БДЗ по прикладной криптографии

Фирсов Георгий, M21-507

12 января 2023 г.

Содержание

Задание 1

При известном заранее значении D нарушитель может единожды найти такое значение z, что:

$$\texttt{SHA256}(z) < \frac{2^n}{D}. \tag{1}$$

Это потребует некоторого времени, но идея в том, что это делается единожды и заранее.

Далее при обнародовании x нарушитель вычисляет $y = z \oplus x$. При этом верна следующая цепочка:

$$H(x, y) = H(x, x \oplus z) = \texttt{SHA256}(x \oplus x \oplus z) = \texttt{SHA256}(z) < \frac{2^n}{D}, \tag{2}$$

то есть нарушитель может для каждого x найти такой y, что $H(x, y) < \frac{2^n}{D}$ за некоторое константное время.

Задание 2

1. На рисунке 1 представлен процесс вычисления хэша R_i (или просто искомого коммитмента S), входящего в заголовок i-го блока, при помощи троичного дерева Меркла. Фактически это достаточно очевидное само по себе переложение процесса вычисления на троичное дерево вместо двоичного.

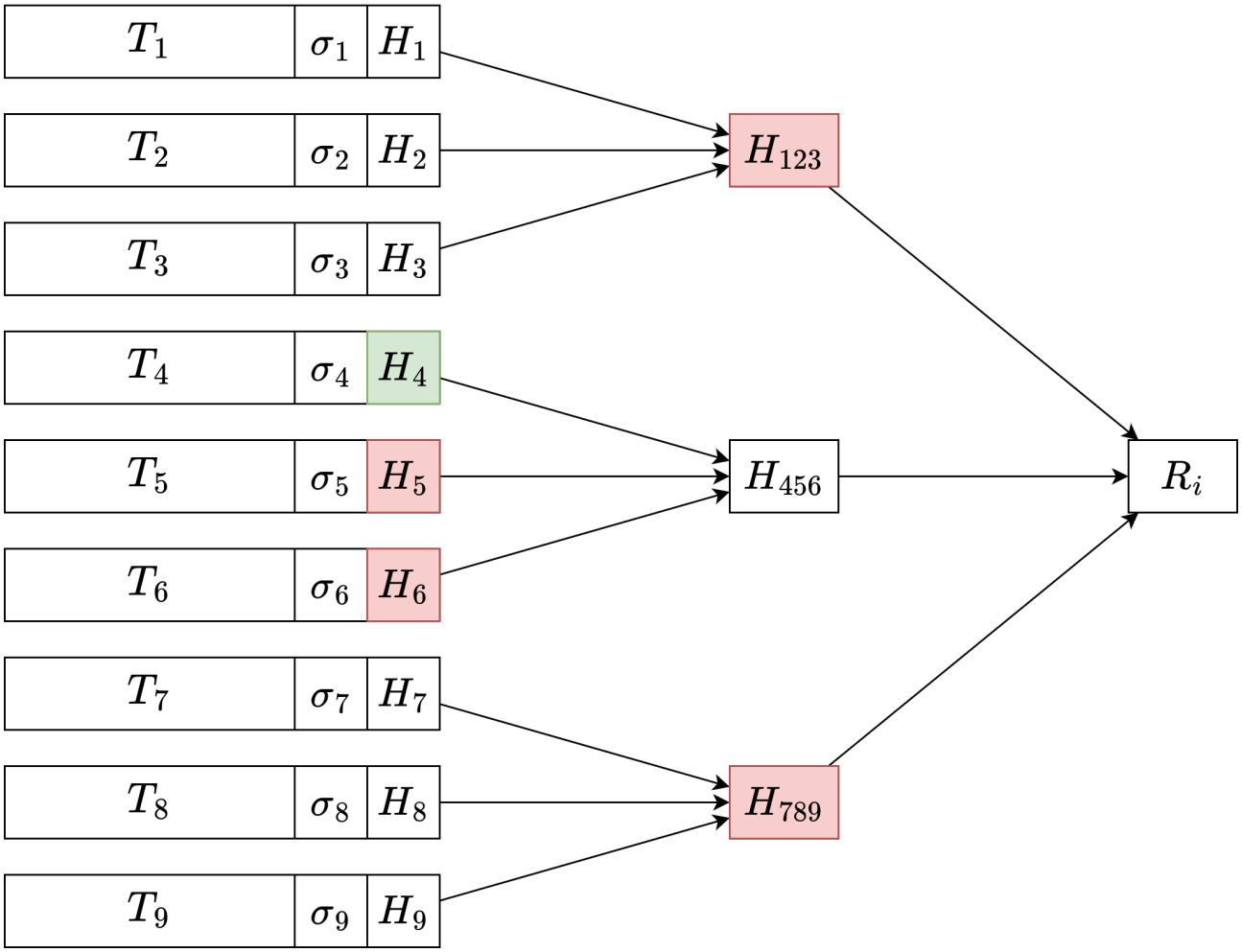

Рис. 1: Схема вычисления хэша при помощи троичного дерева Меркла

Для того, чтобы доказать вхождение транзакции T_4 (хэш которой выделен зеленым на рисунке 1) требуется предоставить Борису следующие значения: H_5, H_6 (соседние хэши) и H_{123}, H_{789} (хэши соседних поддеревьев). Данные значения на рисунке 1 выделены красным.

Борис вычисляет $\tilde{H}_{456}$ на основе известного H_4 и предоставленных H_5, H_6 ($\tilde{H}_{456} = H(H_4||H_5||H_6)$), после чего на основе полученного значения и предоставленных H_{123}, H_{789} рассчитывает $\tilde{S} = H(H_{123}||\tilde{H}_{456}||H_{789})$. Если $\tilde{S} = R_i$, то T_4 содержится в блоке (при условии, что все предоставленные хэши верны, что было бы логично при доказательстве).

