Аннотация

Этот документ описывает иерархические детерминированные кошельки (или "HD кошельки"): кошельки, которые могут быть частично или полностью переданы различным системам, каждая из которых может иметь или не иметь возможность тратить монеты.

Спецификация предназначена для установления стандарта детерминированных кошельков, которые могут быть взаимозаменяемыми между различными клиентами. Хотя описанные здесь кошельки имеют множество функций, не все из них обязательны для поддерживающих клиентов.

Спецификация состоит из двух частей. В первой части представлена система для получения дерева ключевых пар из одного семени. Вторая часть демонстрирует, как построить структуру кошелька на основе такого дерева.

Спецификация:

Получение ключей Соглашения

В дальнейшем тексте мы будем предполагать использование криптографии с открытым ключом, применяемой в Bitcoin, а именно эллиптическую кривую, использующую параметры поля и кривой, определенные secp256k1 (http://www.secg.org/sec2-v2.pdf). Переменные ниже могут быть:

Целыми числами по модулю порядка кривой (обозначается как n). Координатами точек на кривой. Последовательностями байтов. Сложение (+) двух пар координат определяется как применение операции группы EC. Конкатенация (||) – это операция добавления одной последовательности байтов к другой. В качестве стандартных функций преобразования мы предполагаем:

point(p): возвращает пару координат, полученную в результате умножения точки на эллиптической кривой (многократное применение операции группы EC) базовой точки secp256k1 на целое число p. ser32(i): сериализует 32-битное беззнаковое целое число i как последовательность из 4 байтов, начиная с самого значащего байта. ser256(p): сериализует целое число p как последовательность из 32 байтов, начиная с самого значащего байта. serP(P): сериализует пару координат P = (x, y) как последовательность байтов, используя сжатую форму SEC1: (0x02 или 0x03) || ser256(x), где заголовочный байт зависит от четности опущенной координаты y. parse256(p): интерпретирует 32-байтовую последовательность как 256-битное число, начиная с самого значащего байта. Расширенные ключи Далее мы определим функцию, которая выводит несколько дочерних ключей из родительского ключа. Чтобы предотвратить зависимость этих ключей только от самого ключа, мы сначала расширяем как приватные, так и публичные ключи дополнительными 256 битами энтропии. Это расширение,

называемое цепным кодом, идентично для соответствующих приватных и публичных ключей и состоит из 32 байтов.

Мы представляем расширенный приватный ключ как (k, c), где k – это обычный приватный ключ, а c – цепной код. Расширенный публичный ключ представлен как (K, c), где K = point(k) и c – цепной код.

Каждый расширенный ключ имеет 2^31 обычных дочерних ключей и 2^31 усиленных дочерних ключей. Каждый из этих дочерних ключей имеет индекс. Обычные дочерние ключи используют индексы от 0 до 2^31-1. Усиленные дочерние ключи используют индексы от 2^31 до 2^32-1. Для упрощения обозначения индексов усиленных ключей, число iH представляет собой i+2^31.

Функции получения дочерних ключей (CKD)

Имея родительский расширенный ключ и индекс i, можно вычислить соответствующий дочерний расширенный ключ. Алгоритм для этого зависит от того, является ли дочерний ключ усиленным (или, эквивалентно, если i ≥ 2^31), и говорим ли мы о приватных или публичных ключах.

Приватный родительский ключ → приватный дочерний ключ

Функция CKDpriv((kpar, cpar), i) → (ki, ci) вычисляет дочерний расширенный приватный ключ из родительского расширенного приватного ключа:

Проверить, является ли i ≥ 2^31 (усиленный дочерний ключ). Если так (усиленный дочерний ключ): пусть I = HMAC-SHA512(Key = cpar, Data = 0x00 || ser256(kpar) || ser32(i)). (Примечание: 0x00 дополняет приватный ключ до 33 байтов.) Если нет (обычный дочерний ключ): пусть I = HMAC-SHA512(Key = cpar, Data = serP(point(kpar)) || ser32(i)). Разделить I на две 32-байтовые последовательности, IL и IR. Возвращаемый дочерний ключ ki = parse256(IL) + kpar (mod n). Возвращаемый цепной код ci = IR. В случае parse256(IL) ≥ n или ki = 0, полученный ключ недействителен, и следует продолжить с следующим значением i. (Примечание: вероятность этого менее 1 из 2^127.) Функция HMAC-SHA512 описана в RFC 4231.

