ИССЛЕДОВАТЕЛЬСКИЙ БЮЛЛЕТЕНЬ MRL-0009

Thring-подписи и их применение в цифровых криптовалютах, скрывающих лицо, производящее траты

Брэндон Гуделл (Brandon Goodell)[1], Саранг Ноезер (Sarang Noether)[2]
Исследовательская лаборатория Monero (Monero Research Lab)
1 ноября 2018

Аннотация

В этой работе нами предлагаются пороговые кольцевые подписи (thring-подписи) для совместного вычисления кольцевых подписей, а также рассматривается возможность и осуществимость подделки thring-подписей и их применение в цифровых валютах, в частности, атомные свопы между блокчейнами с сокрытием лица, осуществляющего трату, с обеспечением конфиденциальности сумм без доверенных настроек. Также в работе нами представлен вариант реализации thring-подписей, называемый нами связываемыми спонтанными пороговыми анонимными групповыми подписями, а также приводится доказательство того, что такие подписи экзистенциально невозможно подделать.

1 Введение

Криптовалюты и схемы реализации электронных денег в значительной степени опираются на цифровые подписи, исключающие возможность подделки, и существует множество вариантов использования пороговых подписей при таких настройках, начиная с многофакторной аутентификации при утверждении транзакций и заканчивая атомными свопами между блокчейнами. Идея проста: группа взаимодействующих пользователей принимает решение относительно порогового значения, и некоторые из членов группы могут совместно вычислить подписи, но только при условии, что согласится такое количество членов группы, которое будет превышать установленный порог. Изначально для создания подписей Bitcoin использовал алгоритм ECDSA на базе secp256k1, но этот способ не является единственным вариантом для разработчиков криптовалют. Позже популярными стали подписи Шнорра [22], функциональность которых была выше, включая использование пороговых мультиподписей, представленных в работе [15]. Также исследовалась возможность использования подписей Окамото [19] в мультиподписях в криптовалютах [10]. Также существуют другие варианты, такие как кольцевые подписи [13] или аккумуляторы типа Zerocoin [16] , предполагающие аутентификацию сообщений с сокрытием отправляющей стороны, когда верификаторы могут проверить, было ли сообщение аутентифицировано членом анонимной группы. При этом сохраняется незначительное преимущество в определении того, какой именно из членов сделал это.

В случае с криптовалютами аутентификация сообщений с сокрытием лица, *являющегося отправителем*, необходима для обеспечения неопределённости лица, являющегося отправителем. Мы интерпретируем такую неопределённость как свойство, касающееся правдоподобного отрицания в истории транзакций: финансовые системы, использующие эти опции, позволяют поддерживать публично верифицируемые реестры, и при этом пользователи могут по крайней мере правдоподобно отрицать своё

[1] surae.noether@protonmail.com
[2] sarang.noether@protonmail.com

участие в какой-либо транзакции (или в их небольшой группе). Поэтому вполне естественно было бы исследовать пороговое расширение таким схемам аутентификации сообщений с сокрытием отправителя, как кольцевые подписи. Эти кольцевые подписи также могут быть использованы для построения кольцевых конфиденциальных транзакций в контексте цифровой валюты. Функциональность мультиподписей применительно к цифровой валюте также позволяет, например, производить атомные свопы между блокчейнами. Следовательно, схема пороговой кольцевой подписи позволяет проводить атомные свопы между блокчейнами с сокрытием лица, совершающего трату, для обеспечения конфиденциальности сумм без доверенных настроек.

Мы называем пороговые кольцевые подписи *thring-подписями*. Предыдущие предложенные варианты реализации thring-подписей (как, например, в работах [9], [12], [23]) предполагали утечку информации (вскрывались свойства подписывающей группы, такие как количество подписантов, или даже публиковался список публичных ключей, использованных для подписи, с самими подписями) или же вычисление ключей, подобно тому как это показано в работе [3], что делало такие схемы уязвимыми для атак, направленных на кражу ключей. Поэтому мы считаем, что такие предложенные варианты не подходят для использования в случае цифровыми валютами, ориентированными на обеспечение анонимности.

Нами был разработан свой вариант реализации thring-подписи, использующий подход, подобный Musig, применительно к связываемым спонтанным групповым (LSAG) подписям, о которых говорится в работе [13], с учётом изменений, представленных в работах [1] и [18] и предназначенных для применения в цифровых валютах. Подход Musig, описанный в работе [15], предполагает накопление ключей в совокупности с применением подхода, который впервые был предложен в работе [20] и который является устойчивым к атакам, направленным на кражу ключей. Как обычно, в случае со схемами мультиподписей, мы заменяем произвольные данные, используемые для подписания, суммами данных, выбранных участниками, которые раскрываются на этапе совершения и раскрытия. И наконец, как и в случае с Musig, мы включаем ключи подписания в задачу вычисления подписи, что не даёт запросам оракула в доказательстве экзистенциальной невозможности подделки происходить в неверном порядке.

1.1 Наш вклад

Нами предлагается схема LSTAG-подписи, напоминающая мультиподписи Musig, описанные в работе [15], и являющаяся версией LSAG-подписей с заданным порогом. Нами даётся определение экзистенциальной невозможности подделки в параметрах кольцевых подписей и приводится доказательство того, что схема безопасна при таком определении. Нами также отмечается разница между нашей схемой и фиктивными вариантами применения с криптовалютами.

Предлагаемая нами схема имеет структуру n из n, но мы можем расширить её до структуры пороговых схем m из n, воспользовавшись стандартными методами, и даже до многоуровневой схемы для использования в кольцевых конфиденциальных транзакциях. Схема предполагает накопление ключей в том смысле, что верификация подписи не требует знания порогового значения подписания. Размер подписей не зависит от количества участвующих подписантов. Доказательство безопасности соответствует простой модели публичных ключей. Нами используется тот же самый процесс Musig, включающий в себя три этапа, а верификация подписей происходит идентично тому, как это происходит в случае с подписями LSAG.

В Разделе 2 мы разъясняем наши понятия, допуски и прочие необходимые условия. В Разделе 3 мы в общих чертах рассказываем о том, как подписи LSAG в настоящее время работают в Monero, как

применить к ним эвристический подход к определению порогового значения Musig, чтобы получить LSTAG, а также как использовать подписи LSTAG в кольцевых конфиденциальных транзакциях. В Разделе 4 нами даётся определение подписей LSTAG и приводится пример их реализации. В Разделе A мы рассказываем о защите LSTAG-подписей от подделки, приводим доказательства безопасности варианта реализации из Раздела 4 при таком определении. Нами также рассматривается связываемость, оправданность, неопределённость подписанта, а также неразличимость накапливаемых ключей.

1.2 Связанные работы и задачи

Впервые LSAG-подписи были описаны в работе [13]; изменения были предложены в работе [1]. Конфиденциальные транзакции применительно к цифровым валютам были впервые описаны в работе [14], а в работе [18] были в первый раз описаны расширения кольцевых подписей конфиденциальных транзакций, использующие расширения по вектору ключей LSAG-подписей, называемые многоуровневыми связываемыми спонтанными анонимными групповыми (MLSAG) подписями. Для вычисления мультиподписей мы обычно используем подход, описанный в работе [3], применяя при этом схему накопления ключей в стиле Musig, описанную в работе [20], а также подход, предполагающий обязательство с последующим раскрытием (commit-reveal), из работы [15]. Наше доказательство невозможности подделки, подобное описанному в работе [15], использует двойное применение метода реализации форка «обратной перемоткой» по успешному транскрипту (rewind-on-success forking) (впервые чётко был представлен в работе [13]).

Доступны и другие технологии использования мультиподписей, особенно в случае с попарным соединением и обучением с ошибками. Возьмём, например, работу [8], где описан новый и довольно общий гомоморфный метод настройки порогового значения, при котором используется только один сеанс связи в среде обучения с ошибками, позволяющий создавать полностью гомоморфные пороговые подписи и шифрование. Совсем недавно в работе [7] было описано применение настроек на базе попарного соединения, позволяющих создавать компактные мультиподписи с крайне полезными свойствами.

Следует отметить, что эвристический подход к определению порогового значения, описанный в работе [3], предлагает реализацию общей схемы мультиподписей в рамках простой модели публичных ключей. Также в этой работе был подробно рассмотрен допуск знания секретного ключа (KOSK). Стоит отметить проблемы, возникшие с практичностью и безопасностью реализации схем, безопасность которых была доказана при настройках KOSK с использованием доказательств владения секретными ключами при отсутствии специальных мер предосторожности (см., например, работу [21]). Поэтому мы избегаем допуска KOSK, несмотря на то, что доказательства безопасности для таких схем, как наша, гораздо проще настроек KOSK с доказательствами владения, заменяющими сертифицирующий орган. В работе [20] представлен не использующий KOSK метод агрегации совместно используемого публичного ключа мультиподписи, который мы используем в данной работе. Такой метод устойчив к атакам, направленным на кражу ключей (хотя он и использует настройки, основанные на попарном объединении). Ранняя версия Musig подразумевала один сеанс связи, который был позднее заменён на этап обязательства и раскрытия для подписания. Безусловно, в работе [10] было продемонстрировано, что возможность доказательства безопасности схемы подписи подобной Musig за один сеанс связи маловероятна при допуске сложности дискретного логарифма.

Чтобы данные thring-подписи можно было использовать с цифровыми валютами в кольцевых конфиденциальных транзакциях, они должны быть расширены. Например, применительно к цифровым валютам, использующим (некоторый вариант) протокола CryptoNote, таким как Monero, реализация должна учитывать векторы пар ключей, включая ключ просмотра в дополнение к ключу тра-

ты/подписания, а также должна учитывать вычисления одноразового ключа. Доказательство безопасности нашей схемы при условии такого расширения находится вне области применения данного документа. Более подробно кольцевые конфиденциальные транзакции и кольцевые пороговые подписи формализованы, например, в работе [11].

1.3 Особая благодарность

Нам хотелось бы выразить особую благодарность членам сообщества Monero, которые воспользовались Системой общественного финансирования на `GetMonero.org`, чтобы поддержать Исследовательскую лабораторию Monero. Читатель также может рассматривать это как заявление конфликта интересов, так как наше финансирование осуществляется в Monero напрямую членами сообщества через Систему общественного финансирования. Нам также, в частности, хотелось бы поблагодарить Эндрю Поэлстра (Andrew Poelstra) и Яника Сюрина (Yannick Seurin), а также многих других за крайне полезные комментарии.

2 Обозначения и допуски

В основном, мы используем каллиграфический шрифт для обозначения алгоритмов PPT и оракулов: $\mathcal{A}, \mathcal{A}', \mathcal{A}'', \mathcal{B}$ являются алгоритмами PPT, $\mathcal{H}$ является случайным оракулом, а $\mathcal{SO}$ является оракулом подписания. Мы часто даём оракулу алгоритмический доступ, что обозначается верхним индексом: $\mathcal{A}^{\mathcal{SO}}$ означает, что у $\mathcal{A}$ есть доступ оракула к $\mathcal{SO}$. Мы часто оставляем верхние индексы скрытыми для ясности, за исключением случаев, когда это может ввести в заблуждение. Мы используем `шрифт teletype` чтобы описать входные и выходные данные алгоритмов или же названия специальных алгоритмов криптографической схемы: $\texttt{inp}_{\mathcal{A}}$ обозначает входные данные $\mathcal{A}$, $\texttt{out}_{\mathcal{A}}$ обозначает выходные данные, `Sign` является алгоритмом подписания в нашем варианте реализации и так далее. Алгоритмы часто сопровождаются символом отрицательного результата (или набором из таких символов), что обозначается нами как $\perp$: $\perp_{\mathcal{A}}$ является символом отрицательного результата для $\mathcal{A}$, $\perp_{\texttt{Sign}}$ является символом отрицательного результата для `Sign` и так далее.

Мы используем строчные английские и греческие буквы для обозначения целых чисел: $n, r, q, \ell, i, j, k, \alpha, \eta$ - все принадлежат $\mathbb{N}$. Для любого $r \in \mathbb{N} = \{1, 2, \ldots\}$мы указываем ряд элементов r $\{1, 2, \ldots, r\}$ как $[r]$. Мы используем подчёркивание, чтобы обозначить векторы, упорядоченные списки, последовательности, а также множества, проиндексированные вполне упорядоченными индексирующими множествами. Например, в случае с неупорядоченным списком независимых случайных оракулов $\underline{H}$, проиндексированным некоторым произвольным конечным множеством Λ где $n = |\Lambda|$ элементов, мы можем безопасно переиндексировать и допустить, что $\Lambda = [n]$, и записать $\underline{\mathcal{H}} = \{\mathcal{H}_i\}_{i \in [n]}$.

Для любого $n > 1$ мы обозначаем $\mathbb{Z}/n\mathbb{Z}$ как $\mathbb{Z}_n$. Мы допускаем, что перед началом этап настройки производится с безопасным параметром $\eta > 1$, и в результате получается некоторое $(\mathfrak{p}, \mathbb{G}, G, \underline{\mathcal{H}}, \phi, \Phi)$, а именно циклическая группа $\mathbb{G}$ порядка $\mathfrak{p}$ с генератором G, где элементы $\mathbb{Z}_{\mathfrak{p}}$ и $\mathbb{G}$ допускают η-битные представления, последовательность $\underline{\mathcal{H}}$ криптографических хеш-функций, а также две функции агрегирования ключей ϕ, Φ. Мы называем элемент $\mathbb{G}$ *публичным ключом* и применяем это обозначение во всём документе, используя заглавные английские буквы. Например, A, B, C, T, P, X являются публичными ключами, а $\underline{P} = \{P_\ell\}_{\ell \in [r]}$ обозначает последовательность из r публичных ключей. Мы называем такую последовательность *кольцом*, если она будет использоваться в качестве входа в кольцевой подписи. Для вычисления кольцевых подписей с членами кольца $r \in \mathbb{N}, r > 1$ мы допускаем «обёртывание» индексов из $[r]$путём идентификации индекса кольца $r + 1$ индексом кольца 1. Мы используем такое

правило только для индексов членов кольца, но не для каких-либо других индексов.

Мы называем членов $\mathbb{Z}_\mathfrak{p}$ *приватными ключами*. Так как это классы эквивалентности целых чисел, мы также используем прописные английские буквы для обозначения этих приватных ключей, но не тех, которые использовались для целых чисел: a, b, t, p и x являются приватными ключами $\mathbb{Z}_\mathfrak{p}$. Мы называем некоторый $X \in \mathbb{G}$ публичным ключом, *связанным* с некоторым приватным ключом $x \in \mathbb{Z}_\mathfrak{p}$ если $X = xG$ для генератора G, указанного выше, и мы подбираем соответствующие буквы: публичный ключ, связанный с приватным ключом a, обозначается как $A = aG$, публичный ключ, связанный с приватным ключом b обозначается как $B = bG$, и так далее. В случае со списком приватных ключей, скажем, $\underline{x} = (x_1, \ldots, x_n)$, мы записываем список соответствующих публичных ключей как $\underline{X} = (X_1, \ldots, X_n)$, где каждый $X_i = x_i G$. Мы обозначаем это как $\underline{X} = \underline{x}G$ с риском неправильного использования обозначения.

Мы используем правило, согласно которому $\phi(x, \{X\}) = x$ и $\Phi(\{X\}) = X$. В данном случае Φ в качестве входа берёт непустое мультимножество публичных ключей $\underline{X} = (X_1, \ldots, X_n)$ и в качестве выхода выдаёт совместно используемый публичный ключ $X_{\text{sh}} \leftarrow \Phi(\underline{X})$, а ϕ в качестве входа берёт приватный ключ x_i с непустым мультимножеством публичных ключей $\underline{X}$ (чтобы по крайней мере единожды в $\underline{X}$ было $X_i = x_i G$) и в качестве выхода выдаёт коэффициент $\beta_i = \phi(x_i, \underline{X})$. Пользователи устанавливают свою приватную совместно используемую часть как $x_i^* := \beta_i x_i = \phi(x_i, \underline{X}) x_i$. Мы говорим, что X_{sh} is *связано* или *является дочерним элементом* ключей в $\underline{X}$ и используем

$$\Phi(\underline{X}) := \sum_{i \in [n]} \beta_i X_i = \sum_i x_i^* G = X_{\text{sh}}.$$

Мы указываем $\underline{\mathcal{H}}$, шесть входов произвольной длины, криптографические хеш-функции η-битного выхода с независимыми выходами

$$\begin{array}{lll} \mathcal{H}_{\text{ki}} : \{0,1\}^* \to \mathbb{G} & \mathcal{H}_{\text{com}} : \{0,1\}^* \to \{0,1\}^\eta & \mathcal{H}_{\text{agg}} : \{0,1\}^* \to \mathbb{Z}_\mathfrak{p} \\ \mathcal{H}_{\text{sig}} : \{0,1\}^* \to \mathbb{Z}_\mathfrak{p} & \mathcal{H}_{\text{msg}} : \{0,1\}^* \to \mathbb{Z}_\mathfrak{p} & \mathcal{H}_{\text{sess}} : \{0,1\}^* \to \mathbb{Z}_\mathfrak{p} \end{array}$$

в соответствии с моделью случайного оракула. Мы обозначаем связь битовых строк символом $||$. Так как элементы $\mathbb{Z}_\mathfrak{p}$ могут быть описаны η бит, мы также можем взять $\mathcal{H}_{\text{agg}}$, $\mathcal{H}_{\text{sig}}$, $\mathcal{H}_{\text{com}}$, $\mathcal{H}_{\text{msg}}$ и $\mathcal{H}_{\text{sess}}$ для всех пяти в одной и той же кообласти. Таким образом, в целях реализации эти хеш-функции могут быть реализованы одной хеш-функцией $\mathcal{H}_{\text{sc}} : \{0,1\}^* \to \mathbb{Z}_\mathfrak{p}$ с использованием доменного разделения:

$$\begin{aligned} \mathcal{H}_{\text{com}}(x) &:= \mathcal{H}_{\text{sc}}(000 \mid\mid x) \\ \mathcal{H}_{\text{agg}}(x) &:= \mathcal{H}_{\text{sc}}(001 \mid\mid x) \\ \mathcal{H}_{\text{sig}}(x) &:= \mathcal{H}_{\text{sc}}(010 \mid\mid x) \\ \mathcal{H}_{\text{msg}}(x) &:= \mathcal{H}_{\text{sc}}(011 \mid\mid x) \\ \mathcal{H}_{\text{sess}}(x) &:= \mathcal{H}_{\text{sc}}(111 \mid\mid x) \end{aligned}$$

Это позволяет использовать для реализации только две хеш-функции, $\mathcal{H}_{\text{ki}}$ и $\mathcal{H}_{\text{sc}}$. Также в первом примере мы используем только $\mathcal{H}_{\text{sess}}$; наш вариант реализации позволяет использовать только первые четыре варианта $\mathcal{H}_{sc}$ и требует только двух битов префиксов. Следует отметить, что несмотря на то, что у каждой из этих функции будут η–битные выходы, эффективность их энтропии фактически будет ниже из-за этого доменного разделения.