2. Отметим, что путь от корня до проверяемой вершины содержит $\lceil \log_k n \rceil$ элементов дерева. Для каждого элемента необходимо предоставить $k-1$ хэш — это хэши соседних поддеревьев (а для листового уровня — соседние хэши-листы). Таким образом получается следующая формула для длины доказательства:

$$(k-1) \cdot \lceil \log_k n \rceil. \tag{3}$$

Ответ: $(k-1) \cdot \lceil \log_k n \rceil$.

3. Ясно, что для $x > 1$ выполнено: $\log_2 x < 2 \cdot \log_3 x$. Более наглядно это представлено на рисунке 2:

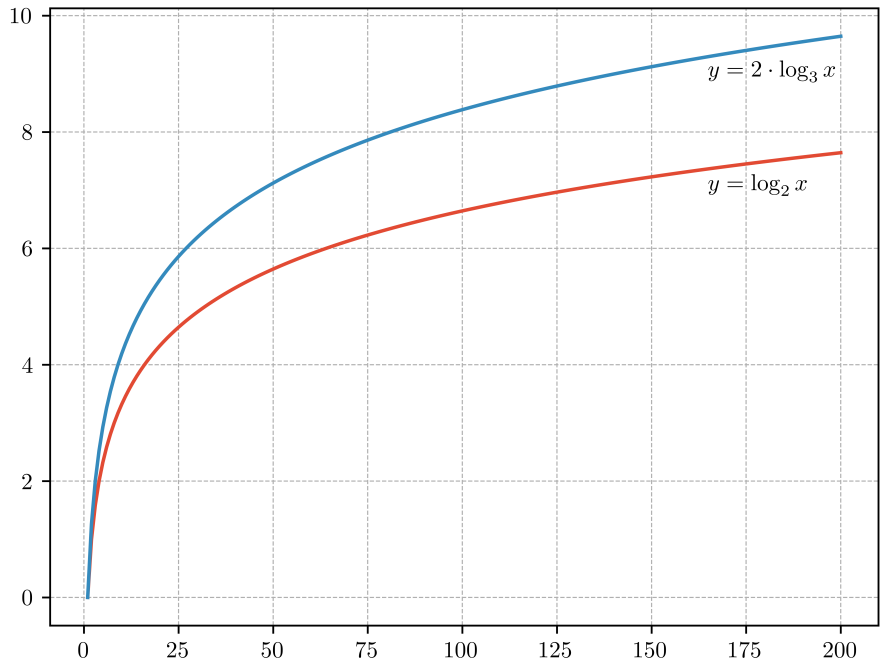

Рис. 2: графики двоичного (красная линия) и удвоенного троичного (синяя линия) логарифмов в одинаковом масштабе

Из этого следует, что использование двоичного дерева Меркла эффективнее, чем троичного (и на самом деле какого-бы то ни было еще), так как длина доказательства для него будет меньше.

Ответ: двоичное дерево использовать оптимальнее, чем троичное.

Задание 3

Все пункты данной задачи будут для наглядности проиллюстрированы конкретным примером, на основе которого далее будет дано обобщенное решение.

1. На рисунке 3 изображена цепочка блоков: зеленым выделен блок, заголовок которого хранится в памяти компьютера Бориса, желтым — блок, в который входит транзакция, для которой требуется доказать вхождение в цепь. Через R_i обозначается хэш-вершина дерева Меркла для транзакций i-го блока, через H_{i-1} — хэш заголовка предыдущего (то есть с индексом $i-1$) блока.

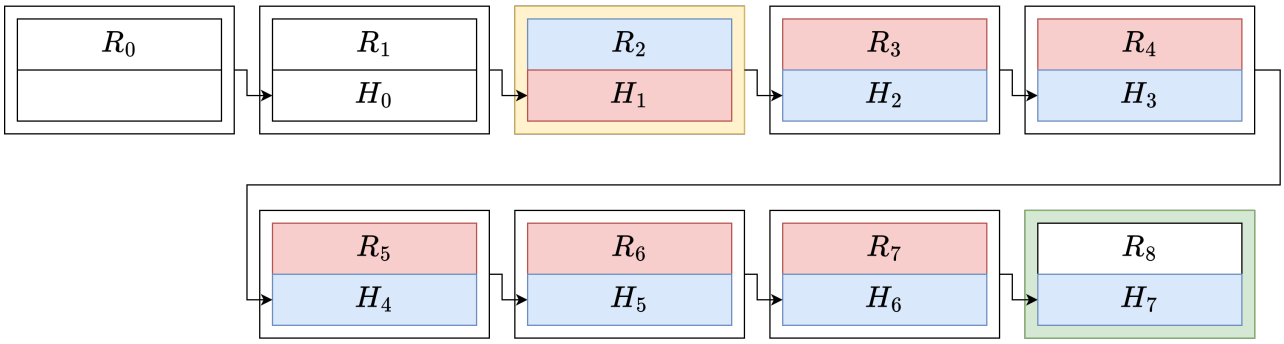

Рис. 3: Доказательство нахождения транзакции из блока с индексом 2 в цепи

Красным цветом на рисунке 3 обозначаются данные, которые предоставляются Анной для доказательства, синим — вычисляемые Борисом значения. Следует отметить, что для вычисления значения R_2 (в изображенном примере) Борису требуется получить от Анны еще $\lceil \log_2 n \rceil$ хэшей для дерева Меркла (см. задачу 2). Проведем обобщение.

Ответ: Анне требуется предоставить Борису следующие данные:

- $\lceil \log_2 n \rceil$ хэшей для дерева Меркла для вычисления хэша R_i, где i — индекс блока, в который входит исследуемая транзакция;
- хэш заголовка предыдущего блока H_{i-1}, на основе которого в совокупности с рассчитанным R_i вычисляется H_i;
- хэши-вершины деревьев Меркла для блоков от $i+1$ до $c-1$, где c — индекс «текущего» блока.