Публичный родительский ключ → публичный дочерний ключ Функция CKDpub((Kpar, cpar), i) → (Ki, ci) вычисляет дочерний расширенный публичный ключ из родительского расширенного публичного ключа. Она определена только для неусиленных дочерних ключей.

Проверить, является ли i ≥ 2^31 (усиленный дочерний ключ). Если так (усиленный дочерний ключ): вернуть ошибку. Если нет (обычный дочерний ключ): пусть I = HMAC-SHA512(Key = cpar, Data = serP(Kpar) || ser32(i)). Разделить I на две 32-байтовые последовательности, IL и IR. Возвращаемый дочерний ключ Ki равен

point(parse256(IL)) + Kpar. Возвращаемый цепной код ci равен IR. В случае parse256(IL) ≥ n или Ki является точкой на бесконечности, полученный ключ недействителен, и следует продолжить с следующим значением i. Приватный родительский ключ → публичный дочерний ключ Функция N((k, c)) → (K, c) вычисляет расширенный публичный ключ, соответствующий расширенному приватному ключу (так называемая "обезвреженная" версия, так как она удаляет возможность подписывать транзакции).

Возвращаемый ключ K равен point(k).

Возвращаемый цепной код c – это переданный цепной код. Для вычисления публичного дочернего ключа от родительского приватного ключа: N(CKDpriv((kpar, cpar), i)) (работает всегда). CKDpub(N(kpar, cpar), i) (работает только для неусиленных дочерних ключей). Тот факт, что они эквивалентны, делает неусиленные ключи полезными (можно выводить дочерние публичные ключи от заданного родительского ключа без знания какого-либо приватного ключа), а также отличает их от усиленных ключей. Причина, по которой не всегда используются неусиленные ключи (которые более полезны), связана с безопасностью; см. ниже для получения дополнительной информации. Публичный родительский ключ → приватный дочерний ключ Это невозможно.

Дерево ключей

Следующий шаг – это каскадирование нескольких конструкций CKD для построения дерева. Мы начинаем с одного корня, мастер-ключа m. Вычисляя CKDpriv(m,i) для нескольких значений i, мы получаем ряд узлов первого уровня. Поскольку каждый из них является расширенным ключом, CKDpriv можно также применить к ним.

Для сокращения обозначений мы будем записывать CKDpriv(CKDpriv(CKDpriv(m,3H),2),5) как m/3H/2/5. Эквивалентно для публичных ключей мы будем записывать CKDpub(CKDpub(CKDpub(M,3),2),5) как M/3/2/5. Это приводит к следующим тождествам:

N(m/a/b/c) = N(m/a/b)/c = N(m/a)/b/c = N(m)/a/b/c = M/a/b/c. N(m/aH/b/c) = N(m/aH/b)/c = N(m/aH)/b/c. Однако, N(m/aH) не может быть переписан как N(m)/aH, так как последнее невозможно. Каждый конечный узел в дереве соответствует фактическому ключу, в то время как внутренние узлы соответствуют коллекциям ключей, которые происходят от них. Цепные коды конечных узлов игнорируются, и важен только их встроенный приватный или публичный ключ. Благодаря этой конструкции, знание расширенного приватного ключа позволяет восстановить все дочерние приватные и публичные ключи, а знание расширенного публичного ключа позволяет восстановить все дочерние неусиленные публичные ключи.

Идентификаторы ключей Расширенные ключи могут быть идентифицированы с помощью Hash160 (RIPEMD160 после SHA256) сериализованного ECDSA публичного

ключа K, игнорируя цепной код. Это соответствует данным, используемым в традиционных адресах Bitcoin. Однако не рекомендуется представлять эти данные в формате base58, так как это может быть интерпретировано как адрес (и программное обеспечение кошелька не обязано принимать оплату на сам цепной ключ).

Первые 32 бита идентификатора называются отпечатком ключа.