Безусловно, если $\mathcal{H}_{\text{ki}}$ можно разбить на некоторые $\mathcal{H}_{\text{ki}} = \mu_G \cdot \mathcal{H}_{\text{ki}}^*$ для некоторых $\mathcal{H}_{\text{ki}}^* : \{0,1\}^* \to \mathbb{Z}_\mathfrak{p}$ (где $\mu_G : \mathbb{Z}_\mathfrak{p} \to \mathbb{G}$ является каноническим необратимым гомоморфизмом, определяемым соответствием

$x \mapsto xG$), то любой, кто узнает об этом разбиении $\mathcal{H}_{\text{ki}}$, сможет вычислить дискретный логарифм выходов $\mathcal{H}_{\text{ki}}$, что является крайне нежелательным свойством. Мы допускаем, что такое разбиение вычислить непросто.

3 Обоснование схемы

В этом разделе мы обоснуем нашу схему пороговой кольцевой подписи. Мы начнём с общего описания, как вычисляются традиционные LSAG-подписи при помощи пространств ключей типа CryptoNote. Затем мы опишем эвристические принципы определения порогов для построения нашего варианта реализации, а после этого кратко прокомментируем применение этих подписей в Monero.

3.1 Традиционные LSAG-подписи

В этом разделе мы кратко рассмотрим, как подписи работают в цифровых валютах на базе протокола CryptoNote, таких как Monero. В подписях, описанных в работе [13], используются образы ключей, зависящие от кольца подписантов, что делает их неподходящими с точки зрения связываемости в нашем контексте. В работе [1] была представлена модификация, которую мы кратко описываем в этой работе. Наше описание использует четыре криптографические хеш-функции, смоделированные как случайные оракулы $\mathcal{H}_{\text{sess}}$, $\mathcal{H}_{\text{sig}}$, $\mathcal{H}_{\text{ki}}$ и $\mathcal{H}_{\text{msg}}$.

Рассмотрим сначала пространства ключей Monero, основанные на протоколе CryptoNote. У пользователей имеются *пары ключей* (включающие в себя *ключ просмотра* и *ключ траты*), а для подписания транзакций они используют *пары ключей подписания* (включающие в себя *ключ транзакции* и *ключ сессии* [или же, как вариант, *одноразовый ключ* или *скрытый ключ*]). Ключи сессии вычисляются на основе пар пользовательских пар ключей и ключей транзакции; кольцевые подписи вычисляются при помощи ключей сессии. Становится доступным характерное пространство ключей, $\mathfrak{K} = \mathbb{Z}_{\mathfrak{p}}^2 \times \mathbb{G}^2$, а оба вида пар ключей выводятся на основе $\mathfrak{K}$.

Пользовательская пара ключей обозначается как $((a,b),(A,B)) \in \mathfrak{K}$, где a является *приватным ключом просмотра*, b обозначает *приватный ключ траты*, A является *публичным ключом просмотра*, а B - *публичным ключом траты*. Честные пользователи выбирают свои приватные ключи единообразно и в случайном порядке. Пара ключей подписания обозначается как $((t,p),(T,P)) \in \mathfrak{K}$, где t является *приватным ключом транзакции*, p - *приватным ключом сессии*, T - *публичным ключом транзакции*, а P обозначает *публичный ключ сессии*. Честные пользователи получают публичную часть их ключа транзакции T от отправителя. В случае с Monero мы говорим, что пара публичных ключей подписания (T,P) с индексом i в некоторой транзакции *адресована* паре публичных пользовательских ключей (A,B), если $P = \mathcal{H}_{\text{sess}}(aT,i)G + B$. При известных T,a,B,i или t,A,B,i владение ключом сессии может быть легко проверено следующим образом: просто необходимо убедиться в том, что $P - \mathcal{H}_{\text{sess}}(aT,i) = B$. При известных T,a,b,i или известных t,A,b,i приватный ключ сессии вычисляется как $p = \mathcal{H}_{\text{sess}}(aT,i) + b$. Далее мы «забываем» индекс транзакции i для простоты представления.

Если у Элис есть пользовательский ключ $((a,b),(A,B)) \in \mathfrak{K}$, и она ранее получила сообщение, содержащее некоторые публичные ключи подписания (T,P) адресованные (A,B), и Элис хочет адресовать некоторые ключи Бобу (у которого есть публичный пользовательский ключ (A',B')), то Элис вычисляет p, берёт новый приватный ключ транзакции $t' \leftarrow \mathbb{Z}_{\mathfrak{p}}$ и вычисляет новый публичный ключ сессии $P' := \mathcal{H}_{\text{sess}}(t'A')G + B'$ для Боба. Затем Элис отправляет (T',P') Бобу в сообщении $\mathfrak{m}$, а затем создаёт кольцевую подпись σ в изменённом сообщении $\mathfrak{m}^*$ содержащем $\mathfrak{m}$ и некоторое кольцо $\underline{P}$ таким образом, чтобы $P \in \underline{P}$. Элис вычисляет кольцевую подпись следующим образом.

Элис выбирает сообщение $\mathfrak{m} \in \{0,1\}^*$, вычисляет свой образ одноразового ключа $J = p\mathcal{H}_{\text{ki}}(P)$ и выбирает кольцо публичных ключей подписания $\underline{P} = \{P_1, \dots P_r\}$ таким образом, чтобы для секретного выделенного индекса π, выполнялось условие $P_\pi = P$. Элис собирает модифицированное сообщение $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, T', P')$ и вычисляет $M = \mathcal{H}(\mathfrak{m}^*)$. Для каждого $\ell = 1, \dots, r$ подписант вычисляет точку на эллиптической кривой на основе ℓ^{th} члена кольца $H_\ell := \mathcal{H}_{\text{ki}}(P_\ell)$. Подписант выбирает случайную секретную скалярную величину $u \xleftarrow{\$} \mathbb{Z}_{\mathfrak{p}}$ и вычисляет начальную временную пару точек $L_\pi := uG$, $R_\pi := u\mathcal{H}_\pi$. Подписант вычисляет начальное обязательство $c_{\pi+1} := \mathcal{H}_{\text{sig}}(M, L_\pi, R_\pi)$. Позднее мы будем использовать вариант этого обязательства с *префиксом из ключа* $c_{\pi+1} := \mathcal{H}_{\text{sig}}(M, P_\pi, L_\pi, R_\pi)$

Подписант продолжает работу с индексами $\ell = \pi+1, \pi+2, \dots, r-1, r, 1, 2, \dots, \pi-1$, выбирая случайную скалярную величину s_ℓ, вычисляя следующую пару точек $L_\ell := s_\ell G + c_\ell P_\ell$ и $R_\ell := s_\ell H_\ell + c_\ell J$, и вычисляет следующее обязательство $c_{\ell+1} := \mathcal{H}_{\text{sig}}(M, L_\ell, R_\ell)$ (или вариант с префиксом из ключа $c_{\ell+1} := \mathcal{H}_{\text{sig}}(M, P_\ell, L_\ell, R_\ell)$). Как только все обязательства c_ℓ будут вычислены, подписант приступит к вычислению $s_\pi := u - c_\pi p$ для следующего выделенного индекса π. $\sigma = (c_1, \underline{s})$ является подписью в сообщении $\mathfrak{m}$. Элис отправляет $(\mathfrak{m}^*, \sigma)$ Бобу. Мы называем ряд уравнений

$$\begin{aligned}
c_2 =& \mathcal{H}_{\text{sig}}(M, s_1 G + c_1 P_1, s_1 H_1 + c_1 J) = \mathcal{H}_{\text{sig}}(M, L_1, R_1) \\
c_3 =& \mathcal{H}_{\text{sig}}(M, L_2, R_2) \\
& \vdots \\
c_r =& \mathcal{H}_{\text{sig}}(M, L_{r-1}, R_{r-1}) \\
c_1 =& \mathcal{H}_{\text{sig}}(M, L_r, R_r)
\end{aligned}$$

(или их вариантов с префиксами из ключей) *верификационными уравнениями*.

После получения $(\mathfrak{m}^*, \sigma)$ Боб проводит анализ $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, (T', P'))$ и проверяет, чтобы $M = \mathcal{H}(\mathfrak{m}^*)$. Боб может легко проверить, действительно ли $P' - \mathcal{H}_{\text{sess}}(a'T')G = B'$, чтобы увидеть, является ли он адресатом этих ключей. Боб также может выделить приватный ключ подписания, вычислив $p' = \mathcal{H}_{\text{sess}}(a'T') + b'$. Таким образом, Боб может провести ту же процедуру, что и Элис выше, чтобы передать новые ключи подписания другим пользователям. Тем не мене Боб не может быть уверен в том, что подпись является подлинной, поэтому он верифицирует подпись следующим образом.

При известном $(\mathfrak{m}^*, \sigma)$ верификатор проводит анализ $\sigma = (c_1, \underline{s})$ и вычисляет $M = \mathcal{H}(\mathfrak{m}^*)$. Верификатор вычисляет каждого $H_\ell = \mathcal{H}_{\text{ki}}(P_\ell)$. Для каждого $1 \le \ell \le r$ верификатор находит $L'_\ell = s_\ell G + c_\ell P_\ell$, $R'_\ell = s_\ell H_\ell + c_\ell J$. Верификатор использует это для вычисления $(\ell+1)^{th}$ обязательства $c_{\ell+1} = \mathcal{H}_{\text{sig}}(M, L'_\ell, R'_\ell)$. После вычисления $\underline{c} = (c_2, c_3, \dots, c_r, c_{r+1})$ верификатор указывает индекс $r+1$ с индексом 1 и проверяет, чтобы $c_{r+1} = c_1$. Если это так, то верификатор убеждается в том, что подпись является подлинной. Верификатор может проверить, подписаны или нет две подписи одним и тем же ключом, просто сравнив два образа ключей.

3.2 Определение порогового значения для обратных LSAG-подписей с накоплением ключей в стиле Musig

В данном разделе Элис и Боб хотят общими усилиями построить версию LSAG-подписи с порогом 2 из 2, чтобы отправить ключ Шарлин. Подписи верифицируются точно так же, как и раньше, то есть не известно, были ли они созданы группой или единственным пользователем. Мы продолжаем подобно тому, как делали до этого, внося изменения в соответствии со следующими эвристическими процедурами.

Ключи являются суммами: Ключи траты заменяются линейными комбинациями долей ключей траты.

Данные подписания являются суммами: Данные подписания (случайные) заменяются суммами.

Обязательство и раскрытие: Перед раскрытием данных подписания появляется этап обязательства.

Создание префиксов из ключей: Добавление члена кольца в каждый вызов подписи.

В Musig функцией накопления ключей является $\phi(b_i, (\underline{A}, \underline{B})) := \mathcal{H}_{\text{agg}}(B_i, (\underline{A}, \underline{B}))$. Следует отметить, что вычисление $\phi(b_i, (\underline{A}, \underline{B}))$ не требует секрета b_i. Элис берёт новый приватный пользовательский ключ (a_1, b_1), вычисляет публичный ключ $A_1 = a_1 G$, $B_1 = b_1 G$ и отправляет (a_1, B_1) Бобу по защищённому стороннему каналу. Боб делает то же самое, выбирая (a_2, b_2) и отправляя (a_2, B_2) Элис. Они вычисляют приватный общий (совместно используемый) ключ просмотра $a_{\text{sh}} = a_1 + a_2$ и публичный общий ключ траты

$$B_{\text{sh}} = \mathcal{H}_{\text{agg}}(B_1, (\underline{A}, \underline{B}))B_1 + \mathcal{H}_{\text{agg}}(B_2, (\underline{A}, \underline{B}))B_2 = \beta_1 B_1 + \beta_2 B_2.$$

Как вариант, Элис и Боб могли бы вычислить приватный общий ключ просмотра a_{sh}, используя любое количество методов вычисления общего секрета.

Элис и Боб получают сообщение $\mathfrak{m}$, содержащее некоторые публичные ключи подписания (T, P), адресованные $(A_{\text{sh}}, B_{\text{sh}})$ таким образом, чтобы $P = \mathcal{H}_{\text{sess}}(a_{\text{sh}} T)G + B_{\text{sh}}$. Элис и Боб хотят передать это вместе Шарлин, у которой имеется публичный пользовательский ключ (A', B'), с некоторым сообщением с кольцевой мультиподписью $\mathfrak{m}'$. Элис и Боб решают делать это по стороннему каналу с некоторым кольцом $\underline{P} = (P_1, \ldots, P_r)$ таким образом, чтобы для секретного индекса π выполнялось условие $P_\pi = P$. По стороннему каналу Элис и Боб вычисляют образ ключа

$$J = \left(\underbrace{\mathcal{H}_{\text{sess}}(a_{\text{sh}} T)}_{\text{общий вид}} + \underbrace{b_1 \mathcal{H}_{\text{agg}}(B_1, (\underline{A}, \underline{B}))}_{\text{первый участник}} + \underbrace{b_2 \mathcal{H}_{\text{agg}}(B_2, (\underline{A}, \underline{B}))}_{\text{второй участник}} \right) \mathcal{H}_{\text{ki}}(P).$$

Элис и Боб также берут новый приватный ключ транзакции $t' \leftarrow \mathbb{Z}_{\mathfrak{p}}$ (выбор самостоятельно осуществляется кем-то из членов, или же он каким-либо образом согласуется участниками, как в случае с общим ключом просмотра). Элис и Боб вычисляют для Шарлин новый публичный ключ сессии $P' = \mathcal{H}_{\text{sess}}(t'A')G + B'$. Элис и Боб вычисляют базовую точку для каждого члена кольца, $H_\ell = \mathcal{H}_{\text{ki}}(P_\ell)$. Затем Элис и Боб выполняют алгоритм кольцевого подписания, предполагающий реализацию следующих этапов:

Обязательство: Элис выбирает $u_1 \xleftarrow{\$} \mathbb{Z}_{\mathfrak{p}}$, а Боб выбирает $u_2 \xleftarrow{\$} \mathbb{Z}_{\mathfrak{p}}$. Элис вычисляет временную пару точек $(L_{1,\pi}, R_{1,\pi}) = (u_1 G, u_1 H_\pi)$, а Боб вычисляет временную пару точек $(L_{2,\pi}, R_{2,\pi}) = (u_2 G, u_2 H_\pi)$. Элис выбирает случайные секретные скалярные величины $\underline{s}^{(1)} = \{s_{1,\ell}\}_{\ell \neq \pi}$, а Боб выбирает случайные секретные скалярные величины $\underline{s}^{(2)} = \{s_{2,\ell}\}_{\ell \neq \pi}$. Элис вычисляет её частичный образ ключа $J_1 = b_1 \mathcal{H}_{\text{agg}}(B_1, (\underline{A}, \underline{B}))\mathcal{H}_{\text{ki}}(P)$, и Боб вычисляет свой частичный образ ключа $J_2 = b_2 \mathcal{H}_{\text{agg}}(B_2, (\underline{A}, \underline{B}))\mathcal{H}_{\text{ki}}(P)$. Элис вычисляет обязательство $\texttt{com}_1 = \mathcal{H}_{\text{com}}(L_{1,\pi}, R_{1,\pi}, \underline{s}^{(1)})$, и Боб вычисляет обязательство $\texttt{com}_2 = \mathcal{H}_{\text{com}}(L_{2,\pi}, R_{2,\pi}, \underline{s}^{(2)})$. Элис отправляет Бобу $(J_1, \texttt{com}_1)$, а Боб отправляет Элис $(J_2, \texttt{com}_2)$.

Раскрытие: После получения $(J_2, \texttt{com}_2)$ Элис отправляет $(L_{1,\pi}, R_{1,\pi}, \underline{s}^{(1)})$ Бобу; после получения $(J_1, \texttt{com}_1)$ Боб отправляет $(L_{2,\pi}, R_{2,\pi}, \underline{s}^{(2)})$ Элис. После получения $(L_{2,\pi}, R_{2,\pi}, \underline{s}^{(2)})$ Элис проверяет открывается ли обязательство как $\texttt{com}_2 = \mathcal{H}_{\text{com}}(L_{2,\pi}, R_{2,\pi}, \underline{s}^{(2)})$. Если нет, Элис выводит

$\perp$ и прекращает транзакцию. После получения $(L_{1,\pi}, R_{1,\pi}, \underline{s}^{(1)})$ Боб проверяет, чтобы $\mathtt{com}_1 = \mathcal{H}_{\mathrm{com}}(L_{1,\pi}, R_{1,\pi}, \underline{s}^{(1)})$. Если это не так, Боб выводит $\perp$ и прекращает транзакцию.

Предварительное вычисление подписи: Элис и Боб вычисляют общий образ ключа $J = J_1 + J_2 + \mathcal{H}_{\mathrm{sess}}(a_{\mathrm{sh}}T)$, собирают изменённое сообщение $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, (T', P'))$, вычисляют $M = \mathcal{H}(\mathfrak{m}^*)$, вычисляют суммы $L_\pi = L_{1,\pi} + L_{2,\pi}$, $R_\pi = R_{1,\pi} + R_{2,\pi}$, а также $s_\ell = s_{1,\ell} + s_{2,\ell}$ для каждого $\ell \neq \pi$. Затем Элис и Боб могут вычислить последовательные обязательства $c_{\ell+1} := \mathcal{H}_{\mathrm{sig}}(M, P_\ell, L_\ell, R_\ell)$ и продолжить с индексами $\ell = \pi+1, \pi+2, \ldots, \pi-1$ с $L_\ell = s_\ell G + c_\ell P_\ell$ и $R_\ell = s_\ell H_\ell + c_\ell J$. Частичную подпись $\widehat{\sigma} = (c_1, \{s_\ell\}_{\ell \in [r]\setminus\pi})$ можно сохранить на потом.

Полная подпись: Элис вычисляет $s_{1,\pi} = u_1 - c_\pi \beta_1 b_1$. Боб вычисляет $s_{2,\pi} = u_2 - c_\pi \beta_2 b_2$. Элис отправляет $s_{1,\pi}$ Бобу, а Боб отправляет $s_{2,\pi}$ Элис. Оба могут вычислить $s_\pi = s_{1,\pi} + s_{2,\pi}$ и опубликовать полную подпись $\sigma = (c_1, \underline{s})$ с изменённым сообщением $\mathfrak{m}^*$.

Следует отметить, что при вычислении подписи используется член P_ℓ в форме предварительного образа (сравните с разделом 3.1); в результате, в случае с нашим доказательством безопасности, запросы оракула происходят в безопасном порядке.