2. Для иллюстрации данного пункта будет использоваться рисунок 3. В этом случае $k = 6$ (то есть прямо как и в условии). Размер хэша 32 байта (это 256 бит SHA256). Как было сказано в предыдущем пункте, требуется передать:

   - $\lceil \log_2 n \rceil$ хэшей для дерева Меркла: $\lceil \log_2 1024 \rceil \cdot 32 = 320$ байт;
   - хэш заголовка предыдущего блока: 32 байта;
   - $k-1$ хэшей-вершин деревьев Меркла блоков между блоком с транзакцией и головным: $(6-1) \cdot 32 = 160$ байт.

   Итоговая формула: $(\lceil \log_2 n \rceil + k)\, h$, где h — размер хэша.

   Ответ: $320 + 32 + 160 = 512$ байт. Формулу для расчета см. выше.

3. Оптимизация происходит за счет того, что при доказательстве можно пропустить проверку некоторых промежуточных блоков. Пример ускорения показан на рисунке 4 для двух случаев: когда текущий блок у Бориса имеет индекс 8 и 9. Красной линией показан «обходной» путь, позволяющий не проверять промежуточные блоки. Так, в первом случае достаточно предоставить только хэши для проверки дерева Меркла генезис-блока, на основе вычисленного значения вершины которого вычисляется хэш

заголовка H_0. Во втором случае текущий блок чуть дальше, а поэтому требуется дополнительно предоставить хэш H_7 и хэш-вершину дерева Меркла для предыдущего блока R_8. При этом данные случаи являются в некотором смысле одними из самых оптимальных.

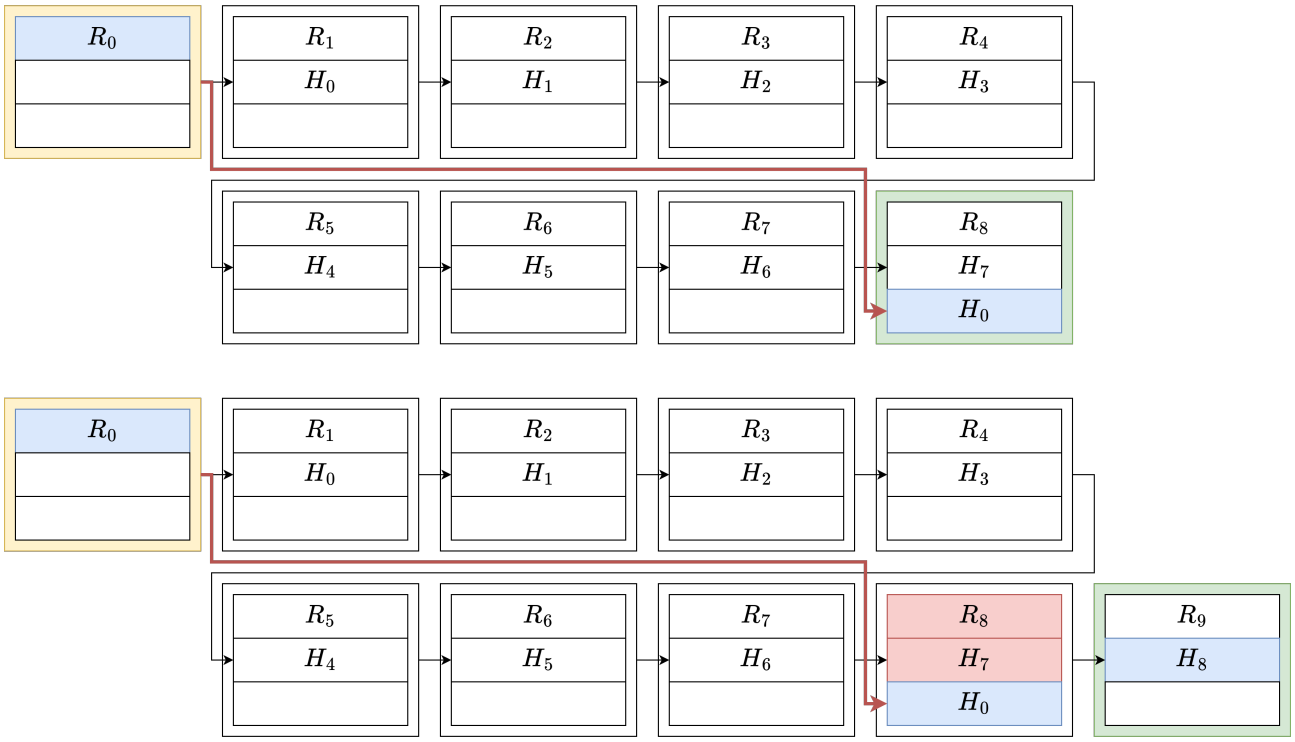

Рис. 4: Пример оптимизации доказательства при помощи рассматриваемой схемы

Рассмотрим теперь худший случай: доказательство вхождения транзакции из блока с индексом 1 при индексе текущего $k = 2^d - 1$ для $d \in \mathbb{N}$. Рассмотрим пример «пути» проведения доказательства на более высоком уровне абстракции (рис. 5). Красным цветом обозначены «обходные» пути.