Формат сериализации Расширенные публичные и приватные ключи сериализуются следующим образом:

4 байта: байты версии (mainnet: 0x0488B21E публичный, 0x0488ADE4 приватный; testnet: 0x043587CF публичный, 0x04358394 приватный) 1 байт: глубина: 0x00 для мастер-узлов, 0x01 для ключей первого уровня и т.д. 4 байта: отпечаток ключа родителя (0x00000000, если мастер-ключ) 4 байта: номер дочернего ключа. Это ser32(i) для i в xi = xpar/i, где xi – это сериализуемый ключ. (0x00000000, если мастер-ключ) 32 байта: цепной код 33 байта: данные публичного или приватного ключа (serP(K) для публичных ключей, 0x00 || ser256(k) для приватных ключей) Эта 78-байтовая структура может быть закодирована как другие данные Bitcoin в Base58, сначала добавив 32 контрольных бита (полученные из двойной контрольной суммы SHA-256), а затем преобразовав в представление Base58. Это приводит к строке, закодированной в Base58, длиной до 112 символов. Из-за выбора байтов версии, представление Base58 будет начинаться с "xprv" или "xpub" в mainnet, "tprv" или "tpub" в testnet. Обратите внимание, что отпечаток родительского ключа служит только для быстрого обнаружения родительских и дочерних узлов в программном обеспечении, и программное обеспечение должно быть готово к работе с коллизиями. Внутри можно использовать полный 160-битный идентификатор.

При импорте сериализованного расширенного публичного ключа, реализации должны проверять, соответствует ли координата X в данных публичного ключа точке на кривой. Если нет, расширенный публичный ключ недействителен.

Генерация мастер-ключа Общее количество возможных расширенных пар ключей почти 2^512, но создаваемые ключи имеют длину всего 256 бит и обеспечивают примерно половину этой безопасности. Поэтому мастер-ключи не генерируются напрямую, а вместо этого из потенциально короткого значения семени.

Сгенерируйте последовательность байтов семени S выбранной длины (между 128 и 512 бит; рекомендуется 256 бит) из (P)RNG. Вычислите I = HMAC-SHA512(Key = "Bitcoin seed", Data = S) Разделите I на две 32-байтовые последовательности, IL и IR. Используйте parse256(IL) как мастер-секретный ключ, и IR как мастер-цепной код. В случае, если parse256(IL) равно 0 или parse256(IL) ≥ n, мастер-ключ недействителен.

Спецификация: Структура кошелька В предыдущих разделах были описаны деревья ключей и их узлы. Следующим шагом является наложение структуры кошелька на это

дерево. Описанная в этом разделе структура является только стандартной, хотя клиентам рекомендуется следовать ей для совместимости, даже если не все функции поддерживаются.

Стандартная структура кошелька HDW организован в виде нескольких «аккаунтов». Аккаунты нумеруются, и по умолчанию аккаунт («») имеет номер 0. Клиенты не обязаны поддерживать более одного аккаунта - если нет, они используют только стандартный аккаунт.

Каждый аккаунт состоит из двух цепочек пар ключей: внутренней и внешней. Внешняя цепочка ключей используется для генерации новых публичных адресов, а внутренняя цепочка ключей используется для всех других операций (адреса для сдачи, генерация адресов и т.д.). Клиенты, не поддерживающие отдельные цепочки ключей, должны использовать внешнюю для всех операций.

m/iH/0/k соответствует k-й паре ключей внешней цепочки аккаунта номер i, производного от мастер-ключа m. m/iH/1/k соответствует k-й паре ключей внутренней цепочки аккаунта номер i, производного от мастер-ключа m. Примеры использования Полный доступ к кошельку: m В случаях, когда двум системам необходимо доступ к одному общему кошельку, и обеим нужно иметь возможность совершать расходы, необходимо поделиться мастер-ключом. Узлы могут хранить пул из N ключей для внешних цепочек, чтобы отслеживать входящие платежи. Look-ahead для внутренних цепочек может быть очень маленьким, так как здесь не ожидаются разрывы. Дополнительный look-ahead может быть активен для цепочек первого неиспользованного аккаунта, что приводит к созданию нового аккаунта при использовании. Обратите внимание, что название аккаунта все еще нужно вводить вручную и оно не может быть синхронизировано через блокчейн.

Аудит: N(m/*) В случае, когда аудитор нуждается в полном доступе к списку входящих и исходящих платежей, можно поделиться всеми публичными расширенными ключами аккаунта. Это позволит аудитору видеть все транзакции кошелька по всем аккаунтам, но ни один из секретных ключей не будет доступен.