4 Связываемые thring-подписи и их реализация

Определение 4.0.1. *Связываемая thring-подпись* является результатом совместного вычисления при помощи четырёх алгоритмов полиномиального времени $(\mathtt{KeyGen}, \mathtt{Sign}, \mathtt{Ver}, \mathtt{Link})$. Мы не принимаем во внимание обозначение общего входа η, параметр безопасности в нашем описании:

1. $\mathtt{KeyGen}$ производит в качестве выхода новое произвольное значение $x \xleftarrow{\$} \mathbb{Z}_{\mathfrak{p}}$, вычисляет $X = xG$ и выдаёт $\mathtt{out}_{\mathtt{KeyGen}} = (x, X)$.

2. $\mathtt{Sign}$ является многосторонним алгоритмом, реализуемым участниками с приватными ключами $\underline{x} = \{x_i\}_{i=1}^n$. Каждый участник использует свой ключ x_i в качестве приватного входа, и все участники используют некоторый общий вход $\mathtt{inp}_{\mathtt{Sign}} = (\mathfrak{m}, \underline{P}, \pi)$, где $\mathfrak{m} \in \{0,1\}^*$, $\underline{P} = \{P_i\}_{i=1}^r$ является кольцом публичных ключей, а π является секретным индексом, удовлетворяющим условие $1 \leq \pi \leq r$. $\mathtt{Sign}$ выдаёт либо выделенный символ отказа $\mathtt{out}_{\mathtt{Sign}} = \perp_{\mathtt{Sign}}$для каждого участника, либо некоторый $\mathtt{out}_{\mathtt{Sign}} = (\mathfrak{m}^*, \sigma)$ для каждого участника, где σ является кольцевой подписью, а $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, \mathtt{aux}_{\mathtt{Sign}})$ для тега связываемости J и некоторых вспомогательных данных $\mathtt{aux}_{\mathtt{Sign}}$.

3. Ver берёт в качестве входа некоторый $\mathtt{inp}_{\mathtt{Ver}} = (\mathfrak{m}^*, \sigma)$ и выводит бит $\mathtt{out}_{\mathtt{Ver}} = b \in \{0,1\}$.

4. Link берёт в качестве входа некоторый $\mathtt{inp}_{\mathtt{Link}} = (\mathfrak{m}_1^*, \sigma_1)$, $(\mathfrak{m}_2^*, \sigma_2)$ и выводит бит $\mathtt{out}_{\mathtt{Link}} = b \in \{0,1\}$.

Мы включаем некоторые дополнительные данные в изменённое сообщение, чтобы, например, упаковать ключи получателей в $\mathfrak{m}^*$, несмотря на то, что при этом мы не используем $\mathtt{aux}$ напрямую.

Определение 4.0.2 (Правильность)**.** Для любого $\mathfrak{m}^* \in \{0,1\}^*$ и любого $\sigma \in \mathbb{Z}_{\mathfrak{p}}^*$ обозначим событие, где $\mathtt{Ver}(\mathfrak{m}^*, \sigma) = 1$ как $V(\mathfrak{m}^*, \sigma)$. Для любого $((\mathfrak{m}, \underline{P}, \pi, \underline{x}), (\mathfrak{m}^*, \sigma))$ обозначим событие, где $\mathtt{Sign}(\mathfrak{m}, \underline{P}, \pi, \underline{x}) = (\mathfrak{m}^*, \sigma)$ и $P_\pi = \Phi(\underline{X})$, как $S((\mathfrak{m}, \underline{P}, \pi, \underline{x}), (\mathfrak{m}^*, \sigma))$. Мы говорим, что схема thring-подписи *правильна* если при измерении вероятности по всем входящим монетам и по каждому выбору хеш-функций $\mathbb{P}[V(\mathfrak{m}^*, \sigma) \mid S((\mathfrak{m}, \underline{P}, \pi, \underline{x}), (\mathfrak{m}^*, \sigma))] = 1$ для любого $(\mathfrak{m}, \underline{P}, \pi, \underline{x})$.

Определение 4.0.3 (Связываемость)**.** Для любой пары подписей, $(\mathfrak{m}_1^*, \sigma_1)$ и $(\mathfrak{m}_2^*, \sigma_2)$ обозначим событие, где $\mathtt{Link}(\mathfrak{m}_1^*, \sigma_1, \mathfrak{m}_2^*, \sigma_2) = 1$, как $L(\mathfrak{m}_1^*, \sigma_1, \mathfrak{m}_2^*, \sigma_2)$. Мы определяем подсобытие $S'(\mathfrak{m}_1^*, \sigma_1, \mathfrak{m}_2^*, \sigma_2) \subseteq V(\mathfrak{m}_1^*, \sigma_1) \cap V(\mathfrak{m}_2^*, \sigma_2)$ как событие, в котором есть некоторый $\underline{x}$, некоторые сообщения $\mathfrak{m}_1$, $\mathfrak{m}_2$, некоторые кольца $\underline{P}_1, \underline{P}_2$ и некоторые индексы π_1, π_2 удовлетворяющие условия $\mathtt{Sign}(\mathfrak{m}_1, \underline{P}_1, \pi_1, \underline{x}) = (\mathfrak{m}_1^*, \sigma_1)$ и $\mathtt{Sign}(\mathfrak{m}_2, \underline{P}_2, \pi_2, \underline{x}) = (\mathfrak{m}_2^*, \sigma_2)$. Мы говорим, что схема thring-подписи *связываема*, если для любого $(\mathfrak{m}_1^*, \sigma_1, \mathfrak{m}_2^*, \sigma_2)$,

$$\mathbb{P}\left[L(\mathfrak{m}_1^*, \sigma_1, \mathfrak{m}_2^*, \sigma_2) \mid S'(\mathfrak{m}_1^*, \sigma_1, \mathfrak{m}_2^*, \sigma_2)\right] = 1,$$

где эта вероятность измерена для монет всех участников и по каждому выбору хеш-функций.

Пример 4.0.4 (LSTAG-подписи)**.** Мы представляем схему связываемой thring-подписи, основанную на LSAG-подписях, описанных в работе [13]. Мы называем эту схему *связываемой спонтанной пороговой подписью анонимной группы* или LSTAG-подписью.

Мы агрегируем ключи, используя подход Musig, и задаём $\Phi(\underline{X}) = \sum_{i\in[n]} \beta_i X_i$, где $\beta_i = \phi(x_i, \underline{X}) = \mathcal{H}_{\text{agg}}(X_i, \underline{X})$. Чтобы гарантировать, что каждый участник будет вычислять ключи последовательно, мы допускаем, что у каждого пользователя есть на некоторой фазе настройки согласованный канонический линейный порядок ключей в $\underline{X}$, такой как побитовое лексикографическое упорядочивание, начиная с младшего, i^{th} член имеет часть приватного ключа $x_i^* = \beta_i x_i = \phi(x_i, \underline{X}) x_i$. Образ ключа вычисляется так же, как это обычно делается в случае с Monero: для любого приватного ключа $x \in \mathbb{Z}_{\mathfrak{p}}$ образом ключа будет $x\mathcal{H}_{\text{ki}}(xG)$, таким образом, образом ключа X_{sh} точно будет $\sum_i \beta_i x_i \mathcal{H}_{\text{ki}}(X_{\text{sh}})$.

1. `KeyGen` случайным образом выбирает некоторый $x \in \mathbb{Z}_{\mathfrak{p}}$, вычисляет $X := xG$ и выводит (x, X).

2. `Sign` запускается по стороннему каналу, после того как группа придёт к соглашению по сообщению $\mathfrak{m}$, по кольцу $\underline{P}$ и секретному индексу π, произведёт предварительное вычисление базовых точек образа ключа для каждого члена кольца $H_\ell := \mathcal{H}_{\text{ki}}(P_\ell)$, вычислит образ ключа $J = \sum_j J_j = \sum_j x_j^* H_\pi$, соберёт изменённое сообщение $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, \mathtt{aux})$ и вычислит $M = \mathcal{H}_{\text{msg}}(\mathfrak{m}^*)$. Все остальные действия выполняются совместно:

   Обязательство: каждый подписант, допустим, с индексом j, удовлетворяющим условие $1 \le j \le n$, делает следующее:

   (a) выбирает случайную скалярную величину u_j, вычисляет точки $U_j = u_j G$ и $V_j = u_j H_\pi$ и выбирает случайные скалярные величины $s_{\pi+1,j}, s_{\pi+2,j}, \ldots, s_{\pi-1,j}$;

   (b) задаёт $\mathtt{dat}_j := (U_j, V_j, \{s_{\ell,j}\}_{\ell\neq\pi})$;

   (c) вычисляет обязательство $\mathtt{com}_j = \mathcal{H}_{\text{com}}(\mathtt{dat}_j)$;

   (d) отправляет $\mathtt{com}_j$ всем остальным подписантам.

   Раскрытие: После получения $\mathtt{com}_j$ от остальных членов объединения каждый подписант, обладающий тем же индексом, что и раньше, делает следующее:

   (a) отправляет $\mathtt{dat}_j$ всем остальным подписантам;

   (b) после того, как все $\mathtt{dat}_{j'}$ были получены, проверяет, чтобы $\mathcal{H}_{\text{com}}(\mathtt{dat}_{j'}) = \mathtt{com}_{j'}$ для каждого $j \neq j'$. Если не все обязательства открываются надлежащим образом, выдаёт $\perp_{\mathtt{Sign}}$ и завершает процесс.

   Предварительная офлайн обработка подписи: каждый подписант, обладающий тем же индексом, что и раньше, делает следующее:

   (a) вычисляет $L_\pi = \sum_j U_j$, $R_\pi = \sum_j V_j$;

   (b) вычисляет каждое $s_\ell = \sum_j s_{\ell,j}$ для каждого $1 \le \ell \le r$ так, чтобы $\ell \neq \pi$;

(c) для каждого $\ell = \pi, \pi+1, \ldots, \pi-1$ (где мы переназначаем избыточные индексы, преобразуя $r+1 \mapsto 1$, $r+2 \mapsto 2$ и так далее), вычисляем следующее:

$$\begin{aligned} c_{\ell+1} =& \mathcal{H}_{\text{sig}}(M, P_\ell, L_\ell, R_\ell) \\ L_{\ell+1} =& s_{\ell+1}G + c_{\ell+1}P_{\ell+1} \\ R_{\ell+1} =& s_{\ell+1}H_{\ell+1} + c_{\ell+1}J \end{aligned}$$

(d) сохраняет предварительно обработанную подпись $(u_j, \mathfrak{m}^*, c_1, c_\pi, \{s_\ell\}_{\ell\neq\pi})$ для последующей работы.

Завершение подписи: чтобы завершить подписание, используя $(u_j, \mathfrak{m}^*, c_1, c_\pi, \{s_\ell\}_{\ell\neq\pi})$, каждый подписант, обладающий тем же индексом, что и раньше, делает следующее:

(a) вычисляет $s_{\pi,j} = u_j - c_\pi x_j^*$;

(b) отправляет $s_{\pi,j}$ другим подписантам;

(c) после получения всех $\{s_{\pi,j'}\}_{j\neq j'}$ вычисляет $s_\pi = \sum_j s_{\pi,j}$;

(d) выводит $(\mathfrak{m}^*, \sigma)$, где $\sigma = (c_1, \underline{s})$, где $\underline{s} = (s_1, s_2, \ldots, s_r)$.

3. `Ver` берёт в качестве входа некоторый $(\mathfrak{m}^*, \sigma)$.

(a) Разбирает $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, \texttt{aux})$ и $\sigma = (c_1, \underline{s})$;

(b) Для $\ell = 1, 2, \ldots, r-1$ вычисляет $L_\ell = s_\ell G + c_\ell P_\ell$, $R_\ell = s_\ell H_\ell + c_\ell J$ и $c_{\ell+1} = \mathcal{H}_{\text{sig}}(M, P_\ell, L_\ell, R_\ell)$;

(c) Вычисляет $c_1' = \mathcal{H}_{\text{sig}}(M, P_r, L_r, R_r)$;

(d) Выводит 1, если $c_1' = c_1$, и 0 в противном случае.

4. `Link` работает так же, как в случае с «обратной» LSAG-подписью: проверяет соответствие образов ключей J.

5 Невозможность подделки и thring-подписи

5.1 Определение подделки

Что точно означает слово «поделка», или что значит «сделать подделку»? Подделкой будет некоторое $(\mathfrak{m}^*, \sigma)$, где σ не будет выходом в записи запросов, отправленных $\mathcal{A}$ для $\mathcal{SO}$ и $\texttt{Ver}(\mathfrak{m}^*, \sigma) = 1$. Тем не менее этого недостаточно. На самом деле, если ни один из ключей не будет агрегирован в $\underline{P}$, поделка нашей подписи сведётся к подделке лежащей в её основе схемы LSAG; без потери общности успешная поделка подразумевает наличие по крайней мере одного агрегированного ключа в $\underline{P}$. Кроме того, если будет агрегирован какой-то ключ, тот, кто занимается подделкой, мог бы просто поместить свой собственный ключ в $\underline{P}$ вместе с агрегированным ключом. Таким образом, если злоумышленнику будет известен дискретный логарифм любого публичного ключа в некотором кольце $\underline{P}$, то он сможет создать только честную подпись, используя $\underline{P}$, которая не может считаться подделкой. Без потери общности все участники кольца должны быть либо честными ключами (то есть задачей на основе дискретного логарифмирования), либо дочерним элементом честного ключа. Если самый мощный злоумышленник повредит все ключи, кроме одного, мы допустим наличие только одного целевого честного ключа X_h.

Следовательно, мы изменяем правила обычной игры в невозможность подделки: при наличии ключа X_h и кольца $\underline{P}$ подделка станет успешной только в том случае, если будет сопровождаться свидетельством того, что каждый участник кольца либо является самим ключом X_h, либо дочерним элементом

X_h, а также, что в $\underline{P}$ присутствует по крайней мере один дочерний элемент X_h. Злоумышленник может просто создать свидетельство этих отношений, представив агрегируемые наборы $\left\{\underline{X}^{(\ell)}\right\}_{\ell\in[r]}$ таким образом, чтобы $X_h \in \underline{X}^{(\ell)}$ и $P_\ell = \Phi(\underline{X}^{(\ell)})$ для каждого $1 \le \ell \le r$. В дальнейшем злоумышленник является PPT алгоритмом $\mathcal{A}$, который берёт в качестве входа некоторый ключ X_h и выдаёт в качестве выхода выделенный символ ошибки $\perp_{\mathcal{A}}$ или успешную подделку $\mathtt{out}_{\mathcal{A}} = \mathtt{forg} = (\mathfrak{m}^*, \sigma, \left\{\underline{X}^{(\ell)}\right\}_{\ell\in[r]})$.

Определение 5.1.1 (Реальная невозможность подделки LSTAG)**.** Мы говорим, что PPT алгоритм $\mathcal{A}$ является подделкой (t, ϵ, q, n), если за время t как самое большее и за q запросов оракула $\mathcal{A}$ будет успешно реализован при следующем сценарии с вероятность, составляющей по крайней мере ϵ.

1. Запросчик берёт пару честных ключей $(x_h, X_h) \leftarrow \mathtt{KeyGen}$ и отправляет публичный ключ X_h алгоритму $\mathcal{A}$.

2. $\mathcal{A}$ может генерировать ключи, может собрать любую группу элементов, используя X_h, и получает доступ к подписывающему оракулу $\mathcal{SO}$ и случайным оракулам $\mathcal{H}_{\text{agg}}$, $\mathcal{H}_{\text{sig}}$, $\mathcal{H}_{\text{com}}$, $\mathcal{H}_{\text{msg}}$ и $\mathcal{H}_{\text{ki}}$. $\mathcal{A}$ может использовать любой из них в любом порядке, адаптивно отвечая на предыдущие результаты.

3. $\mathcal{A}$ выводит некоторое $(\mathfrak{m}^*, \sigma, \left\{\underline{X}^{(\ell)}\right\}_{\ell=1}^{r})$.

4. $\mathcal{A}$ выигрывает, если все следующие условия будут соблюдены:

   Правильность: для каждого ℓ, $P_\ell = \Phi(\underline{X}^{(\ell)})$.

   Ограниченность: для каждого ℓ, $1 \le \left|\underline{X}^{(\ell)}\right| \le n$.

   Честный родитель: для каждого ℓ, $X_h \in \underline{X}^{(\ell)}$.

   Наличие накопленных ключей: для некоторого ℓ, $\left|\underline{X}^{(\ell)}\right| \ge 2$ (накоплен по крайней мере один ключ).

   Нетривиальность: σ не является выходом в транскрипте между $\mathcal{A}$ и $\mathcal{SO}$; и, конечно же

   Корректность: $\mathtt{Ver}(\mathfrak{m}^*, \sigma) = 1$.

Несмотря на то, что для злоумышленника представляется нереальным предоставление свидетельства подделки, нам бы хотелось отметить, что если подделка будет скрыта в «чёрный ящик» при помощи какого-либо верховного алгоритма, запросы накопления ключей, которые будет делать фальсификатор, должны быть смоделированы или также созданы таким верховным алгоритмом. Следовательно, свидетельство этих отношений может быть выделено из транскрипта, в результате чего будет сделана успешная подделка.

5.2 Стратегия доказательства невозможности подделки

Предположим, что $\mathcal{B}$ является метасокращением $\mathcal{A}$, которое выдаёт для некоторого фиксированного сообщения $\mathfrak{m}$ четыре подделанные подписи σ, σ', σ'', σ''' с кольцами $\underline{P}$, $\underline{P}'$, $\underline{P}''$, $\underline{P}'''$ с семейными историями $\underline{X} = \left\{\underline{X}^{(\ell)}\right\}_{\ell\in[r]}$, $\underline{X}' = \left\{(\underline{X}^{(\ell)})'\right\}_{\ell\in[r']}$, $\underline{X}'' = \left\{(\underline{X}^{(\ell)})''\right\}_{\ell\in[r'']}$, and $\underline{X}''' = \left\{(\underline{X}^{(\ell)})'''\right\}_{\ell\in[r''']}$. Кроме того, предположим, что $\mathcal{B}$ может выделить из оракулов, созданных $\mathcal{A}$, выделенный индекс ℓ таким образом, чтобы

$$L_\ell = L'_\ell, L''_\ell = L'''_\ell, P_\ell = P'_\ell, P''_\ell = P'''_\ell.$$

Затем $\mathcal{B}$ может вычислить дискретный логарифм P_ℓ как $(c_\ell - c'_\ell)^{-1}(s'_\ell - s_\ell)$ и дискретный логарифм P''_ℓ как $(c''_\ell - c'''_\ell)^{-1}(s'''_\ell - s''_\ell)$. Тем не менее ключи $P_\ell = P'_\ell$ агрегируются из $\underline{X}$ и $\underline{X}'$, соответственно. Поэтому мы можем записать $P_\ell = P'_\ell = \alpha X_h + Z$ для некоторого α и некоторого Z, используя функцию агрегирования Ф. Подобным образом $P''_\ell = P'''_\ell$ может быть записано как $\alpha' X_h + Z'$ для некоторых α', Z'.