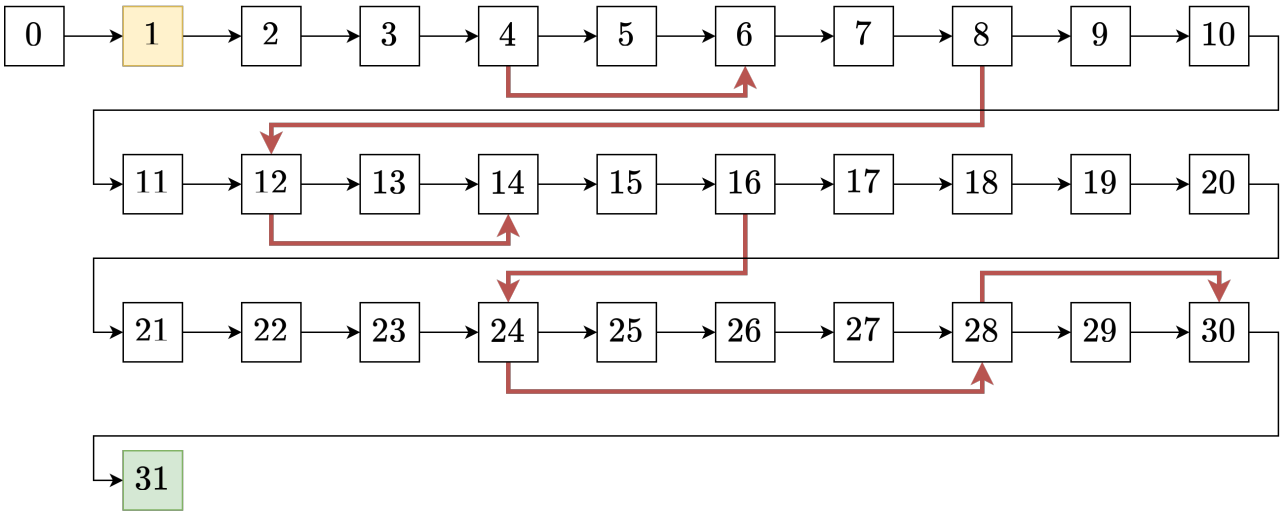

Рис. 5: Пример «худшего» случая доказательства при помощи рассматриваемой схемы для $k = 31$

Как видно из рисунков 3 и 4, один «обычный» переход «стоит» 1 хэш (только хэш-

вершина дерева Меркла для блока, в который осуществляется переход), а «обходной» путь — 2 хэша (тот же хэш-вершина дерева Меркла, а также хэш заголовка блока, который предшествует тому, в который осуществляется переход). Таким образом, полный объем доказательства составляет:

$$\mathfrak{v}(n,k) = \lceil \log_2 n \rceil + 2 \cdot \mathfrak{l}(k) + \mathfrak{s}(k), \tag{4}$$

где $\mathfrak{l}(k)$ — количество «обходных» переходов, $\mathfrak{s}(k)$ — количество «обычных» переходов. При этом хэш заголовка генезис-блока уже учтен в данной формуле, так как переход от 1-го блока ко 2-му как раз ему и соответствует (в остальных случаях переход соответствует, как и было сказано, хэшу-вершине дерева Меркла).

В приложении А предоставлен код, который для входного параметра k возвращает количество «обходных» путей, количество «обычных» путей (то есть путей в следующий блок) и список блоков, для которых требуется доказательство правильности хэша.

В таблице 1 представлены результаты работы функции `get_path` из приложенного листинга для $d = \overline{2, 15}$.

Таблица 1: Значения $\mathfrak{l}(k)$ и $\mathfrak{s}(k)$ для значений $k = 2^d - 1$ для последовательных значений $d = \overline{2, 15}$

d	k	$\mathfrak{l}(k)$	$\mathfrak{s}(k)$
2	3	0	2
3	7	1	4
4	15	3	6
5	31	6	8
6	63	10	10
7	127	15	12
8	255	21	14
9	511	28	16
10	1023	36	18
11	2047	45	20
12	4095	55	22
13	8191	66	24
14	16383	78	25
15	32767	91	26

Обозначим: $l_d = \mathfrak{l}(k)$, $s_d = \mathfrak{s}(k)$, где $d = \log_2(k+1)$ и заметим следующее:

$$\begin{aligned} l_d &= l_{d-1} + (d-2), l_2 = 0 \\ s_d &= 2 \cdot (d-1). \end{aligned} \tag{5}$$

Далее получим явное выражение для l_d:

$$\begin{aligned} l_d = l_{d-2} + (d-3) + (d-2) = ... = l_2 + 1 + ... + (d-3) + (d-2) = \\ = \sum_{j=0}^{d-2} j = \frac{(d-2)(d-1)}{2}. \end{aligned} \tag{6}$$

Тогда:

$$\begin{aligned} \mathfrak{v}(n,k) &= \lceil \log_2 n \rceil + 2 \cdot \frac{(d-2)(d-1)}{2} + 2 \cdot (d-1) = \\ &= \lceil \log_2 n \rceil + d(d-1). \end{aligned} \tag{7}$$

Ответ: $\mathfrak{v}(n,k) = \lceil \log_2 n \rceil + d(k)(d(k)-1)$, где $d(k) = \log_2(k+1)$.

Задание 4

1. **Ответ:** блок с невалидными транзакциями будет принят с большей вероятностью: происходит ситуация вилки, в которой майнеры с реализацией A вероятнее всего быстрее «соберут» более длинную цепочку блоков (в которой будет как раз невалидный). Фактически цепь с невалидным блоком будет навязана все остальным. При этом (для полноты картины) имеется все-таки ненулевая вероятность того, что майнеры с реализацией B успеют собрать более длинную цепь, которая будет принята в итоге как основная, но эта вероятность ниже.

2. **Ответ:** ситуация ровно противоположная: блок с большей вероятностью окажется в побочной ветке при вилке и будет отброшен. Однако и тут существует вероятность того, что такая цепь будет принята, так как успеть «собрать» более длинную невалидную цепочку майнеры с реализацией A все-таки могут. Рассуждения аналогичны приведенным для п. 1.

