Усреднённый риск - 1% от суммы на каждые 6 сек задержки. Причём это для каждого блока, а не только для того, где получатель выиграл. Так что сумму потерь надо умножать на ***risk***:
loss = risk * block_rewrd * time_delay / 600s.
Заметим также, что для осуществления такой атаки надо иметь много монет, долго вылежанных, и атака возможна далеко не всегда. В общем, для систематического воровства – способ не годится.

Рекомендации продавцу товара (получателю платежа):

Основная рекомендация - как и в обычных транзакциях, ждать нескольких подтверждений (хотя бы одного). Это закроет вас от происков мошенников без лишних проблем.

Если всё же требуется продажа в realtime, без подтверждений:

- Ведите список клиентов (хотя бы по IP), и не давайте непроверенным клиентам использовать большое значение ***risk***.
- Для непроверенных клиентов - делайте задержку отправки товара или услуги.
- Отслеживайте DS через другой кошелёк-приёмник, не связанный с основным.
- Запретите кому попало присоединиться к основному кошельку - держите ***connect*** только на пиры, которым Вы склонны доверять (например, из пула **seed.emercoin.com**)
- Ну и если товар или услуга не особо ценны для вас, и для вас приемлемо их отдать без компенсации - смиритесь с риском потери от мелкого мошенничества, как супермаркеты смиряются с мелким воровством.

Атака со стороны продавца

Возникает вопрос: Может ли продавец обмануть плательщика, предоставив пространство адресов **addrchap**, в котором находится более одного “выигрышного” адреса, то есть адреса, к которому у продавца имеется приватный ключ? Иными словами - может ли продавец в задаваемом плательщику данном адресном пространстве иметь два или более выигрышных билета? Ибо в этом случае - шансы произвести платёж удваиваются, что также удваивает матожидание средней уплаченной суммы.

Ответ на этот вопрос таков: Да, теоретически, это возможно, но вероятность такого события ничтожно мала, и её можно не принимать во внимание. Рассмотрим эту ситуацию подробнее.

Как известно, адрес платежа в Emercoin - это 160-битовое число, полученное как результат криптографически стойкого хеширования публичного ключа, и оно имеет равномерное распределение во всём интервале возможных значений. Примем среднее значение **risk** в системе - 2^20 бит, или примерно 1,000,000 (реальные значения буду намного меньше). Тогда атакующему для подбора остаётся 2^140 (140=160-20) вариантов addrchap, для перебора и попыток сгенерировать пару адресов, попадающую на один и тот же **addrchap**. Используя стратегию, основанную на парадоксе дней рождения, атакующий может решить задачу поиска пары за примерно 2^70 операций подбора, имея 2^70 ячеек памяти для хранения ключей. Учитывая скорость генерации очередного ключа в 10ms на современном компьютере получается, что время подбора пары превышает 374 миллиарда лет. Для сравнения - возраст Вселенной примерно 12 миллиардов лет. Даже если предположить, что у атакующего есть кластер из

миллиона компьютеров, производительность каждого из которых в 1000 раз превосходит современный PC, то время подбора становится сущим пустяком - всего-то 374 года.

В общем, атака теоретически возможна, но её организация требует огромного времени и ресурсов - практичнее просто майнить Emercoin.

- Автор документа и системы Randpay: **Олег Ховайко**