Если $\mathcal{B}$ может гарантировать, что $Z = Z'$ и $\alpha \neq \alpha'$, то $P_\ell - P''_\ell = (\alpha - \alpha')X_h$ и $\mathcal{B}$ могут получить дискретный логарифм X_h:

$$x_h = (\alpha - \alpha')^{-1} \left(\frac{s'_\ell - s_\ell}{c_\ell - c'_\ell} - \frac{s'''_\ell - s''_\ell}{c''_\ell - c'''_\ell}. \right)$$

Следовательно, чтобы продемонстрировать абсурдность существования фальсификатора $\mathcal{A}$, достаточно указать на существование метасокращения, которое может произвести четыре транскрипта, из которых затем можно будет извлечь следующее: индекс $1 \leq \ell$, некоторые публичные ключи $(P_\ell, P'_\ell, P''_\ell, P'''_\ell, L_\ell, L'_\ell, L''_\ell, L'''_\ell, Z, Z')$ и некоторые скалярные величины $(s_\ell, s'_\ell, s''_\ell, s'''_\ell, c_\ell, c'_\ell, c''_\ell, c'''_\ell, \alpha, \alpha')$ так, чтобы:

$$\begin{aligned}
L_\ell &= s_\ell G + c_\ell P_\ell & L'_\ell &= s'_\ell G + c'_\ell P'_\ell \\
L''_\ell &= s''_\ell G + c''_\ell P''_\ell & L'''_\ell &= s'''_\ell G + c'''_\ell P'''_\ell \\
L_\ell &= L'_\ell & L''_\ell &= L'''_\ell \\
P_\ell &= P'_\ell & P''_\ell &= P'''_\ell \\
P_\ell &= \alpha X_h + Z & P''_\ell &= \alpha' X_h + Z'
\end{aligned}$$

и так, чтобы $c''_\ell \neq c'''_\ell$, $c_\ell \neq c'_\ell$ и $\alpha \neq \alpha'$. Мы называем это *системой уравнений и неравенств подделки по дискретному логарифму.*

В Приложении А мы показываем, как это делается с тщательным построением оракулов и двумя форками: самый первый запрос верификации подписи в форме $\mathcal{H}(M, P_\ell, L_\ell, R_\ell)$, гарантирующий, что $L_\ell = L'_\ell$ и $L''_\ell = L'''_\ell$, и второй — после вычисления коэффициентов накопления ключей, гарантирующий, что $Z_\ell = Z'_\ell = Z$ and $\alpha \neq \alpha'$.

6 Правильное и не правильное использование вариантов применения и реализации

В этом разделе нами будут рассмотрены некоторые варианты реализации схем thring-подписей, их расширения, а также их применения в кольцевых конфиденциальных транзакциях.

6.1 Опасность не случайного или повторного подписания

Протокол становится опасным, если данные генерируются не случайным образом при каждой попытке создания подписи, или же если публикуется более одной подписи на ключ. Если одни и те же данные подписи $u_{j,\pi}$ используются стороной-участником дважды, существует риск раскрытия приватных ключей: равенства $s = u - cb^*$ и $s' = u - c'b^*$ могут использоваться для вычисления $b^* = \frac{s-s'}{c'-c}$. Следовательно, никогда не следует использовать какой-либо не случайный метод выбора данных подписи. Подобным образом никогда не следует использовать две подписи с одним и тем же ключом сессии, так как появляется возможность извлечения логарифма подписывающего ключа.

6.2 Вопросы свойств группы

Также следует отметить идеальную безопасность использования группы $\mathbb{G}$ *сложного*, а не простого порядка. Тем не менее мы должны ограничить наш выбор публичного ключа подгруппой $\mathbb{G}$ определён-

ного простого порядка. При использовании так называемой кривой Ed25519 (которая также называется «скрученной кривой Эдвардса», описана в работе [6] и бирационально эквивалентна так называемой кривой Curve25519, описанной в работе [5]) существуют некоторые риски, связанные с реализацией при выборе публичного ключа за пределами подгруппы простого порядка.

И в самом деле, подгруппы простого порядка имеют кофактор 8, что обеспечивает возможность их использования в своих корыстных целях. Чтобы избежать этого, все варианты реализации, использующие эту кривую, требуют проверки, чтобы группа элементов, используемая в качестве публичных ключей, находилась в пределах подгруппы простого порядка. Для этого проверяется их порядок. Определённо, любой приватный ключ $x \in \mathbb{Z}_{\mathfrak{p}}$ будет иметь соответствующий публичный ключ $X = xG$ для этой подгруппы простого порядка, так как G является генератором этой подгруппы. С другой стороны, случайный выбор публичного ключа не гарантирует дискретного логарифмирования предварительного образа относительно G. Нам необходимо изменить функцию хеширования в точку $\mathcal{H}_{\text{ki}}$, чтобы получить кообласть, равную подгруппе простого порядка. Это означает умножение публичных ключей без соответствующих приватных ключей на кофактор 8 при использовании кривой Ed25519 или Curve25519.

6.3 Расширение ключа просмотра и конфиденциальные thring-транзакции

Среди прочих причин можно указать на то, что наши thring-подписи нельзя напрямую сравнить с подписями, используемыми в криптовалютах, такими как MLSAG-подписи, которые используются в протоколах подобных CryptoNote. В самом деле, пользовательские ключи CryptoNote идут парами с ключом просмотра и ключом траты, а подписи вычисляются при помощи одноразовых ключей, выводимых из них. Эвристические принципы пороговой обработки, описанные в Разделе 3.2, естественным образом распространяются на систему с этим расширением одноразовых ключей, но наши доказательства свойств безопасности не применяются непосредственно в отношении этих расширений, если их не доработать. Обеспечение невозможности подделки варианта реализации LSTAG thring-подписей, использующих одноразовые ключи в стиле CrypytoNote, с расширением ключа просмотра остаётся открытой задачей.

Одной из моделей, которая могла бы помочь доказать невозможность подделки расширения ключа просмотра в рамках предлагаемого нами ниже варианта реализации, является получение каждой из сторон-участников общего секрета $y = a_{\text{sh}}$ и использование ключей $\underline{x} = (x_1, \ldots, x_n)$ для вычисления общего ключа в форме $Y + \Phi(\underline{X})$. Атаки с кражей ключа или его отменой при использовании этого метода можно избежать путём тщательного построения y. Например, в случае с протоколами типа CryptoNote фальсификация Y требует фальсификации хеш-суммы ключа, полученного при обмене по протоколу Диффи-Хеллмана.

Подобным образом наш эвристический подход таким же естественным образом распространяется на MLSAG, но и тут наши доказательства безопасности нельзя будет применить напрямую, не доработав. Мы кратко описываем кольцевые конфиденциальные транзакции Monero и пороговое расширение конфиденциальных thring-транзакций.

MLSAG-подписи строятся на основе векторов ключей в форме $(\mathcal{H}_{\text{sess}}(aT)G + B, \texttt{PedCom}(v, r))$, где `PedCom` является схемой обязательства Педерсена, v - суммой транзакции, а r является приватным ключом от суммы-обязательства. MLSAG-подписи по своей структуре являются мультиподписями, так как используют множество ключей для подписания сообщения, но они вычисляются не совместными усилиями. Размер подписи в этом случае не зависит от количества подписантов, но MLSAG-подписи всё же раскрывают количество подписывающих ключей. Интерпретируя список подписывающих ключей

подписантов $\underline{P} = (P_1, \ldots, P_n)$ как вектор, подписант (или подписанты) случайно выбирает схожие векторы из блокчейна для построения кольца из таких векторов ключей $\widetilde{\underline{P}}$, объединяя $\underline{P}$ в столбец π^{th} секрета π.

$$\widetilde{\underline{P}} = \begin{pmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,r} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,r} \\ \vdots & & & \vdots \\ P_{n,1} & P_{n,2} & \cdots & P_{n,r} \end{pmatrix} = (\underline{P}_1, \ldots, \underline{P}_r)$$

где каждое $\underline{P}_\ell = \{P_{j,\ell}\}_{j\in[n]}$. Для каждого $j = 1, 2, \ldots, n$ и $\ell = 1, 2, \ldots, r$ с составляющей $P_{j,\ell}$ в $\widetilde{\underline{P}}$ мы вычисляем $H_{j,\ell} := \mathcal{H}_{\text{ki}}(P_{j,\ell})$. Для каждого компонента $p_j \in \underline{p}$ подписант вычисляет образ ключа $J_j = p_j \mathcal{H}_{\text{ki}}(p_j G)$. Подписант выбирает вектор случайных скалярных величин $\underline{u} = (u_1, \ldots, u_n) \in \mathbb{Z}_{\mathfrak{p}}^n$ для столбца π^{th} и для каждого $\ell \neq \pi$, подписант выбирает вектор скалярных величин $\underline{s}_\ell = (s_{1,\ell}, s_{2,\ell}, \ldots, s_{n,\ell})$. Затем для каждого подписывающего ключа $P_{j,\ell}$ подписант вычисляет пару точек и обязательство

$$L_{j,\ell} = s_{j,\ell} G + c_\ell P_{j,\ell}, \qquad R_{j,\ell} = s_{j,\ell} H_{j,\ell} + c_\ell J_j, \qquad c_{\ell+1} = \mathcal{H}_{\text{sig}}\left(M, \left\{(L_\ell^j, R_\ell^j)\right\}_{j=1}^n\right).$$

Как только каждое обязательство будет вычислено, подписант обычным образом вычисляет каждое $s_{j,\pi} = u_j - c_\pi p_j$, собирает $\underline{s}_\pi = (s_{j,\pi})_{j\in[n]}$ и получает MLSAG-подпись $\sigma = (c_1, (\underline{s}_\ell)_{\ell\in[r]})$, которая верифицируется подобно LSAG-подписям.

Изначально эта схема может показаться недостаточно неопределённой в отношении идентификации подписанта. Любой может определить, что один из этих столбцов содержит все ключи подписания. Другими словами, не могут $p_{2,2}$ и $p_{1,1}$ одновременно являться действительными ключами, используемыми в кольцевой подписи. Это, скорее, не недостаток, а то, как Monero связывает ключи подписания с суммами транзакций. В случае с Monero первый ряд ключей в $\widetilde{\underline{P}}$ является ключами подписания, второй ряд составляют разницы между обязательствами Педерсена и по входящим суммам транзакций и выходящим суммам. Подписание с использованием этой матрицы демонстрирует как знание приватного ключа подписания со специальным индексом, так и способность раскрыть обязательство по сумме по этому индексу с точностью до нуля.

Чтобы увидеть, как эвристический подход к пороговому сравнению естественным образом распространяется на MLSAG-подписи, рассмотрим следующее. Для объединения подписантов допустим, что каждый участвующий подписант имеет часть ключа, которая накапливается, как и в случае с Musig, и каждый участник добавляет некоторые случайные скалярные величины, которые суммируются для получения u_j, $s_{j,\ell}$ и так далее на этапе обязательства и раскрытия. Формально доказательство невозможности подделки такого варианта реализации конфиденциальных MLSAG thring-транзакций, использующих одноразовые ключи в стиле CryptoNote при расширении ключа просмотра, также остаётся открытой задачей. Мы оставляем решение этой задачи более общего формального определения и реализации конфиденциальных thring-транзакций и их соответствующих свойств безопасности на будущее, например, в работе [11], которая будет вскоре опубликована.

6.3.1 Расширение до схемы m из n

В разделе 3.2 представлен простой пример реализации схемы 2 из 2 , который естественным образом расширяется до схемы n из n. При помощи перестановки Диффи-Хеллмана мы можем расширить вышеуказанный подход до схемы пороговой подписи $(n-1)$ из n следующим образом: участники делятся друг с другом своими B_j и попарно вычисляют общие секреты $z_{i,j} = \mathcal{H}_{\text{agg}}(b_i B_j)$. Есть $\frac{n(n-1)}{2}$

отдельных общих секретов, совместно используемых n участников, таким образом, $n-1$ участников могут восстановить все секреты. Следовательно, схема $(n-1)$ из n может быть реализована как схема $\frac{n(n-1)}{2}$ из $\frac{n(n-1)}{2}$. Очевидно, существуют более общие подходы к реализации схемы m из n, но мы не коснёмся их далее в этой работе.

6.4 Конфиденциальные атомные свопы между блокчейнами с неопределённым отправителем

Простые атомные свопы между блокчейнами с использованием thring-подписей возможны. Если всё идёт хорошо, своп происходит следующим образом: Элис отправляет x монет AliceCoins в блокчейн AliceCoins с ключом 2 из 2, Боб отправляет y монет BobCoins в блокчейн BobCoins с ключом 2 из 2, и как только обе стороны будут удовлетворены, они смогут забрать свои средства. Безусловно, мы не можем допускать, что всё пройдёт хорошо; как описано в работе [2], транзакции возмещения позволяют не совсем честным сторонам остановить процесс в опасной среде. Остаётся закончить модель проведения атомным свопов между блокчейнами с неопределённым отправителем, описанную в работе [2], для цифровой валюты, использующей схему проведения конфиденциальных транзакций, формализовав возможность транзакций возмещения для такой валюты (см. [17]).

Список литературы

[1] A. Back. Ring signature efficiency (Эффективность кольцевых подписей). `https://bitcointalk.org/index.php?topic=%20972541.msg10619684#msg10619684`, 2015. Accessed: 16-03-2018.

[2] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains (Инновация в области блокчейн технологии – искуственно поддерживаемые побочные блокчейны). *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

[3] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma (Мультиподписи в рамках простой модели публичных ключей и общая лемма реализации форков). In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399. ACM, 2006.

[4] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles (Кольцевые подписи: более строгие определения и конструкции без использования случайных оракулов). In *TCC*, volume 6, pages 60–79. Springer, 2006.

[5] Daniel J Bernstein. Curve25519: new Diffie-Hellman speed records (Кривая curve25519: новые скоростные рекорды Диффи-Хеллмана). In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[6] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures (Высокоскоростные и безопасные подписи). *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[7] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains (Компактные мультиподписи для малых блокчейнов). Cryptology ePrint Archive, Report 1990/001, 2018. `https://eprint.iacr.org/2018/483`.

[8] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption (Пороговые криптосистемы на основе полного порогового гомоморфного шифрования). Cryptology ePrint Archive, Report 1990/001, 2017. `https://eprint.iacr.org/2017/956.pdf`.

[9] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold ring signatures and applications to ad-hoc groups (Пороговые кольцевые подписи и их применение в специализированных группах). In *Annual International Cryptology Conference*, pages 465–480. Springer, 2002.

[10] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, and Gregory Neven. Okamoto beats Schnorr: On the provable security of multi-signatures (Окамото побеждает Шнорра – к вопросу доказуемой безопасности мультиподписей). Technical report, IACR Cryptology ePrint Archive, Report 2018/417, 2018. Available at http://eprint. iacr. org/2018/417, 2018.

[11] Russel WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Foundations of ring confidential transactions (Основы кольцевых конфиденциальных транзакций). *In prep.*, 2018.

[12] Joseph K Liu, Victor K Wei, and Duncan S Wong. A separable threshold ring signature scheme (Схема разделяемых пороговых кольцевых подписей). In *International Conference on Information Security and Cryptology*, pages 12–26. Springer, 2003.

[13] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups (Связываемая подпись спонтанной анонимной группы для случайно создаваемых групп). In *ACISP*, volume 4, pages 325–335. Springer, 2004.

[14] Gregory Maxwell. Confidential transactions (Конфиденциальные транзакции). *URL: https://people. xiph. org/˜ greg/confidential_ values. txt (Accessed 08/24/2018)*, 2015.

[15] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin (Простые мультиподписи Шнорра и их применение в Bitcoin). Cryptology ePrint Archive, Report 1990/001, 2018. `https://eprint.iacr.org/2018/068.pdf`.

[16] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin (zerocoin: анонимная распределённая электронная валюта на базе bitcoin). In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[17] Sarang Noether and BG Goodell. Dual-output ring signatures: Spender-ambiguous cross-chain confidential atomic swaps (Кольцевые подписи с двойным выходом: конфиденциальные атомные свопы между блокчейнами с сокрытием отправителя). *In prep.*, 2018.

[18] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions (Кольцевые конфиденциальные транзакции). *Ledger*, 1:1–18, 2016.

[19] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes (Доказуемо безопасные и практичные схемы идентификации и соответствующие схемы подписей). In *Annual International Cryptology Conference*, pages 31–53. Springer, 1992.

[20] Haifeng Qian and Shouhuai Xu. Non-interactive multisignatures in the plain public-key model with efficient verification (Неинтерактивные мультиподписи в рамках простой модели публичных ключей с эффективной верификацией). *Information Processing Letters*, 111(2):82–89, 2010.

[21] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks (Сильная сторона доказательства владения: обеспечение безопасности подписей с привлечением множества сторон против атак с кражей ключа). In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 228–245. Springer, 2007.

[22] Claus-Peter Schnorr. Efficient signature generation by smart cards (Эффективное генерирование подписей смарт-картами). *Journal of cryptology*, 4(3):161–174, 1991.

[23] Patrick P Tsang, Victor K Wei, Tony K Chan, Man Ho Au, Joseph K Liu, and Duncan S Wong. Separable linkable threshold ring signatures (Разделяемые связываемые пороговые кольцевые подписи). In *Indocrypt*, volume 3348, pages 384–398. Springer, 2004.

A Безопасность

В этом Приложении мы доказываем следующую теорему на основе игры, которая приводится в Определении 5.1.1. В Приложении B нами рассматриваются некоторые другие свойства безопасности кольцевых подписей и их LSTAG варианта реализации. В пункте A.1 мы подробно описываем подписывающий оракул, о котором говорится в Определении 5.1.1, и его моделирование. В пункте A.2 нами рассматривается лемма реализации форков «обратной перемоткой» при успешном выполнении. В пункте A.3 мы обсуждаем первое сокращение возможности подделки и установку некоторых границ вероятности. В пункте A.4 нами разъясняется подход к двойной реализации форка, а также приводится доказательство следующей теоремы.

Теорема A.0.1. Допустим, $\mathcal{A}$ является фальсификатором (t, ϵ, q, n) для некоторого ϵ, которое не является незначительным. Существует некоторое $t' > 0$ и алгоритм $\mathcal{B}$, то есть решатель $(t + t', \epsilon', q)$ некоторой задачи дискретного логарифмирования для некоторого ϵ', которое не является незначительным.

A.1 Подписывающий оракул

В данном разделе мы объясняем, как используется подписывающий оракул в рамках игры, описанной в Определении 5.1.1.