Задание 5

1. Пусть η — доля вознаграждения, затрачиваемая на оплату электроэнергии, f — частота майнинга новых блоков (количество блоков в день), r — вознаграждение за майнинг одного блока, c_a^b — цена единицы валюты a, выраженная в единицах b, T — тариф.

   Тогда, искомая величина дневного потребления электроэнергии выражается следующей формулой:

   $$E = \frac{\eta f r c_{\text{BTC}}^{\text{РУБ}}}{T}. \tag{8}$$

   В нашем случае:

   $$\begin{aligned} &\eta = 0.8 \\ &f = 24 \cdot 6 = 144 \text{ (6 блоков за час)} \\ &r = 6.25 \\ &c_{\text{BTC}}^{\text{РУБ}} = c_{\text{BTC}}^{\text{USD}} \cdot c_{\text{USD}}^{\text{РУБ}} = 16858 \cdot 72,68 = 1225239,44 \text{ (данные на 22:51 05.01.2023)} \\ &T = 5,15 \text{ (Москва)} \end{aligned} \tag{9}$$

   Тогда:

   $$E = \frac{0.8 \cdot 144 \cdot 6.25 \cdot 1225239,44}{5.15} = 171295611,02913 \text{ кВт} \cdot \text{ч}. \tag{10}$$

   Ответ: $E = 171295611,02913$ кВт $\cdot$ ч.

2. Ясно, что дневное потребление электроэнергии одного майнера равно: $24P$ кВт $\cdot$ ч, где P — потребляемая им мощность. Значит требуется найти количество майнеров, которое на самом деле находится достаточно просто:

   $$N = \frac{D}{H\tau}, \tag{11}$$

   где H — хэшрейт, τ — время генерации одного блока. То есть:

   $$E = \frac{24PD}{H\tau} = \frac{24 \cdot 1.7 \cdot 2^{75}}{18 \cdot 10^{12} \cdot 600} = \frac{51 \cdot 2^{78}}{108 \cdot 10^{15}} = 142720409,26 \text{ кВт} \cdot \text{ч}. \tag{12}$$

   Ответ: $E = 142720409,26$ кВт $\cdot$ ч.

3. Исходя из формулы (12) «сложность» генерации блока пропорциональна потребляемой мощности и обратно пропорциональна хэшрейту. В п. 2 наблюдается идиализированная ситуация (все устройства лучшие из возможных), а в п. 1 — реальная (есть устройства и хуже), то есть в п. 1 устройства имеют либо меньший хэшрейт, либо больше потребляют мощности, либо и то и то (в вырожденном случае никаких отличий нет — все устройства самые лучшие). Таким образом «сложность» генерации блока в п. 1 заведомо больше или равна той, что в п. 2 (равенство достигается при условии использования в п. 1 всех самых лучших устройств).

   Ответ: нет, «сложность» генерации блока в п. 2 не может превысить ту же величину из п. 1.

Задание 6

Для начала следует отметить, как могут вести себя нечестные участники протокола:

- нечестный отправитель может отправить разные биты разным участникам (при этом подпись каждого сообщения будет правильной) либо не отправить бит какому-либо подмножеству участников;
- нечестный участник (не отправитель) может больше: отправить разным участникам разные биты с корректной своей подписью, проигнорировать неправильную подпись полученного от другого участника сообщения, не отправить кому-либо бит на втором шаге, передать и вернуть разные биты.

Также отметим, что при отсутствии нечестных участников свойства безопасности, валидности и тотальности выполняются. Будем иметь это в виду на протяжении дальнейшего повествования.

1. Первый случай — нечестный отправитель: в этом случае все участники, что-либо получившие от отправителя, перешлют всем остальным полученную информацию. Если отправителем были разосланы разные биты, то на втором шаге каждый честный участник получит разные биты с валидной подписью отправителя, а значит вернут 0. Если были разосланы одинаковые биты, но не всем, то на втором шаге все честные участники в любом случае от все получат одни и те же правильно подписанные биты, а значит этот бит и возвращают.

   Второй случай – нечестный участник, но не отправитель: такой нечестный участник не может кого-то заставить подумать о нечестности отправителя (только в этом случае честный участник может вернуть бит, отличный от отправленного отправителем, а значит и отличного от того, что вернут другие честные участники протокола). Если он отправит неправильный бит честному участнику, то он будет подписан неверно, а участник просто проигнорирует его. Если он не отправит бит какому-либо участнику, то последний все равно получит биты от остальных.

   Ответ: если нечестных участников не более одного, то протокол *обладает* свойством безопасности.

2. Тут возможен только случай, когда нечестный только участник, не являющийся отправителем (в противном случае свойство валидности неприменимо). В предыдущем пункте было сказано, что нечестный участник один не может заставить честного участника вернуть не тот бит: сообщения с неправильной подписью будут проигнорированы.

   Ответ: если нечестных участников не более одного, то протокол *обладает* свойством валидности.

3. Пусть нечестными являются отправитель и еще один участник. Рассмотрим следующую атаку (рис. 6): на первом шаге отправитель посылает единицу всем честным участникам с правильной подписью и ноль с правильной подписью — нечестному участнику, после чего возвращает единицу. На втором шаге нечестный участник не пересылает ноль с правильной подписью одному из участников (на рисунке 6 не пересылает правому честному участнику). Таким образом, участник, которому на втором шаге не отправили ноль, возвращает единицу — бит, присланный отправителем, а вот те участники, которые получили от нечестного участника ноль сталкиваются с ситуацией, что разные биты имеют валидные подписи отправителя, а значит возвращают ноль.