Балансы по офисам: m/iH Когда у компании есть несколько независимых офисов, они могут использовать кошельки, производные от одного мастер-ключа. Это позволит головному офису поддерживать «супер-кошелек», который видит все входящие и исходящие транзакции всех офисов и даже позволяет перемещать деньги между офисами.

Регулярные бизнес-транзакции: N(m/iH/0) В случае, когда два бизнес-партнера часто переводят деньги друг другу, можно использовать расширенный публичный ключ внешней цепочки определенного аккаунта (M/i h/0) как своего рода «супер-адрес», позволяющий частые транзакции, которые не могут быть легко связаны, без необходимости запрашивать новый адрес для каждого платежа. Такой механизм также

может использоваться операторами майнинговых пулов как адрес для переменных выплат.

Небезопасный получатель денег: N(m/iH/0) Когда для работы сайта электронной коммерции используется небезопасный веб-сервер, ему необходимо знать публичные адреса, которые используются для получения платежей. Веб-серверу нужно знать только публичный расширенный ключ внешней цепочки одного аккаунта. Это означает, что кто-то, незаконно получивший доступ к веб-серверу, сможет в лучшем случае видеть все входящие платежи, но не сможет украсть деньги, не сможет (легко) отличить исходящие транзакции и не сможет видеть платежи, полученные другими веб-серверами, если их несколько.

Совместимость Для соответствия этому стандарту клиент должен как минимум уметь импортировать расширенный публичный или приватный ключ, чтобы предоставить доступ к его прямым потомкам в качестве ключей кошелька. Структура кошелька (мастер/аккаунт/цепочка/подцепочка), представленная во второй части спецификации, является рекомендательной, но предлагается в качестве минимальной структуры для легкой совместимости - даже если не создаются отдельные аккаунты или не делается различие между внутренними и внешними цепочками. Однако реализации могут отклоняться от нее для конкретных нужд; более сложные приложения могут требовать более сложной структуры дерева.

Безопасность В дополнение к ожиданиям от самой криптографии с открытым ключом EC:

Имея публичный ключ K, злоумышленник не может найти соответствующий приватный ключ более эффективно, чем решая задачу дискретного логарифма EC (предполагается, что требуется 2^128 групповых операций). Ожидаемые свойства безопасности этого стандарта таковы: Имея дочерний расширенный приватный ключ (ki, ci) и целое число i, злоумышленник не может найти родительский приватный ключ kpar более эффективно, чем 2^256 переборов HMAC-SHA512. Имея любое количество ($2 \leq N \leq 2^{32}-1$) кортежей (индекс, расширенный приватный ключ) (ij, (kij, cij)) с различными ij, определение того, производны ли они от общего родительского расширенного приватного ключа (то есть существует ли (kpar, cpar), такой что для каждого j в (0..N-1) CKDpriv((kpar, cpar), ij) = (kij, cij)), не может быть выполнено более эффективно, чем 2^256 переборов HMAC-SHA512. Однако, обратите внимание, что следующие свойства не существуют: Имея родительский расширенный публичный ключ (Kpar, cpar) и дочерний публичный ключ (Ki), сложно найти i. Имея родительский расширенный публичный ключ (Kpar, cpar) и нехардненый дочерний приватный ключ (ki), сложно найти kpar. Последствия Приватные и публичные ключи должны храниться в безопасности, как обычно. Утечка приватного ключа означает доступ к монетам, утечка публичного ключа может привести к утрате конфиденциальности.

Необходимо проявлять больше осторожности в отношении расширенных ключей, так как они соответствуют целому (под)дереву ключей.

Одной из слабостей, которая может быть неочевидной, является то, что знание родительского расширенного публичного ключа плюс любого нехардненого приватного ключа, происходящего от него, эквивалентно знанию родительского расширенного приватного ключа (а следовательно, и всех производных от него приватных и публичных ключей). Это означает, что расширенные публичные ключи должны храниться более тщательно, чем обычные публичные ключи. Это также причина существования хардненых ключей и того, почему они используются на уровне аккаунта в дереве. Таким образом, утечка приватных ключей, относящихся к конкретному аккаунту (или ниже), никогда не ставит под угрозу мастер-ключ или другие аккаунты.