Описывающий оракул, используемый в рамках игры по обеспечению невозможности подделки, предполагает наличие ситуации, когда фальсификатор $\mathcal{A}$ убеждает честную сторону вместе подписать какие-то документы перед тем, как сделать подделку. В случае с мультиподписями нам также необходимо учесть то, что злоумышленник $\mathcal{A}$ будет пытаться убедить честную сторону совместно построить некоторые мультиподписи для схожих документов. Благодаря такому сотрудничеству $\mathcal{A}$ получает некоторый контроль над данными подписи, поэтому необходимо, чтобы $\mathcal{SO}$ был интерактивным. При честном сотрудничестве $\mathcal{A}$ также необходимо знать, что индекс π соответствует действительному подписанту, поэтому мы допускаем, что подписывающий оракул запрашивается со специальным индексом подписания π. Более того, сокращение $\mathcal{A}'$ от $\mathcal{A}$ должно моделировать взаимодействие между $\mathcal{A}$ и любыми другими оракулами, в частности, с подписывающим оракулом.

$\mathcal{SO}$ берёт в качестве входа некоторый $\mathtt{inp}_{\mathcal{SO}} = (\mathfrak{m}, \underline{P}, \pi, \left\{\underline{X}^{(\ell)}\right\}_{\ell\in[r]})$, где $\mathfrak{m} \in \{0,1\}^*$ является сообщением, $\underline{P}$ является кольцом публичных ключей, π является специальным индексом, а $\left\{\underline{X}^{(\ell)}\right\}_{\ell\in[r]}$ является мультимножеством публичных ключей. $\mathcal{SO}$ и $\mathcal{A}$ взаимодействуют. $\mathcal{SO}$ выводит выделенный символ ошибки $\perp_{\mathcal{SO}}$, или же $\mathcal{A}$ успешно моделирует сотрудничество с честной стороной для получения мультиподписи.

Оракул: $\mathcal{A}$ запрашивает $\mathcal{SO}$ при помощи некоторого $\left(M, \underline{P}, \pi, \left\{\underline{X}^{(\ell)}\right\}_{\ell\in[r]}\right)$.

1. $\mathcal{SO}$ проверяет, для каждого ли ℓ будет действительным $P_\ell = \Phi(\underline{X}^{(\ell)})$, а также проверяет соответствие $X_h \in \underline{X}^{(\ell)}$ и проверяет, чтобы было накоплено по крайней мере одно P_ℓ. Если это не так, $\mathcal{SO}$ выводит $\perp_{\mathcal{SO}}$ и завершает процесс.
2. В противном случае, $\mathcal{SO}$ выбирает данные подписания $\mathtt{dat}_1 = (U_1, V_1, \{s_{1,\ell}\}_{\ell\neq\pi})$, вычисляет обязательство $\mathtt{com}_1 \leftarrow \mathcal{H}_{\mathrm{com}}(\mathtt{dat}_1)$ и отправляет $\mathtt{com}_1$ для $\mathcal{A}$.
3. После того как $\mathcal{A}$ ответит множеством $\{\mathtt{com}_j\}_{j=2}^n$, $\mathcal{SO}$ отправляет $\mathtt{dat}_1$ для $\mathcal{A}$.
4. После того как $\mathcal{A}$ ответит $\{\mathtt{dat}_j\}_{j=2}^n$, $\mathcal{SO}$ проверяет, чтобы $\mathtt{com}_j = \mathcal{H}_{\mathrm{com}}(\mathtt{dat}_j)$ для каждого j. Если это не так, $\mathcal{SO}$ выводит $\perp_{\mathcal{SO}}$ и завершает процесс.
5. $\mathcal{SO}$ завершает этап предварительной офлайн обработки подписи, решает $s_{\pi,j}$ и отправляет $s_{\pi,j}$ для $\mathcal{A}$.

Символ $\perp_{\mathcal{SO}}$ означает только то, что моделирование $\mathcal{SO}$ не удалось. Если $\mathcal{A}$ ведёт себя неверно, но $\mathcal{SO}$ был смоделирован успешно, тогда возможно, что $\mathcal{SO}$ успешно смоделирует фальшивое выполнение `Sign`. Если $\mathcal{SO}$ был смоделирован успешно, это не приводит к $\perp_{\mathcal{SO}}$, но приводит к неверной подписи или сбою $\perp_{\mathtt{Sign}}$ из `Sign`.

A.2 Лемма реализации форка «обратной перемоткой» при успешном выполнении

В данном разделе нами рассматривается технология двойного форка и общая лемма реализации форка. Вспомните, что для доказательства невозможности подделки нашей схемы достаточно доказать, что фальсификатор $\mathcal{A}$ с значимым преимуществом в рамках экзистенциальной игры невозможности подделки может быть сокращён до некоторого $\mathcal{B}$, который создаст четыре подделки, как было описано выше. Как обычно, мы используем сокращения и метасокращения $\mathcal{A}$ и общую лемму реализации форка для выполнения нашей задачи. Мы обозначаем сокращение $\mathcal{A}$ со скрытым (black box) доступом к $\mathcal{A}$ как $(\mathcal{A}')^{\mathcal{A}}$, и мы обозначаем процесс принятия сокращений как $\mathcal{A} \rightsquigarrow \mathcal{A}'$. Наша стратегия доказательства примерно соответствует стратегии, описанной в работе [15], и может быть представлена следующим образом:

$$\mathcal{A} \rightsquigarrow \mathcal{A}' \rightsquigarrow \mathtt{fork}^{\mathcal{A}'} \rightsquigarrow \mathcal{A}'' \rightsquigarrow \mathtt{fork}^{\mathcal{A}''} \rightsquigarrow \mathcal{B}.$$

К слову, наша стратегия состоит в доказательстве того, что если существует фальсификатор $\mathcal{A}$, то есть и сокращение $\mathcal{A}'$ от $\mathcal{A}$, удовлетворяющее условие Леммы A.2.1, позволяющей реализовать форк. Алгоритм реализации форка может быть «обёрнут» в $\mathcal{A}''$, по которому так же может быть реализован форк. Если $\mathcal{A}'$ и $\mathcal{A}''$ имеют значимую вероятность принятия, то и $\mathtt{fork}^{\mathcal{A}''}$ тоже, сокращается нами до некоторого алгоритма решения дискретного логарифма $\mathcal{B}$.

Лемма A.2.1 (Общая лемма реализации форка)**.** Допустим, $q, \eta \geq 1$. Допустим, $\mathcal{P}$ является любым PPT алгоритмом, который в качестве входа берёт $\mathtt{inp}_{\mathcal{P}} = (\mathtt{inp}, \underline{h})$, где $\underline{h} = (h_1, \ldots, h_q)$ является последовательностью ответов на запросы оракула (η-битными строками), и возвращает в качестве выхода $\mathtt{out}_{\mathcal{P}}$ либо выделенный символ ошибки $\perp$, либо пару $(i, \mathtt{out})$, где $i \in [q]$ а `out` является некоторым выходом. Допустим, $\mathtt{acc}_{\mathcal{P}}$ обозначает вероятность, что $\mathcal{P}$ не выводитт $\perp$ (где эта вероятность берётся по всем случайным монетам $\mathcal{P}$, распределению `inp`, всем выбранным $\underline{h}$).

Есть алгоритм $\mathtt{fork}^{\mathcal{P}}$, который берёт в качестве входа некоторый $\mathtt{inp}_{\mathtt{fork}^{\mathcal{P}}} = \mathtt{inp}_{\mathcal{P}}$ и в качестве выхода $\mathtt{out}_{\mathtt{fork}^{\mathcal{P}}}$ производит либо выделенный символ ошибки $\perp$, либо пару, состоящую из пар $((i, \mathtt{out})$,

$(i', \mathtt{out}'))$, где $(i, \mathtt{out})$ и $(i', \mathtt{out}')$ являются выходами $\mathcal{P}$, при этом $i = i'$. Кроме того, допуская вероятность, что $\mathtt{fork}^{\mathcal{P}}$ связан снизу, мы получаем

$$\mathtt{acc}_{\mathtt{fork}^{\mathcal{P}}} \geq \mathtt{acc}_{\mathcal{P}} \left(\frac{\mathtt{acc}_{\mathcal{P}}}{q} - \frac{1}{2^{\eta}} \right).$$

Доказательство читатель может найти в работе [3].

Алгоритм $\mathtt{fork}^{\mathcal{P}}$: Допустим, $\eta > 1$ является параметром безопасности, $q > 1$ является полиномиальной функцией η. Допустим, $\mathcal{P}$ является любым PPT алгоритмом, соответствующим условиям Леммы A.2.1.

Алгоритм $\mathtt{fork}^{\mathcal{P}}$ в качестве выхода выдаёт выделенный символ ошибки $\mathtt{out}_{\mathtt{fork}^{\mathcal{P}}} = \perp$ или какой-то $\mathtt{out}_{\mathtt{fork}^{\mathcal{P}}} = (i', \mathtt{out}')$, где j является индексом в $[q]$, а $\mathtt{out}' = (\mathtt{out}, \mathtt{out}'', \mathtt{aux})$ таким образом, что $\mathtt{out}$, $\mathtt{out}''$ являются двумя выходами $\mathcal{P}$.

1. Берутся случайные монеты $\rho = \rho_{\mathcal{P}}$ для $\mathcal{P}$ и выбирается $\underline{h} \leftarrow (\{0,1\}^{\eta})^q$.
2. Выполняется $\mathtt{out}_{\mathcal{P}} \leftarrow \mathcal{P}(\mathtt{inp}_{\mathcal{P}}, \underline{h}; \rho)$.
3. Если $\mathtt{out}_{\mathcal{P}} = \perp_{\mathcal{P}}$, выводится $\perp_{\mathtt{fork}^{\mathcal{P}}}$ и процесс завершается. В противном случае $\mathtt{out}_{\mathcal{P}} = (i, \mathtt{out})$ для некоторых $\mathtt{out}$.
4. Берётся новый $\underline{h}' \leftarrow (\{0,1\}^{\eta})^q$.
5. Последовательности запросов оракула склеиваются вместе $\underline{h}^* := (h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_q)$.
6. Выполняется $\mathtt{out}''_{\mathcal{P}} \leftarrow \mathcal{P}(\mathtt{inp}_{\mathcal{P}}, \underline{h}^*; \rho)$.
7. Если $\mathtt{out}''_{\mathcal{P}} = \perp_{\mathcal{P}}$, выводится $\perp_{\mathtt{fork}^{\mathcal{P}}}$ и процесс завершается. В противном случае $\mathtt{out}''_{\mathcal{P}} = (i'', \mathtt{out}'')$.
8. Если $i \neq i''$ или $i = i''$, но $h_i = h'_i$, выводится $\perp_{\mathtt{fork}^{\mathcal{P}}}$, и процесс завершается.
9. В противном случае выбираются некоторые вспомогательные данные $\mathtt{aux}$, собирается $\mathtt{out}' = (\mathtt{out}, \mathtt{out}'', \mathtt{aux})$, выводится $(i, \mathtt{out}')$, и процесс завершается.

Если $\mathcal{P}$ в качестве входа берёт сразу несколько последовательностей запросов оракула, мы можем провести соответствующий разбор, чтобы включить их в $\mathtt{inp} \in \mathtt{inp}_{\mathcal{P}}$. Если $\mathtt{inp}_{\mathcal{P}}$ имеет некоторую другую последовательность $\underline{h}^*$ запросов оракула, объединённую подобным образом, то форк $\mathtt{fork}^{\mathcal{P}}$ может быть снова реализован в блокчейне. В следующем разделе мы описываем каждый алгоритм в цепочке сокращений, включающих нашу стратегию $\mathcal{A} \rightsquigarrow \mathcal{A}' \rightsquigarrow \mathtt{fork}^{\mathcal{A}'} \rightsquigarrow \mathcal{A}'' \rightsquigarrow \mathtt{fork}^{\mathcal{A}''} \rightsquigarrow \mathcal{B}$, доказывающую их существование и значимость вероятности их принятия по мере развития.

A.3 Сокращение $\mathcal{A}$

В данном разделе мы начинаем наш процесс реализации форка путём построения первого сокращения $\mathcal{A}'$ от $\mathcal{A}$ в цепочке сокращений. Мы объясняем, как $\mathcal{A}'$ моделирует каждого оракула, и даём лемму в отношении вероятности принятия $\mathcal{A}'$.

Мы производим сокращение $\mathcal{A}'$ от $\mathcal{A}$подобно тому, как это происходит в случае с Леммой A.2.1, которая применяется к алгоритму $\mathcal{P}$, который берёт в качестве входа некоторый $\mathtt{inp}_{\mathcal{P}} = (\mathtt{inp}, \underline{h})$. $\mathcal{A}'$ берёт в качестве входа $\mathtt{inp}_{\mathcal{A}'} = (\mathtt{inp}, \underline{h}_{\mathrm{sig}})$. Тем не менее $\mathcal{A}'$ должен смоделировать оба запроса $\mathcal{H}_{\mathrm{sig}}$ и $\mathcal{H}_{\mathrm{agg}}$, поэтому мы определяем $\mathtt{inp} := (\mathtt{inp}_{\mathcal{A}}, \underline{h}_{\mathrm{agg}})$. Другими словами, $\mathcal{A}'$ берёт в качестве входа $\mathtt{inp}_{\mathcal{A}'} = (\mathtt{inp}_{\mathcal{P}}, \underline{h}_{\mathrm{agg}}) = \big((X_h, \underline{h}_{\mathrm{agg}}), \underline{h}_{\mathrm{sig}}\big)$.

$\mathcal{A}'$ отвечает на запросы оракула, отправляя $\underline{h}_{\text{agg}}$ и $\underline{h}_{\text{sig}}$, как описано ниже. $\mathcal{A}'$ дополняет выход $\mathcal{A}$, используя в качестве выхода $\perp$, если $\mathcal{A}$ выдаст ошибку. С другой стороны, если выход $\mathcal{A}$ будет не $\perp$, допустим, $\texttt{forg} = \left(\mathfrak{m}^*, \sigma, \left\{\underline{X}^{(\ell)}\right\}_\ell\right)$, $\mathcal{A}'$ воспроизведёт это в своём выходе, но дополнит некоторой информацией транскрипта, $\texttt{out}_{\mathcal{A}'} = (i_{\text{sig}}, \texttt{out}^*)$, где $\texttt{out}^* = (\texttt{forg}, i_{\text{sig}}, h_{\text{sig},i_{\text{sig}}}, \ell_{\text{sig}}, \pi_{\text{sig}}, i_{\text{agg}}, h_{\text{agg},i_{\text{agg}}}, \underline{a})$ для некоторой подписи и накопления индексов $i_{\text{sig}}, i_{\text{agg}}$ и ответов $h_{\text{sig},i_{\text{sig}}}, h_{\text{agg},i_{\text{agg}}}$, некоторых индексов кольца $\ell_{\text{sig}}, \pi_{\text{sig}}$ и списка коэффициентов $\underline{a}$ выделенных из транскрипта следующим образом:

1. $(i_{\text{sig}}, \ell_{\text{sig}})$ определяются таким образом, что ответ на первый запрос, обращённый $\mathcal{A}$ к $\mathcal{H}_{\text{sig}}$ для решения любого уравнения верификации, используемого при фальсификации, будет i_{sig}^{th} запросом и займёт место для ℓ_{sig}^{th} члена кольца, то есть $h_{\text{sig},i_{\text{sig}}} = c_{\ell_{\text{sig}}+1} = \mathcal{H}_{\text{sig}}(M, P_{\ell_{\text{sig}}}, L_{\ell_{\text{sig}}}, R_{\ell_{\text{sig}}})$ для некоторого индекса ℓ.

2. π_{sig} определяется таким образом, что ответ на окончательный запрос, сделанный $\mathcal{A}$ к $\mathcal{H}_{\text{sig}}$ для решения любого уравнения верификации, используемого при фальсификации, будет π_{sig}^{th} задачей подписи, то есть окончательной задачей подписи, которая должна быть решена в транскрипте, будет $c_{\pi_{\text{sig}}} = \mathcal{H}_{\text{sig}}(M, P_{\pi_{\text{sig}}-1}, L_{\pi_{\text{sig}}-1}, R_{\pi_{\text{sig}}-1})$.

3. i_{agg} является индексом первого запроса накопления, сделанного любым членом $\underline{X}^{(\ell_{\text{sig}})}$, то есть коэффициентом накопления по X_h в $\underline{X}^{(\ell)}$ будет $h_{\text{agg},i_{\text{agg}}}$.

4. $\underline{a}$ содержит коэффициенты накопления всех выбранных со злым умыслом ключей в $\underline{X}^{(\ell_{\text{sig}})}$ (упорядоченных некоторым установившимся образом).

Лемма А.3.1. Допустим, $\mathcal{A}$ делает самое большее q запросов случайного оракула и не выводит $\perp_{\mathcal{A}}$. Каждый запрос, обращённый к $\mathcal{H}_{\text{sig}}$ для решения уравнений верификации, делается $\mathcal{A}$ до завершения процесса, за исключением вероятности, заданной выше $1 - (1 - (\mathfrak{p} - q)^{-1})^r$ (обусловленной тем, что $\mathcal{A}$ не выдаст $\perp_{\mathcal{A}}$).

Доказательство. В случае если $\mathcal{A}$ не запросит у $\mathcal{H}_{\text{sig}}$ верификации, запрос потребует, чтобы $\mathcal{A}$ угадал выход $\mathcal{H}_{\text{sig}}$ для некоторого запроса случайным образом. Это может произойти с самой большой вероятностью $(\mathfrak{p} - q)^{-1}$ (результат должен исключать уже сделанные запросы). Следовательно, вероятность успеха в этом случае даже по одному для любого из запросов верификации r составит $1 - (1 - (\mathfrak{p} - q)^{-1})^r$. Это верхний предел вероятности, что $\mathcal{A}$ сделает это случайным образом, а не запрашивая решения уравнения верификации. □

Следствием этой леммы является то, что $\mathcal{A}'$ может без каких-либо затруднений найти i_{sig}, $h_{\text{sig},i_{\text{sig}}}$, ℓ_{sig} и π_{sig}.

Лемма А.3.2. Допустим, $\mathcal{A}$ делает самое большее q запросов случайного оракула и не выводит $\perp$. Каждый запрос, обращённый к $\mathcal{H}_{\text{agg}}$ для получения каждого коэффициента накопления по X_h, делается $\mathcal{A}$ до завершения процесса, за исключением вероятности, заданной выше $1 - (1 - (\mathfrak{p} - q)^{-1})^{nr}$ (обусловленной тем, что $\mathcal{A}$ не выдаст $\perp_{\mathcal{A}}$).

Доказательство. В случае если $\mathcal{A}$ не сделает один этих запросов, необходимо, чтобы $\mathcal{A}$ угадал выход $\mathcal{H}_{\text{agg}}$. При наличии r членов кольца, каждый из которых имеет дополняющие ключи $\underline{X}^{(\ell)}$ таким образом, что $\left|\underline{X}^{(\ell)}\right| \leq n$ имеет самое большее nr запросов с вероятностью успешного угадывания $(\mathfrak{p} - q)^{-1}$. Следовательно, вероятность успеха в этом случае, даже в случае nr попыток, составит $1 - (1 - (\mathfrak{p} - q)^{-1})^{nr}$. □

Следствием является то, что $\mathcal{A}'$ может без каких-либо затруднений выделить i_{agg} и $h_{\text{agg},i_{\text{agg}}}$.