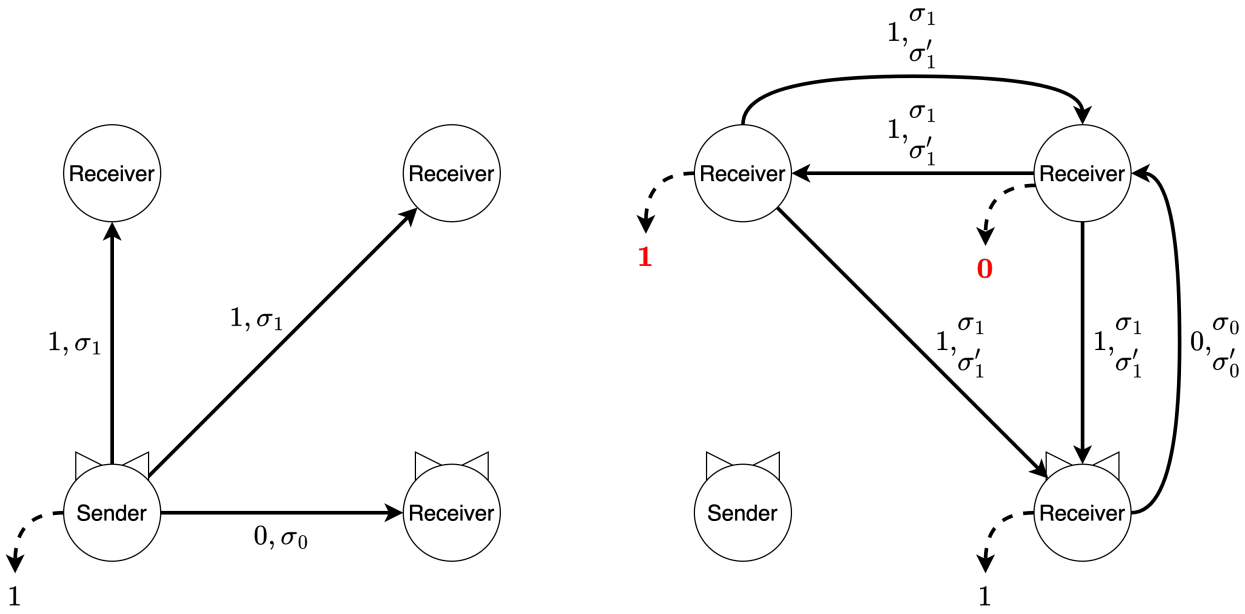

Рис. 6: Иллюстрация атаки на свойство безопасности при наличии нечестного отправителя и нечестного участника для $n = 4$

Таким образом, построена атака, позволяющая заставить каких-то честных участников вернуть ноль, а других — единицу.

Ответ: при наличии двух нечестных участников протокол *не обладает* свойством безопасности.

4. В п. 1 показано, что при наличии одного нечестного участника свойство валидности выполнено. Покажем, что это свойство сохраняется и при наличии двух нечестных участников. Отметим, что возвращаемое значение нечестных участников никак не учитывается, поэтому игнорирование неправильной подписи (при подмене бита другим нечестным участником) ни на что не влияет. Если нечестный участник (или оба сразу) отправляют подмененный бит честному, то последний просто игнорирует эти биты при принятии решения: отправка неправильного бита не приводит к нарушению свойства валидности. Если нечестный участник не отправил бит кому-либо, то это так же никак не влияет на принятие решения. При этом следует иметь в виду, что каждый честный участник уже имеет у себя бит от отправителя с правильной подписью, то есть возвращает в случае чего он именно его, так как не может получить от остальных честных участников другие биты.

Ответ: при наличии не более двух нечестных игроков свойство валидности *имеет место.*

5. Пусть, как и в п. 3, нечестными будут отправитель и один участник. Также положим, что существуют еще по меньшей мере два честных участника. Рассмотрим следующую атаку (рис. 7): отправитель пересылает единицу с правильной подписью только нечестному участнику и возвращает после этого единицу. Затем нечестный участник (уже на втором шаге) отправляет единицу с правильной подписью одному из честных участников (в общем случае — некоторому собственному подмножеству $\mathcal{V}$ множества $\mathcal{H}$ честных участников). После этого, данный участник возвращает единицу. Более никаких передач не осуществляется согласно протоколу (шагов передачи всего два). При этом другой честный участник (а в общем случае все участники из подмножества $\mathcal{H} \setminus \mathcal{V}$), который не получил ничего ни от кого, ничего в итоге не возвращает, что обозначает прямое нарушение свойства всеобщности.