Лемма А.3.3. Допустим, $\mathcal{A}$ является фальсификатором (t, ϵ, n, q). Допустим, E является случаем, когда $\mathcal{A}$ не выводит $\perp$. В случае E для каждой $1 \leq \ell \leq r$, запрос, направленный $\mathcal{H}_{\text{sig}}$ для решения задачи решения уравнения верификации $c_{\ell+1}$, делается после запроса, направленного $\mathcal{H}_{\text{agg}}$ для получения коэффициента накопления по X_h в соответствующем $\underline{X}^{(\ell)}$, за исключением вероятности, заданной выше $1 - (1 - (\mathfrak{p} - q)^{-1})^r$.

Доказательство. Так как $\Phi(\underline{X}^{(\ell)}) = P_\ell$ является частью предварительного образа, для $c_{\ell+1}$, вероятность того, что $\mathcal{A}$ сможет вычислить $c_{\ell+1}$ до запроса вычисления P_ℓ (если допустить, что все запросы, кроме этого окончательного, были сделаны), составляет самое большее $(\mathfrak{p} - q)^{-1}$, и существует самое большее r таких запросов, связанных с фальшивкой. □

Эти леммы и следствия демонстрируют, что все запросы верификации появляются, и каждое из них появляется после того, как будут вычислены коэффициенты накопления. Имея данные леммы, мы можем описать $\mathcal{A}'$. $\mathcal{A}'$ помещает $\mathcal{A}$ в «чёрный ящик», моделируя все запросы оракулов, которые делает $\mathcal{A}$. $\mathcal{A}'$ ведёт внутренние таблицы, обозначенные как $\mathbb{T}_{\text{sig}}$, $\mathbb{T}_{\texttt{agg}}$, и счётчик, обозначенный как $\texttt{ctr}$, чтобы отслеживать запросы, которые делаются с целью поддержания внутренней согласованности, а также, чтобы отслеживать индексы запросов оракула.

Алгоритм $\mathcal{A}'$: В качестве входа $\mathcal{A}'$ берёт $\texttt{inp}_{\mathcal{A}'} = ((X_h, \underline{h}_{\text{agg}}), \underline{h}_{\text{sig}})$. $\mathcal{A}'$ имеет доступ к «чёрному ящику» с $\mathcal{A}$, моделирует запросы, как описано ниже, и в качестве выхода выдаёт либо выделенный символ ошибки $\perp$, либо некоторый $(\underline{i}, \texttt{out})$.

1. $\mathcal{A}'$ выбирает случайные монеты $\rho = \rho_{\mathcal{A}}$, устанавливает $\texttt{ctr} := 0$ и задаёт $\texttt{inp}_{\mathcal{A}} := \{X_h\}$.
2. $\mathcal{A}'$ выполняет $\mathcal{A}(\texttt{inp}_{\mathcal{A}}; \rho_{\mathcal{A}})$, отвечая на запросы оракула, которые делает $\mathcal{A}$, как описано ниже.
3. Если $\mathcal{A}$ выводит $\perp_{\mathcal{A}}$, $\mathcal{A}'$ выводит $\perp_{\mathcal{A}'}$ и завершает процедуру.
4. В противном случае $\mathcal{A}$ выводит поделку $\texttt{forg} = (\mathfrak{m}^*, \sigma, \left\{\underline{X}^{(\ell)}\right\}_\ell)$.
5. $\mathcal{A}'$ находит все запросы верификации в транскрипте и находит следующее:
   (a) индекс запроса i_{sig} первого уравнения верификации, которое использовалось при фальсификации;
   (b) ответ $h_{\text{sig},i_{\text{sig}}}$ на этот первый запрос верификации;
   (c) индекс кольца ℓ_{sig} для входа этого первого запроса верификации (соответствующий уравнению верификации $c_{\ell_{\text{sig}}+1} = \mathcal{H}(M, P_{\ell_{\text{sig}}}, L_{\ell_{\text{sig}}}, R_{\ell_{\text{sig}}}) = h_{\text{sig},i_{\text{sig}}}$);
   (d) ответ $h_{\text{sig},i'_{\text{sig}}}$ на окончательный запрос верификации;
   (e) индекс кольца $\pi_{\text{sig}} - 1$ для входа этого запроса, соответствующий схожему уравнению верификации $c_{\pi_{\text{sig}}} = \mathcal{H}(M, P_{\pi_{\text{sig}}-1}, L_{\pi_{\text{sig}}-1}, R_{\pi_{\text{sig}}-1})$;
   (f) индекс запроса i_{agg} первого запроса накопления для любого члена $\underline{X}^{(\ell_{\text{sig}})}$;
   (g) ответ $h_{\text{agg},i_{\text{agg}}}$ на этот запрос; и, наконец,
   (h) $\mathcal{A}'$ находит все остальные коэффициенты накопления для членов $\underline{X}^{(\ell_{\text{sig}})}$ в $\mathbb{T}_{\texttt{agg}}$, скажем, $\underline{a}$. Смотрите описание оракула $\mathcal{H}_{\text{agg}}$ для получения дополнительной информации.
6. $\mathcal{A}'$ выводит $\texttt{out}_{\mathcal{A}'} = (i_{\text{sig}}, (\texttt{forg}, i_{\text{sig}}, h_{\text{sig},i_{\text{sig}}}, \ell_{\text{sig}}, \pi_{\text{sig}}, i_{\text{agg}}, h_{\text{agg},i_{\text{agg}}}, \underline{a}))$.

Моделирование $\mathcal{H}_{\texttt{com}}$: Чтобы смоделировать запросы в форме $\mathcal{H}_{\text{com}}(\texttt{inp})$, $\mathcal{A}'$ отслеживает внутреннюю таблицу $\mathbb{T}_{\text{com}}$. $\mathcal{A}'$ проверяет, пуста или нет $\mathbb{T}_{\text{com}}[\texttt{inp}]$. Если пуста, выбирается случайный $\texttt{out} \xleftarrow{\$} \{0,1\}^\eta$ и сохраняет $\mathbb{T}_{\text{com}}[\texttt{inp}] \leftarrow \texttt{out}$. В противном случае $\mathbb{T}_{\text{com}}[\texttt{inp}]$ отправляется $\mathcal{A}$.

Моделирование $\mathcal{H}_{\mathtt{ki}}$: Чтобы смоделировать запросы, которые делает $\mathcal{A}$ в форме $\mathcal{H}_{\mathrm{ki}}(\mathtt{inp})$, $\mathcal{A}'$ отслеживает внутреннюю таблицу $\mathbb{T}_{\mathrm{ki}}$. $\mathcal{A}'$ проверяет, пуста или нет $\mathbb{T}_{\mathrm{ki}}[\mathtt{inp}]$. Если пуста, выбирается случайная точка $Y' \in \mathbb{G}$ и сохраняется $\mathbb{T}_{\mathrm{ki}}[\mathtt{inp}] \leftarrow Y'$. В противном случае $\mathbb{T}_{\mathrm{ki}}[\mathtt{inp}]$ отправляется $\mathcal{A}$.

Моделирование $\mathcal{H}_{\mathtt{agg}}$: $\mathcal{A}'$ тщательно отслеживает запросы накопления и всегда гарантирует, что коэффициент накопления для X_h будет выбран после всех остальных коэффициентов.

1. $\mathcal{A}$ запрашивает $\mathcal{H}_{\mathrm{agg}}$, отправляя $\mathtt{inp}$.
2. $\mathcal{A}'$ проверяет, определена или нет $\mathbb{T}_{\mathrm{agg}}[\mathtt{inp}]$. Если не определена, $\mathcal{A}$ делает следующее:
   (a) $\mathcal{A}'$ проверяет, можно ли разобрать $\mathtt{inp}$ как некоторое $(Y, \underline{X})$. Если нет, $\mathcal{A}'$ берёт случайные данные $\mathbb{T}_{\mathtt{agg}}[\mathtt{inp}] \xleftarrow{\$} \mathbb{Z}_{\mathtt{p}}$;
   (b) в противном случае $\mathcal{A}'$ разбирает $\mathtt{inp} = (Y, \underline{X})$ и проверяет, чтобы $X_h \in \underline{X}$. Если это не так, то для каждого $Y' \in \underline{X}$, $\mathcal{A}'$ берёт случайные данные $\mathbb{T}_{\mathtt{agg}}[Y', \underline{X}] \xleftarrow{\$} \mathbb{Z}_{\mathtt{p}}$;
   (c) в противном случае $\mathtt{inp} = (Y, \underline{X})$ для некоторого $Y \in \underline{X}$ так, чтобы $X_h \in \underline{X}$, но $\mathbb{T}_{\mathtt{agg}}[\mathtt{inp}]$ пока остаётся неопределённой. $\mathcal{A}'$ проверяет, является ли $\mathbb{T}_{\mathtt{agg}}[X_h, \underline{X}]$ определённой. Если да, $\mathcal{A}'$ выводит $\perp_{\mathrm{agg}}$ и завершает процесс;
   (d) в противном случае для каждого $Y' \in \underline{X} \setminus \{X_h\}$ так, чтобы $\mathbb{T}_{\mathrm{agg}}[Y', \underline{X}]$ была неопределённой, $\mathcal{A}'$ выбирает случайные данные $\mathbb{T}_{\mathtt{agg}}[Y', \underline{X}] \xleftarrow{\$} \mathbb{Z}_{\mathtt{p}}$;
   (e) после того как все данные $\underline{X}$, кроме X_h, будут введены в $\mathbb{T}_{\mathrm{agg}}$, $\mathcal{A}'$ увеличит значение $\mathtt{ctr}_{\mathtt{agg}}$ и сохранит $\mathbb{T}_{\mathtt{agg}}[X_h, \underline{X}] \leftarrow h_{\mathrm{agg},\mathtt{ctr}_{\mathtt{agg}}}$.
3. $\mathcal{A}'$ выводит $\mathbb{T}_{\mathrm{agg}}[\mathtt{inp}]$.

Следует отметить, что $\mathcal{A}'$ выдаёт $\perp_{\mathrm{agg}}$ только в том случае, если $\mathbb{T}_{\mathrm{agg}}[X_h, \underline{X}]$ определена, но некоторое $Y \in \underline{X}$ имеет определённую $\mathbb{T}_{\mathrm{agg}}[Y, \underline{X}]$. Если $\mathcal{A}'$ следует протоколу, этого никогда не произойдёт. Во всех остальных случаях коэффициент накопления для X_h определяется после того, как будут определены все остальные коэффициенты. Это важно для нашего доказательства невозможности подделки.

Моделирование $\mathcal{H}_{\mathtt{sig}}$: Чтобы смоделировать запросы, которые делает $\mathcal{A}$ для подписания $\mathcal{H}_{\mathrm{sig}}(\mathtt{inp})$, $\mathcal{A}'$ проверяет, определена или нет $\mathbb{T}_{\mathtt{sig}}[\mathtt{inp}]$. Если нет, $\mathcal{A}'$ проверяет, можно ли разобрать $\mathtt{inp}$ как $\mathtt{inp} = (M, P, U, V)$ для некоторого M и групповых точек P, U, V. Если нет, случайным образом выбирается $\mathbb{T}_{\mathtt{sig}}[\mathtt{inp}] \xleftarrow{\$} \{0,1\}^{\eta}$. В противном случае $\mathtt{ctr}_{\mathtt{sig}}$ увеличивается и сохраняется $\mathbb{T}_{\mathtt{sig}}[M, P, U, V] \leftarrow h_{\mathrm{sig},\mathtt{ctr}_{\mathtt{sig}}}$. Так или иначе, $\mathcal{A}'$ отправляет $\mathbb{T}_{\mathtt{sig}}[\mathtt{inp}]$ для $\mathcal{A}$.

Моделирование $\mathcal{SO}$: Для простоты обозначения $\mathcal{SO}$ использует обозначения, предполагающие, что это первый член объединения, начинающий процесс подписания (с индексом $j = 1$ в объединении). $\mathcal{A}'$ моделирует $\mathcal{SO}$ следующим образом:

1. $\mathcal{A}$ запрашивает $\mathcal{SO}$, используя $\mathtt{inp}_{\mathcal{SO}}$.
2. После получения запроса $\mathcal{A}'$ разбивает $\mathtt{inp}$ как $\left(M, \underline{P}, \pi, \left\{\underline{X}^{(\ell)}\right\}_{1 \le \ell \le r}\right)$ так, чтобы $\left|\underline{X}^{(\ell)}\right| \ge 2$ для некоторого ℓ и для каждого ℓ, $\left|\underline{X}^{(\ell)}\right| \le n$, $X_h \in \underline{X}^{(\ell)}$ и $P_\ell \in \underline{P}$ выводились на основе соответствующего $\underline{X}^{(\ell)}$. Если $\mathcal{A}'$ не может произвести такое разбиение, $\mathcal{A}'$ отправляет $\perp_{\mathcal{SO}}$ $\mathcal{A}$, чтобы обозначить, что запрос отклонён, и останавливает моделирование $\mathcal{SO}$.
3. В противном случае $\mathcal{A}'$ выбирает данные подписания $\mathtt{dat}_1$, как это делалось на этапе 2, описанном в разделе A.1, вычисляет обязательство $\mathtt{com}_1$ и отправляет $\mathtt{com}_1$ $\mathcal{A}$ следующим образом:

(a) $\mathcal{A}'$ увеличивает значение $\mathtt{ctr}$ и берёт критическое обязательство $c_\pi \leftarrow h_{\mathrm{sig},\mathtt{ctr}_{\mathtt{sig}}}$.

(b) $\mathcal{A}'$ выбирает случайный набор данных подписания $\{s_{1,\ell}\}_{\ell\in[r]}$ (включая индекс $\ell = \pi$).

(c) $\mathcal{A}'$ вычисляет $L_{1,\pi} = s_{1,\pi}G + c_\pi P_\pi$ и $R_{1,\pi} = s_{1,\pi}H_\pi + c_\pi J$, собирает данные подписи $\mathtt{dat}_1 := (L_{1,\pi}, R_{1,\pi}, \{s_{1,\ell}\}_{\ell\in[r],\ell\neq\pi})$ и выбирает случайное $\mathtt{com}_1 \xleftarrow{\$} \{0,1\}^\eta$.

(d) если $\mathbb{T}_{\mathtt{com}}[\mathtt{dat}_1]$ определена, $\mathcal{A}'$ останавливает моделирование $\mathcal{SO}$, выводит $\perp_1$ чтобы указать, что некоторые $\mathtt{dat}_1$ уже были использованы, и завершает процедуру;

(e) в противном случае, $\mathcal{A}'$ моделирует запрос $\mathcal{H}_{\mathrm{com}}$ устанавливая $\mathbb{T}_{\mathtt{com}}[\mathtt{dat}_1] \leftarrow \mathtt{com}_1$, отправляет $\mathtt{com}_1$ to $\mathcal{A}$.

4. После того как $\mathcal{A}$ ответит обязательствами $\underline{\mathtt{com}} = \{\mathtt{com}_j\}_{j\in[n],j\neq1}$, $\mathcal{A}'$ произведёт некоторую «распаковку», после чего отправит $\mathtt{dat}_1$ $\mathcal{A}$.

(a) Для каждого $j > 1$, $\mathcal{A}'$ ищет $\mathbb{T}_{\mathtt{com}}$ на предмет наличия $\mathtt{dat}$ так, чтобы $\mathbb{T}_{\mathtt{com}}[\mathtt{dat}] = \mathtt{com}_j$.

(b) Если для какого-либо j найдено более одного $\mathtt{dat}$, $\mathcal{A}'$ останавливает моделирование $\mathcal{SO}$, выводит $\perp_2$ и завершает процедуру.

(c) Если для какого-либо j не было найдено никаких $\mathtt{dat}$, $\mathcal{A}'$ задаёт $\mathtt{alert}_1 \leftarrow \mathtt{true}$.

(d) В противном случае для каждого $\mathtt{dat} = \mathtt{dat}_j$ в $\mathbb{T}_{\mathtt{com}}$ обнаруживаются ровно одни $\mathtt{com}_j$. Если какие-либо $\mathtt{dat}_j$ не могут быть разбиты как $(U_j, V_j, \{s_{j,\ell}\}_{\ell\neq\pi})$, $\mathcal{A}'$ задаёт $\mathtt{alert}_2 \leftarrow \mathtt{true}$.

(e) В противном случае для каждого $\mathtt{dat}_j$ в $\mathbb{T}_{\mathtt{com}}$ обнаруживаются ровно одни $\mathtt{com}_j$, и они могут быть разбиты на последовательность $(U_j, V_j, \{s_{j,\ell}\})$. После разбиения каждых $\mathtt{dat}_j$, $\mathcal{A}'$ выполняет следующее:

i. $\mathcal{A}'$ вычисляет $U = \sum_j U_j$, $V = \sum_j V_j$ и $s_\ell = \sum_j s_{j,\ell}$ для каждого $\ell \neq \pi$;

ii. $\mathcal{A}'$ проверяет, есть ли последовательность (M, P_ℓ, U, V) в $\mathbb{T}_{\mathtt{sig}}$. Если есть, $\mathcal{A}'$ останавливает моделирование $\mathcal{SO}$, выводит $\perp_3$ и завершает процедуру;

iii. в противном случае, $\mathcal{A}'$ увеличивает значение $\mathtt{ctr}_{\mathtt{sig}}$, задаёт $c_{\pi+1} \leftarrow h_{\mathrm{sig},\mathtt{ctr}_{\mathtt{sig}}}$ и сохраняет $\mathbb{T}_{\mathtt{sig}}[M, P_\ell, U, V] \leftarrow c_{\pi+1}$;

iv. для каждого $\ell = \pi+1, \pi+2, \ldots, \pi-2$, $\mathcal{A}'$ вычисляет $L_\ell = s_\ell G + c_\ell P_\ell$ и $R_\ell = s_\ell H_\ell + c_\ell J$ и проверяет есть ли последовательность $(M, P_\ell, L_\ell, R_\ell)$ в $\mathbb{T}_{\mathtt{sig}}$. Если есть, $\mathcal{A}'$ останавливает моделирование $\mathcal{SO}$, выводит $\perp_3$ и завершает процедуру;

v. в противном случае, $\mathcal{A}'$ увеличивает значение $\mathtt{ctr}_{\mathtt{sig}}$, задаёт $c_{\ell+1} \leftarrow h_{\mathrm{sig},\mathtt{ctr}_{\mathtt{sig}}}$ и сохраняет

$$\mathbb{T}_{\mathtt{sig}}[M, P_\ell, L_\ell, R_\ell] \leftarrow c_{\ell+1}$$

а затем переходит к следующему ℓ;

vi. $\mathcal{A}'$ проверяет, является ли $\mathbb{T}_{\mathtt{sig}}[M, P_{\pi-1}, L_{\pi-1}, R_{\pi-1}]$ пустой. Если нет, $\mathcal{A}'$ останавливает моделирование $\mathcal{SO}$, выводит $\perp_3$ и завершает процедуру;

vii. в противном случае, $\mathbb{T}_{\mathtt{sig}}[M, s_{\pi-1}G + c_{\pi-1}P_{\pi-1}, s_{\pi-1}H_{\pi-1} + c_{\pi-1}J] \leftarrow c_\pi$.

(f) $\mathcal{A}'$ отправляет $\mathtt{dat}_1$ $\mathcal{A}$.

5. После того как $\mathcal{A}$ ответит, отправив $\underline{\mathtt{dat}}' = \{\mathtt{dat}'_j\}_{j\neq1}$, $\mathcal{A}'$ проверит $\mathtt{com}_j = \mathbb{T}_{\mathtt{com}}[\mathtt{dat}'_j]$ для каждого j. Если есть какое-либо несоответствие, $\mathcal{A}'$ отправляет $\perp_{\mathcal{SO}}$ $\mathcal{A}$, чтобы указать на успешное моделирование неудавшегося процесса подписания.

6. В противном случае, если $\mathtt{alert}_1 = \mathtt{true}$ или $\mathtt{alert}_2 = \mathtt{true}$, $\mathcal{A}'$ выдаёт $\perp_4$ и завершает процедуру.

7. В противном случае, $\mathcal{A}'$ отправляет $s_{1,\pi}$ $\mathcal{A}$ что для $\mathcal{A}$ является достаточной информацией для того, чтобы вычислить оставшуюся часть подписи.

Если выход является одним из символов из $\{\perp_1, \perp_2, \perp_3, \perp_4\}$, то $\mathcal{A}'$ фактически завершает процесс: это символы ошибки, которые указывают на то, что происходит что-то странное с порядком присвоения значений оракулом при моделировании $\mathcal{SO}$. Это указывает на ошибку подписывающего оракула из-за неправильно сформированного запроса или какого-либо другого неверного порядка событий. Мы докажем, что такая вероятность ничтожна.

Более того, $\perp_{\mathcal{SO}}$ появляется только в том случае, если $\mathcal{A}$ запрашивает $\mathcal{SO}$, используя что-то, что выходит за рамки игры доказательства невозможности подделки, описанной в определении 5.1.1, или же отправляет обязательства, которые не открылись надлежащим образом. $\mathcal{A}'$ не заканчивает процесс, поскольку это удачные случаи моделирования ошибки процесса подписания, а не ошибка моделирования.

Нами исследуется вероятность приемлемости для $\mathcal{A}'$.

Лемма A.3.4. Допустим $\mathcal{A}$ является фальсификатором (t, ϵ, q, n), имеющим доступ к $\mathcal{SO}$ и $\underline{\mathcal{H}}$, и допустим, что $\mathcal{A}'$ является каким-либо сокращением $\mathcal{A}$, моделирующим запросы оракула, как это было описано выше. Допустим $t' > 0$, и, допустим, $c > 0$ является тем временем, что необходимо $\mathcal{A}'$ для того, чтобы случайным образом выбрать новый элемент $\mathbb{Z}_{\mathfrak{p}}$ или $\mathbb{G}$. Тогда за время, составляющее самое большее $t + t'$ и с самой большей вероятностью $\epsilon' = \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4$, $\mathcal{A}'$ завершит процесс, не выводя какой-либо $\perp_i \in \{\perp_1, \perp_2, \perp_3, \perp_4, \perp_{\text{agg}}\}$, где

$$\epsilon_1 = 1 - \prod_{k \in [q-1]} (1 - k\mathfrak{p}^{-r-1}) \qquad \epsilon_2 = 1 - \prod_{k \in [q-1]} (1 - k\mathfrak{p}^{-1})$$

$$\epsilon_3 = 1 - \prod_{k \in [q-1]} (1 - k\mathfrak{p}^{-4}) \qquad \epsilon_4 = 1 - \exp(-\frac{-t'(t'-c)}{2(\mathfrak{p}-q)c^2}).$$

Доказательство. Символы ошибки разбиваются, если $\mathcal{A}'$ выдаёт некоторый $\perp$ и завершает процесс, поэтому вероятность появления какого-либо символа $\perp$ равна сумме каждого из них, взятого отдельно. Другими словами, если E является случаем, когда $\mathcal{A}'$ выдаёт некоторый $\perp$ и завершает процесс, а E_i является случаем, когда $\mathcal{A}'$ выдаёт $\perp_i$ для некоторого $i \in \{1, 2, 3, 4, \text{agg}\}$ и завершает процесс, то мы получаем $\epsilon_i = \mathbb{P}[E_i]$ и по закону полной вероятности $\mathbb{P}[E] = \mathbb{P}[E \mid E_{\text{agg}}]\mathbb{P}[E_{\text{agg}}] + \sum_{i \in [4]} \mathbb{P}[E \mid E_i]\mathbb{P}[E_i]$.

Более того, каждая условная вероятность в данном случае, очевидно, будет равна 1 (вероятность того, что $\mathcal{A}'$ некоторый $\perp_i$, учитывая тот факт, что $\mathcal{A}'$ выводит $\perp_3$, равна 1, например), поэтому мы имеем простую сумму. Следовательно, ограничение вероятности приемлемости, описанное ниже, эквивалентно ограничению каждой вероятности каждого из этих случаев ошибки, описанных выше. Мы упрощаем анализ: $\mathbb{P}[E_{\text{agg}}] = 0$ по структуре.

Ограничение ϵ_1: Если $\mathcal{A}'$ при моделировании запроса $\mathcal{SO}$ для $\mathcal{A}$ выбирает $\mathtt{dat}_1$ таким образом, что $\mathbb{T}_{\mathtt{com}}[\mathtt{dat}_1]$ не будет пустой, то $\perp_1$ будет выходом. Следовательно, этот символ появляется, только если $\mathcal{A}'$ выберет случайные данные подписания $\mathtt{dat}_1$, для которых уже был определён образ по $\mathcal{H}_{\text{com}}$. $\mathcal{A}'$ никогда не выбирает одни и те же случайные данные дважды в соответствии со спецификацией моделирования $\mathcal{SO}$, а это подразумевает, что $\mathcal{A}$ запросил $\mathcal{H}_{\text{com}}$, отправив $\mathtt{dat}_1$ в какой-то момент в прошлом, и, более того, $\mathcal{A}'$ случайно раскрыл свой предварительный образ. Существует $\mathfrak{p}^{r+1}$ выборов $(U_\ell, V_\ell, \{s_\ell\})$; этот сценарий является точным сценарием атаки «дня рождения». Если предположить, что при самом большем q такие выборы будут использоваться в $\mathcal{H}_{\text{com}}$, вероятность того, что $\mathcal{A}'$ не увидит конфликтов, точно составляет $\prod_{k \in [q-1]}(1 - k\mathfrak{p}^{-r-1})$. Следовательно, мы получаем

$$\epsilon_1 \leq 1 - \prod_{k \in [q-1]} (1 - k\mathfrak{p}^{-r-1}).$$

Ограничение ϵ_2: Символ ошибки $\perp_2$ появляется только в том случае, если в таблице $(U, V, \{s_\ell\})$ будет найдено по крайней мере два $\mathbb{T}_{\mathtt{com}}[U, V, \{s_\ell\}] = \mathtt{com}_j$ для одного и того же $\mathtt{com}_j$. Это подразумевает конфликт смоделированного случайного оракула. Мы получаем

$$\epsilon_2 \leq 1 - \prod_{k \in [q-1]} (1 - k\mathfrak{p}^{-1}).$$

Ограничение ϵ_3: Допустим, E является случаем, когда $\mathcal{A}'$ моделирует запрос для $\mathcal{SO}$, проверяет $\mathbb{T}_{\mathtt{sig}}$ и обнаруживает, что некоторый запрос в форме $(M, P_\ell, U_\ell, V_\ell)$ уже был сделан, и выдаёт $\perp_3$. Это также представляет собой сценарий атаки «дня рождения»: так как M является выходом $\mathcal{H}_{\mathrm{msg}}$, мы имеем $\mathfrak{p}^4$ таких возможных выборов, и мы делаем до q запросов, таким образом, вероятность двух конфликтов составляет точно $\prod_{k \in [q-1]}(1 - k\mathfrak{p}^{-4})$.

Ограничение ϵ_4: Символ ошибки $\perp_4$ появляется только в том случае, если $\mathtt{alert}_1 = \mathtt{true}$ и моделирование практически подошло к концу. $\mathcal{A}'$ добирается до этой точки только в том случае, если $\mathcal{A}$ ведёт себя неправильно и отправляет обязательство $\mathtt{com}_j$ на этапе обязательства и раскрытия, который не был связан ещё ни с одним запросом в форме $\mathcal{H}_{\mathrm{com}}$ и всё же создал данные открытия $\{\mathtt{dat}_j\}_j$, которые переходят на фазу раскрытия: $\mathcal{A}$ угадал $\mathtt{com}_j = \mathcal{H}_{\mathrm{msg}}(\mathrm{dat}_j)$, не запрашивая $\mathcal{H}_{\mathrm{msg}}$. Это также сценарий атаки «дня рождения». Вероятно, что злоумышленнику, проводящему атаку, понадобится более k попыток до того, как он увидит, что первый конфликт ограничен сверху при помощи $\exp(\frac{-k(k-1)}{2(\mathfrak{p}-q)})$. Если допустить, что каждая попытка занимает постоянное количество времени (скажем, c единиц времени на попытку), так как $\mathcal{A}'$ даётся $t' > 0$ времени в добавок к времени выполнения $\mathcal{A}$, вероятность того, что $\mathcal{A}'$ выдаст $\perp_4$ составляет самое большее $1 - \exp(-\frac{-t'(t'-c)}{2(\mathfrak{p}-q)c^2})$.

□

Следует отметить, что путём перемасштабирования времени $(t', t) \mapsto (\frac{t'}{c}, \frac{t}{c})$ мы можем переписать $\epsilon_4 = 1 - e^{-t'(t'-1)/(\alpha(\mathfrak{p}-q))}$ для некоторого $\alpha > 0$. Также следует отметить, что каждое ϵ_i является незначительным в $\mathfrak{p}$, поэтому и их сумма будет незначительной в $\mathfrak{p}$. Так как $\mathcal{A}'$ сравним с гипотезой, изложенной в лемме A.2.1, мы получаем непосредственно следующее.

Следствие A.3.5. Алгоритм $\mathtt{fork}^{\mathcal{A}'}$ и $\mathtt{fork}^{\mathcal{A}''}$ имеют вероятность принятия, ограниченную снизу:

$$\mathtt{acc}_{\mathtt{fork}^{\mathcal{A}'}} \geq \mathtt{acc}_{\mathcal{A}'} \left(\frac{\mathtt{acc}_{\mathcal{A}'}}{q} - \frac{1}{2^\eta} \right)$$
$$\mathtt{acc}_{\mathtt{fork}^{\mathcal{A}''}} \geq \mathtt{acc}_{\mathtt{fork}^{\mathcal{A}'}} \left(\frac{\mathtt{acc}_{\mathtt{fork}^{\mathcal{A}'}}}{q} - \frac{1}{2^\eta} \right)$$

В следующем разделе мы описываем $\mathtt{fork}^{\mathcal{A}'}$ и $\mathtt{fork}^{\mathcal{A}''}$.

A.4 Двойная реализация форка

В данном разделе мы дважды применим общую лемму реализации форка и достигнем кульминации - решения дискретного логарифма. Мы реализуем форк $\mathcal{A}'$ в два этапа. Сначала мы строим $\mathtt{fork}^{\mathcal{A}'}$, чтобы взять в качестве входа некоторый $\mathtt{inp}_{\mathtt{fork}^{\mathcal{A}'}} = (X_h, \underline{h}_{\mathrm{agg}})$, случайным образом выбрать некоторое $\underline{h}_{\mathrm{sig}}$ и выполнить $\mathcal{A}'$ со входом $\mathtt{inp}_{\mathcal{A}'} = (X_h, \underline{h}_{\mathrm{sig}})$. Если $\mathcal{A}'$ выдаёт какой-либо символ ошибки $\perp$, то $\mathtt{fork}^{\mathcal{A}'}$ выдаёт $\perp_{\mathtt{fork}^{\mathcal{A}'}}$ и завершает процесс.

В противном случае $\mathcal{A}'$ выводит некоторый $(i_{\text{sig}}, \mathtt{out})$, где $\mathtt{out} = (h_{\text{agg},i_{\text{agg}}}, h_{\text{sig},i_{\text{sig}}}, \underline{a}, \mathtt{forg})$. Второй $\underline{h}^*_{\text{sig}}$ выбирается случайным образом, последовательности $\underline{h}_{\text{sig}}$ и $\underline{h}^*_{\text{sig}}$ объединяются, как обычно, чтобы получить $\underline{h}'_{\text{sig}}$. $\mathcal{A}'$ запускается снова за исключением $(X_h, \underline{h}'_{\text{sig}})$ вслед за вторым успешным выполнением. Если $\mathcal{A}'$ выдаёт какой-либо символ ошибки $\perp$, то $\mathtt{fork}^{\mathcal{A}'}$ выводит $\perp_{\mathtt{fork}^{\mathcal{A}'}}$ и завершает процесс.

В противном случае $\mathcal{A}'$ проходит второе успешное выполнение, скажем, $(i^*_{\text{sig}}, \mathtt{out}^*)$. Если $i_{\text{sig}} \neq i^*_{\text{sig}}$, $\mathtt{fork}^{\mathcal{A}'}$ выдаёт $\perp_{\mathtt{fork}^{\mathcal{A}'}}$ и завершает процесс. В противном случае $\mathtt{fork}^{\mathcal{A}'}$ выводит

$$\mathtt{out}_{\mathtt{fork}^{\mathcal{A}'}} = (i_{\text{sig}}, (i, \mathtt{out}), (i^*, \mathtt{out}^*)).$$

Алгоритм $\mathtt{fork}^{\mathcal{A}'}$: берёт в качестве входа $\mathtt{inp}_{\mathtt{fork}^{\mathcal{A}'}} = (\mathtt{inp}, \underline{h}_{\text{agg}}) = (X_h, \underline{h}_{\text{agg}})$.

1. $\mathtt{fork}^{\mathcal{A}'}$ берёт случайные монеты для $\mathcal{A}'$, выбирает $\underline{h}_{\text{sig}} \leftarrow (\{0,1\}^\eta)^q$, собирает $\mathtt{inp}_{\mathcal{A}'} = (\mathtt{inp}, \underline{h}_{\text{agg}})$.
2. $\mathtt{fork}^{\mathcal{A}'}$ запускает $\mathcal{A}'$, используя $\mathtt{inp}_{\mathcal{A}'}$.
3. Если $\mathcal{A}'$ выдаёт $\perp_{\mathcal{A}'}$, $\mathtt{fork}^{\mathcal{A}'}$ выводит $\perp_{\mathtt{fork}^{\mathcal{A}'}}$ и завершает процесс.
4. В противном случае $\mathcal{A}'$ выводит некоторый $\mathtt{out}_{\mathcal{A}'} = (i_{\text{sig}}, \mathtt{out})$ где
$$\mathtt{out} = (i_{\text{sig}}, \ell_{\text{sig}}, \pi_{\text{sig}}, i_{\text{agg}}, h_{\text{sig},i_{\text{sig}}}, h_{\text{agg},i_{\text{agg}}}, \underline{a}, \mathtt{forg}).$$
5. $\mathtt{fork}^{\mathcal{A}'}$ выбирает $\underline{h}'_{\text{sig}} \leftarrow (\{0,1\}^\eta)^q$, задаёт объединяющий индекс $j = i_{\mathtt{sig}}$ и, как обычно, объединяет последовательности ответов на запросы оракула
$$\underline{h}''_{\text{sig}} = (h_{\text{sig},1}, h_{\text{sig},2}, \ldots, h_{\text{sig},j-1}, h'_{\mathtt{sig},j}, h'_{\mathtt{sig},j+1}, \ldots).$$
6. $\mathtt{fork}^{\mathcal{A}'}$ запускает $\mathcal{A}'$, используя $\mathtt{inp}'_{\mathcal{A}'} = (\mathtt{inp}, \underline{h}''_{\text{sig}})$.
7. Если $\mathcal{A}'$ выдаёт $\perp$, $\mathtt{fork}^{\mathcal{A}'}$ выводит $\perp$.
8. В противном случае $\mathcal{A}'$ выводит некоторый $\mathtt{out}^*_{\mathcal{A}'} = (i^*_{\text{sig}}, \mathtt{out}^*)$ где
$$\mathtt{out}^* = (i^*_{\text{sig}}, \ell^*_{\text{sig}}, \pi^*_{\text{sig}}, i^*_{\text{agg}}, h''_{\text{sig},i^*_{\text{sig}}}, h_{\text{agg},i^*_{\text{agg}}}, \underline{a}^*, \mathtt{forg}^*).$$
9. Если $i_{\text{sig}} \neq i^*_{\text{sig}}$, выводится $\perp_{\mathtt{fork}^{\mathcal{A}'}}$ и процесс завершается.
10. В противном случае выводится $(i_{\text{sig}}, \mathtt{out}')$ где $\mathtt{out}' = (\mathtt{out}, \mathtt{out}^*)$.

Следующая лемма очевидна: коэффициенты накопления для P_ℓ должны быть определены до того, как будет запрошен оракул $\mathcal{H}_{\text{sig}}$ при помощи $(M, P_\ell, L_\ell, R_\ell)$ за исключением ничтожной вероятности. Следовательно, запросы, определяющие $h_{\text{agg},i_{\text{agg}}}$ и $h^*_{\text{agg},i^*_{\text{agg}}}$, делаются до форка, поэтому они должны быть теми же запросами.

Лемма А.4.1. Успешный выход $\mathtt{fork}^{\mathcal{A}'}$ содержит $i_{\text{agg}} = i^*_{\text{agg}}$ за исключением ничтожной вероятности.

Доказательство. Угадать выход $\mathcal{H}_{\text{sig}}(M, P_\ell, L_\ell, R_\ell)$ до того, как случится P_ℓ, с вероятностью самое большее $(\mathfrak{p} - q)^{-1}$. Этого достаточно, но совсем не обязательно: также можно узнать P_ℓ, не делая запросов коэффициентов накопления.

Следовательно, за исключением некоторой вероятности, составляющей самое большее $(\mathfrak{p} - q)^{-1}$, P_ℓ можно узнать до того, как будет сделан этот запрос. При условии, что $\mathcal{A}'$ известно P_ℓ, вероятность, что P_ℓ будет угадано без вычисления каких-либо коэффициентов накопления, также составляет самое большее $(\mathfrak{p} - q)^{-1}$.