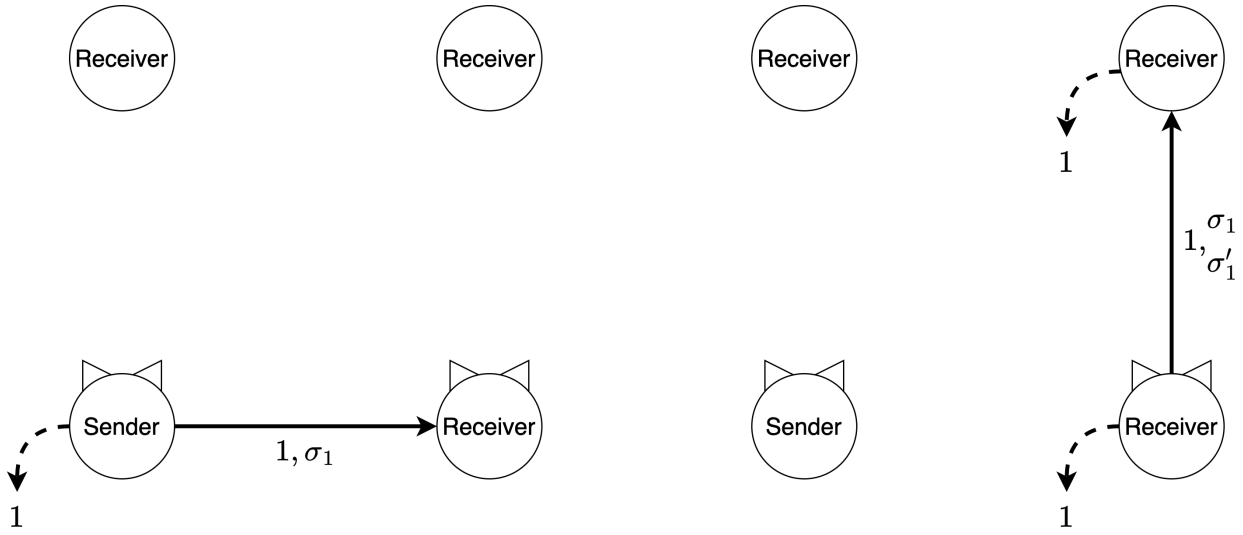

Рис. 7: Иллюстрация атаки на свойство всеобщности при наличии нечестного отправителя и нечестного участника для $n = 4$

Таким образом, при наличии как минимум двух участников протокол уже не может обладать свойством всеобщности. Рассмотрим теперь ситуацию с одним нечестным участником.

Случай первый — нечестный отправитель. Если участникам отправлены разные биты, то они так или иначе что-либо вернут, а значит свойство всеобщности выполняется — если хотя бы один честный участник возвращает бит, то все честные участники тоже в том случае что-либо возвращают. Если какому-либо участнику бит не отправлен, то он все равно получает информацию от остальных и что-либо возвращает — свойство всеобщности вновь выполнено.

Второй случай — нечестный участник, но не отправитель. Его возвращаемое значение в учет не берется. Но он не может заставить честного участника не вернуть хотя бы что-то, так как это возможно только тогда, когда этот участник *вообще ничего* не получил или получил бит отправителя с неправильной подписью, но все участники получают бит от отправителя с валидной подписью, а значит что-либо да вернут. То есть свойство всеобщности вновь выполнено.

Ответ: свойство всеобщности *соблюдается* при наличии не более одного нечестного участника, но как минимум при двух нечестных участниках возможна атака, *нарушающаяя* его.

Задание 7

1. **Ответ:** да, так как $1 \wedge b = b$, то есть результат вычисления попросту равен биту Бориса.

2. **Ответ:** нет, так как $0 \wedge b = 0$, то есть каким бы ни был бит Бориса, результат будет равняться нулю, и Анна не сможет по нему достоверно определить бит Бориса.

3. **Ответ:** суть атаки в следующем: на ключах K_L^0 и K_R^1 зашифровывается не 0, а 1, т.е. $c_{01} = E_{K_L^0}\left(E_{K_R^1}(\mathbf{1})\right)$. Борис корректно расшифрует c_{00} (получит 0) или c_{01} (получит 1). Получив от Бориса бит b, Анна сделает вывод, что у Бориса был изначально бит b. Отметим, что в остальном Анна не отклоняется от протокола.

   Борис не сможет уличить Анну в атаке, так как для него все получаемые значения выглядят ровно так же, как и в «обычном» случае: значения c_{ij}, $i, j \in \{0, 1\}$ являются неотличимыми от случайных значениями (при использовании стойкого шифра). Это объясняется случайным и независимым выбором ключей шифрования. В частности, Борис не может вычислительно различить значения $E_{K_L^0}\left(E_{K_R^1}(\mathbf{1})\right)$ (при атаке) и $E_{K_L^0}\left(E_{K_R^1}(\mathbf{0})\right)$ (атаки нет).

Задание 8

1. Пусть Анна сгенерировала свои доли r_a, r_b, r_c, а Борис — две свои s_a, s_b. Задача: вычислить s_c (Борис это значение у себя оставляет и никуда не отдает). Так как требуется построить тройку Бивера, то $r_c \oplus s_c = (r_a \oplus s_a) \wedge (r_b \oplus s_b)$. Значит требуется вычислить следующую функцию: $f = (r_a \oplus s_a) \wedge (r_b \oplus s_b) \oplus r_c$.

   Вычисление производится более чем очевидным способом (прямо как в задаче 7): Анна генерирует 10 ключей (по два для каждой из 5 переменных), зашифровывает все значения таблицы истинности и передает в случайном порядке их Борису, а также и свои нужные ключи. Далее Борис при помощи (2, 4)-забывающей передачи получает у Анны ключи, соответствующие его значениям s_a, s_b.

   После этого, Борис расшифровывает значения и верно расшифрованное будет как раз искомым значением s_c. Передавать его Анне не потребуется.

   Корректность выработки следует из построения функции f. Стойкость неформально обосновывается тем фактом, что функция f является равновероятной, а значит $\Pr[r_j = 0 | s_c = 0] = \Pr[r_j = 1 | s_c = 0] = \Pr[r_j = 0 | s_c = 1] = \Pr[r_j = 1 | s_c = 1] = \frac{1}{2}$, $j \in \{a, b, c\}$. То есть Борис не может по полученному значению достоверно восстановить биты Анны.

2. Анна генерирует значения f для своих r_1, r_b, r_c, а также для всех 4 вариантов комбинаций s_a, s_b битов Бориса. После этого Борис при помощи (1, 4)-забывающей передачи получает то значение, которое соответствует его конкретным значениям s_a, s_b.

3. Рассмотрим следующий протокол[1]. Анна имеет у себя значения $m_1, ..., m_{2^l} \in \{0, 1\}$. Борис хочет получить значение m_I, где $I \in \{1, ..., 2^l\}$.