Вероятность, что такой порядок не сохранится, ограничивается сверху $(\mathfrak{p}-q)^{-1} + (1-(\mathfrak{p}-q)^{-1})(\mathfrak{p}-q)^{-1} = (2-(\mathfrak{p}-q)^{-1})(\mathfrak{p}-q)^{-1})$. Таким образом, вероятность, что данный порядок сохранится, составляет по крайней мере $(1-(\mathfrak{p}-q)^{-1})^2$. □

Вероятность приемлемости для $\mathtt{fork}^{\mathcal{A}'}$ обеспечивается общей леммой реализации форка. Из-за нашего выбора варианта реализации форка запрос $\mathcal{H}_{\mathrm{sig}}$ в двух полученных транскриптах имеет одинаковый вход $(M, P_\ell, L_\ell, R_\ell)$, но различные выходы. Это условие, что $c_\ell \neq c'_\ell$ взято из *уравнений и неравенств системы подделки по дискретному логарифму*. Мы оборачиваем этот алгоритм алгоритмом $\mathcal{A}''$, который отклоняет определённые исполнения и переформатирует выходы.

Алгоритм $\mathcal{A}''$: 1. В качестве входа берёт некоторый $\mathtt{inp}_{\mathtt{fork}^{\mathcal{A}'}} = (X_h, \underline{h}_{\mathrm{agg}})$.

2. Выбирает некоторые случайные монеты $\rho = \rho_{\mathtt{fork}^{\mathcal{A}'}}$.
3. Выполняет $\mathtt{fork}^{\mathcal{A}'}$, используя $\mathtt{inp}_{\mathtt{fork}^{\mathcal{A}'}}$ и эти случайные монеты.
4. Если результатом является $\perp_{\mathtt{fork}^{\mathcal{A}'}}$, выводится $\perp_{\mathcal{A}''}$ и процесс завершается. В противном случае объединяются $\mathtt{out}_{\mathtt{fork}^{\mathcal{A}'}} = (i_{\mathrm{sig}}, \mathtt{out}')$ где $\mathtt{out}' = (\mathtt{out}, \mathtt{out}^*)$ и $\mathtt{out}_{\mathtt{fork}^{\mathcal{A}'}}$.
5. Находится $\ell_{\mathrm{sig}}, \pi_{\mathrm{sig}} \in \mathtt{out}$; находится $\ell^*_{\mathrm{sig}}, \pi^*_{\mathrm{sig}} \in \mathtt{out}^*$.
6. Если $\ell_{\mathrm{sig}} \neq \ell^*_{\mathrm{sig}}$ или $\pi_{\mathrm{sig}} \neq \pi^*_{\mathrm{sig}}$, выводится $\perp$ и процесс завершается. В противном случае объединяются $\mathtt{out}_{\mathcal{A}''} = (i_{\mathrm{agg}}, \mathtt{out}')$ и выход $\mathtt{out}_{\mathcal{A}''}$.

Доказательство значимости вероятности приемлемости $\mathcal{A}''$ довольно схоже с доказательством, представленным в работе [13] для не пороговых LSAG-подписей. Мы пропустим это, но отметим, что пороговое свойство нашей схемы подразумевает отсутствие разницы при определении вероятности приемлемости.

Теперь мы делаем форк по $i_{\mathtt{agg}}$. Таким образом, все запросы, сделанные до этого момента, остаются теми же. Более того, так как коэффициенты накопления по выбранным со злым умыслом ключам определяются случайными монетами, а не по ряду запросов хешей и из-за нашей структуры моделей $\mathcal{H}_{\mathtt{agg}}$, эти случайные монеты всегда выбираются до появления выхода для честного ключа. Следовательно, если некоторый алгоритм принимает решения адаптивно на основе предыдущего входа, случайные монеты, выбранные для коэффициентов накопления по выбранным со злым умыслом ключей, будут идентичными в двух ветках с вероятностью, равной 1.

Алгоритм $\mathtt{fork}^{\mathcal{A}''}$: В качестве входа берёт некоторый $\mathtt{inp}_{\mathtt{fork}^{\mathcal{A}''}} = X_h$.

1. Выбирает некоторые случайные монеты $\rho = \rho_{\mathcal{A}''}$.
2. $\mathtt{fork}^{\mathcal{A}''}$ выбирает $\underline{h}_{\mathrm{agg}} \leftarrow (\{0,1\}^\eta)^q$ и задаёт $\mathtt{inp}_{\mathcal{A}''} = \mathtt{inp}_{\mathtt{fork}^{\mathcal{A}'}} = (X_h, \underline{h}_{\mathrm{agg}})$.
3. $\mathtt{fork}^{\mathcal{A}''}$ запускает $\mathcal{A}''$, используя $\mathtt{inp}_{\mathcal{A}''}$.
4. Если $\mathcal{A}''$ выдаёт $\perp_{\mathcal{A}''}$, $\mathtt{fork}^{\mathcal{A}''}$ выводит $\perp_{\mathtt{fork}^{\mathcal{A}''}}$ и завершает процесс. В противном случае $\mathtt{fork}^{\mathcal{A}''}$ получает $(i_{\mathrm{agg}}, \mathtt{out}_{\mathcal{A}''})$.
5. $\mathtt{fork}^{\mathcal{A}''}$ выбирает $\underline{h}'_{\mathrm{agg}} \leftarrow (\{0,1\}^\eta)^q$, задаёт объединяющий индекс $j = i_{\mathtt{agg}}$ и, как обычно, объединяет последовательности ответов на запросы оракула

$$\underline{h}''_{\mathrm{agg}} = (h_{\mathrm{agg},1}, h_{\mathrm{agg},2}, \ldots, h_{\mathrm{agg},j-1}, h'_{\mathtt{agg},j}, h'_{\mathtt{agg},j+1}, \ldots),$$

   и объединяет $\mathtt{inp}'_{\mathcal{A}''} = (X_h, \underline{h}''_{\mathrm{agg}})$
6. $\mathtt{fork}^{\mathcal{A}''}$ запускает $\mathcal{A}''$, используя $\mathtt{inp}'_{\mathtt{fork}^{\mathcal{A}''}}$.
7. Если $\mathcal{A}''$ выдаёт $\perp$, $\mathtt{fork}^{\mathcal{A}''}$ выводит $\perp$. В противном случае $\mathtt{fork}^{\mathcal{A}''}$ получает некоторый $(i^*_{\mathrm{agg}}, \mathtt{out}^*_{\mathcal{A}''})$.
8. Если $i_{\mathrm{agg}} \neq i^*_{\mathrm{agg}}$, выводится $\perp_{\mathtt{fork}^{\mathcal{A}''}}$ и процесс завершается.

9. В противном случае выводится $(i_{\mathtt{agg}}, \mathtt{out}_{\mathtt{fork}^{\mathcal{A}''}})$, где $\mathtt{out}_{\mathtt{fork}^{\mathcal{A}''}} = (\mathtt{out}_{\mathcal{A}''}, \mathtt{out}^*_{\mathcal{A}''})$

Вероятность приемлемости $\mathtt{fork}^{\mathcal{A}''}$ ограничивается снизу некоторой значимой функцией, соответствующей общей лемме реализации форка, ограниченной снизу:

$$\mathtt{acc}_{\mathtt{fork}^{\mathcal{A}''}} \geq \mathtt{acc}_{\mathcal{A}''} \left(\frac{\mathtt{acc}_{\mathcal{A}''}}{q} - \frac{1}{2^{\eta}} \right).$$

Наконец, мы строим наш решатель. Следует отметить, что следующий алгоритм будет выполнен успешно только в случае успешного выполнения $\mathtt{fork}^{\mathcal{A}''}$, поэтому и вероятность успеха будет идентичной, на чём основывается наша теорема.

Алгоритм $\mathcal{B}$: $\mathcal{B}$ имеет скрытый доступ к $\mathtt{fork}^{\mathcal{A}''}$. В качестве входа $\mathcal{B}$ берёт честный публичный ключ X_h, получает доступ к случайному оракулу и выводит дискретный логарифм x_h.

1. В качестве входа $\mathcal{B}$ берёт некоторый X_h.
2. $\mathcal{B}$ выполняет $\mathtt{fork}^{\mathcal{A}''}$, используя $\mathtt{inp} = \{X_h\}$.
3. Если $\mathtt{fork}^{\mathcal{A}''}$ выдаёт $\perp_{\mathtt{fork}^{\mathcal{A}''}}$, $\mathcal{B}$ выводит $\perp_{\mathcal{B}}$ и завершает процесс. В противном случае $\mathcal{B}$ получает $(i_{\mathrm{agg}}, \mathtt{out}_{\mathtt{fork}^{\mathcal{A}''}})$.
4. $\mathcal{B}$ проверяет $\mathtt{out}_{\mathtt{fork}^{\mathcal{A}''}}$, чтобы выделить четыре подписи, $\sigma^{(j)}$ (для $j \in [4]$), индексы запросов подписания $i^{(j)}_{\mathrm{sig}}$, ответы на запросы подписания $h^{(j)}_{\mathrm{sig},i^{(j)}_{\mathrm{sig}}}$, индексы кольца $\ell^{(j)}_{\mathrm{sig}}$ и $\pi^{(j)}_{\mathrm{sig}}$ индексы накопления i_{agg} и ответы на запросы накопления $h^{(j)}_{\mathrm{agg},i^{(j)}_{\mathrm{agg}}}$.
5. $\mathcal{B}$ проверяет следующую систему уравнений и неравенств:

$$\begin{aligned}
i^{(1)}_{\mathrm{sig}} &= i^{(2)}_{\mathrm{sig}} & \ell^{(1)}_{\mathrm{sig}} &= \ell^{(2)}_{\mathrm{sig}} & \pi^{(1)}_{\mathrm{sig}} &= \pi^{(2)}_{\mathrm{sig}} \\
i^{(3)}_{\mathrm{sig}} &= i^{(4)}_{\mathrm{sig}} & \ell^{(3)}_{\mathrm{sig}} &= \ell^{(4)}_{\mathrm{sig}} & \pi^{(3)}_{\mathrm{sig}} &= \pi^{(4)}_{\mathrm{sig}} \\
i^{(i)}_{\mathrm{agg}} &= i^{(j)}_{\mathrm{agg}} \text{ для каждого } i,j & h^{(1)}_{\mathrm{agg},i_{\mathrm{agg}}} &= h^{(2)}_{\mathrm{agg},i_{\mathrm{agg}}} & h^{(3)}_{\mathrm{agg},i_{\mathrm{agg}}} &= h^{(4)}_{\mathrm{agg},i_{\mathrm{agg}}} \\
h^{(1)}_{\mathrm{sig},i^{(1)}_{\mathrm{sig}}} &\neq h^{(2)}_{\mathrm{sig},i^{(1)}_{\mathrm{sig}}} & h^{(3)}_{\mathrm{sig},i^{(1)}_{\mathrm{sig}}} &\neq h^{(4)}_{\mathrm{sig},i^{(1)}_{\mathrm{sig}}} & h^{(1)}_{\mathrm{agg},i_{\mathrm{agg}}} &\neq h^{(3)}_{\mathrm{agg},i_{\mathrm{agg}}}
\end{aligned}$$

   и выводит $\perp_{\mathcal{B}}$, если происходит ошибка.
6. Из каждой $\mathtt{forg}^{(j)}$ можно выделить случайные данные подписи $s^{(j)}_{\ell}$.
7. $\mathcal{B}$ выводит

$$\widehat{x}_h := (h^{(1)}_{\mathrm{agg},i_{\mathrm{agg}}} - h^{(3)}_{\mathrm{agg},i_{\mathrm{agg}}})^{-1} \left(\frac{s^{(2)}_{\ell} - s^{(1)}_{\ell}}{h^{(2)}_{\mathrm{sig},i^{(2)}_{\mathrm{sig}}} - h^{(1)}_{\mathrm{sig},i^{(1)}_{\mathrm{sig}}}} - \frac{s^{(4)}_{\ell} - s^{(3)}_{\ell}}{h^{(3)}_{\mathrm{sig},i^{(3)}_{\mathrm{sig}}} - h^{(4)}_{\mathrm{sig},i^{(4)}_{\mathrm{sig}}}} \right).$$

   Безусловно, $\mathcal{B}$ не выполняется только в том случае, если не выполняется $\mathtt{fork}^{\mathcal{A}''}$ или же если система не верифицирована, но это уже подсобытие невыполнения $\mathtt{fork}^{\mathcal{A}''}$. Таким образом, вероятность того, что $\mathcal{B}$ будет не выполнен, ограничена сверху вероятностью того, что не будет выполнен $\mathtt{fork}^{\mathcal{A}''}$.

B Другие свойства, кроме невозможности подделки

Мы определяем правильность и связываемость, а также описываем некоторые свойства безопасности, такие как неопределённость подписанта и её связь с невозможностью определения того, как накапливаются ключи. Правильность и связываемость учитывают действия получестных злоумышленников, использующих алгоритмы, соответствующие спецификации (вместе с тем они могут предпринять дополнительные шаги, не предусмотренные спецификацией). В данном случае это группа «странных, но честных» друзей, желающих вместе создать подпись.

Лемма B.0.1. Пример 4.0.4 правилен.

Доказательство. В случае $S((\mathfrak{m}, \underline{P}, \pi, \underline{x}), (\mathfrak{m}^*, \sigma))$, где $\sigma = (c_1, \underline{s})$ and $\mathfrak{m}^* = (\mathfrak{m}, \underline{P}, J, \mathtt{aux})$, получестный подписант вычисляет $c_{\pi+1}$, $c_{\pi+2}$, ..., c_r и c_1 при помощи обычных/честных запросов, отправляемых $\mathcal{H}_{\text{sig}}$, с вероятностью, равной 1. Кроме того, получестный подписант также отправляет $c_2, \ldots, c_{\pi-1}, c_\pi$ при помощи обычных/честных запросов, отправляемых $\mathcal{H}_{\text{sig}}$, с вероятностью, равной 1. Получестный подписант также по спецификации убеждается в соответствии уравнений верификации при помощи секретного ключа p_π. Следовательно, получестный верификатор при наличии $\mathfrak{m}^*$ и $\sigma = (c_1, (s_1, s_2, \ldots, s_r))$ вычисляет следующее

$$\begin{aligned}
L'_1 &:= c_1 G + s_1 P_1 & R'_1 &:= c_1 H_1 + s_1 J & c'_2 &:= \mathcal{H}_{\text{sig}}(M, P_1, R_1, L_1) \\
L'_2 &:= c'_2 G + s_2 P_2 & R'_2 &:= c'_2 H_2 + s_2 J & c'_3 &:= \mathcal{H}_{\text{sig}}(M, P_2, R_2, L_2) \\
&\vdots & &\vdots & &\vdots \\
L'_r &:= c'_r G + s_r P_r & R'_r &:= c'_r H_r + s_r J & c'_1 &:= \mathcal{H}_{\text{sig}}(M, P_r, R_r, L_r)
\end{aligned}$$

и получает $c'_1 = c_1$ с вероятностью, равной 1. □

Лемма B.0.2. Пример 4.0.4 может быть связан.

Доказательство. В случае $S'(\mathfrak{m}^*_1, \sigma_1, \mathfrak{m}^*_2, \sigma_2)$ получестный злоумышленник использовал один и тот же ключ в каждой подписи. Другими словами, для колец $\underline{P}_1$, $\underline{P}_2$ существует общий ключ; то есть для некоторых индексов i_1, i_2, $(\underline{P}_1)_{i_1} = (\underline{P}_2)_{i_2} = P_{\text{common}}$. В обеих подписях образом ключа будет $J = P_{\text{common}} \mathcal{H}_{\text{ki}}(P_{\text{common}})$, и поэтому связывание происходит с вероятностью, равной 1. □

Мы изменяем определение анонимности, дополняя его членами кольца, выбранными со злым умыслом, как это описано в работе [4], учитывая таким образом накопление ключей; более сильное определение, представленное в работе [4] с *полным раскрытием ключей* невозможно удовлетворить в случае со связываемыми транзакциями с кольцевыми подписями CryptoNote, которые используются Monero.

Определение B.0.3 (Невозможность порогового определения подписанта при наличии членов кольца, выбранных со злым умыслом)**.** Допустим, $f(-)$ положительным полиномом. Рассмотрим следующую игру:

1. Множество пар приватных-публичных ключей $\{(x_i, X_i)\}_{i \in [f(\lambda)]}$ выбирается запросчиком при помощи `KeyGen`. Обозначим $SK = \{x_i\}$ and $PK = \{X_i\}$.

2. Публичные ключи PK отправляются $\mathcal{A}$, у которого есть доступ к подписывающему оракулу $\mathcal{SO}$.

3. $\mathcal{A}$ выводит сообщение $\mathfrak{m}$ два непустых мультимножества накопленных ключей $\underline{X}^{(0)}$, $\underline{X}^{(1)} \subset PK$ и кольцо $\underline{P}$ так, чтобы $P_{i_j} = \Phi(\underline{X}^{(j)})$ для $j = 0, 1$.

4. Запросчик выбирает случайный бит b, вычисляет $(\mathtt{m}^*, \sigma) \leftarrow \mathtt{Sign}(\mathtt{m}, \underline{P}, \pi_b)$, где π_b означает индекс P_{i_b} in $\underline{P}$, and sends $(\mathtt{m}^*, \sigma)$ $\mathcal{A}$.

5. $\mathcal{A}$ выводит бит b'.

$\mathcal{A}$ побеждает в игре, если $b' = b$, а образы ключей для P_{i_0} и P_{i_1} не появляются в выходе какого-либо запроса, отправленного $\mathcal{SO}$ от $\mathcal{A}$.

Следующая лемма, которая определяется невозможностью определения того, как накапливаются ключи, очевидна в том случае с получестным злоумышленником, так как распределение $(c_1, \underline{s})$ в рамках обеих схем подписи определяется хеш-функцией $\mathcal{H}_{\text{sig}}$ и выбором подписанта, сделанным для $\underline{s}$. В рамках модели случайного оракула c_1 равномерно распределяется по $\mathbb{Z}_{\mathfrak{p}}$. В случае с получестным злоумышленником каждое $s_i \in \underline{s}$ также равномерно распределяется по $\mathbb{Z}_{\mathfrak{p}}$ и, более того, все они не зависят друг от друга.

Лемма B.0.4. Подписи, создаваемые получестными злоумышленниками, при помощи Примера 4.0.4, статистически неотличимы от LSAG-подписей.

Как следствие, злоумышленник, который может нарушить неопределённость подписанта из Примера 4.0.4, подобным образом должен быть способен нарушить неопределённость подписанта в LSAG-подписях.

Следствие B.0.5. В случае с Примером 4.0.4 подписант остаётся неопределённым при наличии членов кольца, выбранных со злым умыслом.