   Анна заготавливает $2l$ ключей (каждый вырабатывается случайно и равновероятно из $\{0, 1\}^\lambda$):

$$\left(K_1^0, K_1^1\right), \left(K_2^0, K_2^1\right), ..., \left(K_l^0, K_l^1\right). \tag{13}$$

[1]Идея протокола была взята из статьи: Naor, M., Pinkas, B. (1999). Oblivious transfer and polynomial evaluation. Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing - STOC '99. doi:10.1145/301250.301312

Обозначим через $i[j]$ j-й бит l-битного числа i (т.е. $j \in \{1, ..., l\}$).

Для каждого $1 \leq i \leq 2^l$ Анна вычисляет значение $c_i = m_i \oplus \bigoplus_{j=1}^{l} F\left(K_j^{i[j]}, i\right)$.

Далее наступает шаг использования (1, 2)-забывающей передачи. Борис знает номер I сообщения, которое хочет получить, и поочередно получает ключи $K_j^{I[j]}$ для всех $j \in \{1, ..., l\}$ (то есть используется как раз l вызовов протокола (1, 2)-забывающей передачи λ-битных сообщений).

После этого Анна передает Борису все c_i для $i \in \{1, ..., 2^l\}$, а Борис восстанавливает искомое сообщение:

$$m_I = c_I \oplus \bigoplus_{j=1}^{l} F\left(K_j^{I[j]}, I\right). \tag{14}$$

Задание 9

1. Будем считать, что при возможных значениях m оно достаточно быстро находится по известным g и g^m (то есть задача дискретного логарифма для таких значений m решается за приемлемое время).

   Тогда алгоритм $\texttt{Dec}(sk, (u, v))$ описывается следующей последовательностью действий:

   - вычисляется значение $w = \frac{u^c}{v^a} = g^{m(c-ab)}$ (так как $c \neq ab$, показатель может быть равен нулю только если $m = 0$);
   - перебираются различные кандидаты $\tilde{m}$ для m, пока не будет найдено такое, что $g^{\tilde{m}(c-ab)} = w$. Это и будет расшифрованным сообщением.

2. Отметим, что значение r никак не участвует в расшифровании, поэтому алгоритм $\texttt{Add}(pk, ct_1, ct_2)$ строится достаточно очевидным способом:

$$\begin{aligned} &\texttt{Add}(pk, ct, ct') = (u_{sum}, v_{sum}) \\ &u_{sum} = u \cdot u' = g^{m+ar} g^{m'+ar'} = g^{(m+m')+a(r+r')} \\ &v_{sum} = v \cdot v' = g^{mb+cr} g^{m'b+cr'} = g^{(m+m')b+c(r+r')} \end{aligned} \tag{15}$$

3. Запишем значения u, u', v, v' в аддитивной нотации (все-таки аддитивные группы):

$$\begin{aligned} u &= (m + ra)g \\ v &= (mb + rc)g \\ u' &= (m' + r'a')h \\ v' &= (m'b' + r'c')h \end{aligned} \tag{16}$$

   В силу билинейности отношения спаривания e верно:

$$\begin{aligned} \alpha_1 &= (m + ra)(m' + r'a') \\ \alpha_2 &= (m + ra)(m'b' + r'c') \\ \alpha_3 &= (mb + rc)(m' + r'a') \\ \alpha_4 &= (mb + rc)(m'b' + r'c') \end{aligned} \tag{17}$$

   Тогда запишем:

$$mm' = C_1\alpha_1 + C_2\alpha_2 + C_3\alpha_3 + C_4\alpha_4, \tag{18}$$

для некоторых C_j, $j \in \{1, ..., 4\}$.

Теперь подставим значения для α_j, $j \in \{1, ..., 4\}$ и раскроем скобки:

$$\begin{aligned} mm' &= C_1(m + ra)(m' + r'a') + C_2(m + ra)(m'b' + r'c') + \\ &+ C_3(mb + rc)(m' + r'a') + C_4(mb + rc)(m'b' + r'c') = \\ &= mm'(C_1 + C_2b' + C_3b + C_4bb') + mr'(C_1a' + C_2c' + C_3ba' + C_4bc') + \\ &+ m'r(C_1a + C_2ab' + C_3c + C_4cb') + rr'(C_1aa' + C_2ac' + C_3ca' + C_4cc'). \end{aligned} \tag{19}$$

По равенству несложно составить следующую СЛАУ относительно C_j, $j \in \{1, ..., 4\}$:

$$\begin{cases} C_1 + C_2b' + C_3b + C_4bb' = 1 \\ C_1a' + C_2c' + C_3ba' + C_4bc' = 0 \\ C_1a + C_2ab' + C_3c + C_4cb' = 0 \\ C_1aa' + C_2ac' + C_3ca' + C_4cc' = 0 \end{cases} \tag{20}$$

Несложно заметить, что при условии, что C_j, $j \in \{1, ..., 4\}$ являются решениями данной системы, равенство (19) выполняется. Осталось доказать, что данная система совместна. Воспользуемся теоремой Кронекера-Капелли.

Приведем (расширенную) матрицу системы к каноническому виду:

$$\begin{aligned} &\left\| \begin{array}{cccc|c} 1 & b' & b & bb' & 1 \\ a' & c' & ba' & bc' & 0 \\ a & ab' & c & cb' & 0 \\ aa' & ac' & ca' & cc' & 0 \end{array} \right\| \sim \left\| \begin{array}{cccc|c} 1 & b' & b & bb' & 1 \\ a' & c' & ba' & bc' & 0 \\ a & ab' & c & cb' & 0 \\ 0 & 0 & a' & c' & 0 \end{array} \right\| \sim \\ &\sim \left\| \begin{array}{cccc|c} 1 & b' & b & bb' & 1 \\ 0 & c' - a'b' & 0 & b(c' - a'b') & -a' \\ 0 & 0 & c - ab & b'(c - ab) & -a \\ 0 & 0 & a' & c' & 0 \end{array} \right\| \sim \left\| \begin{array}{cccc|c} 1 & b' & b & bb' & 1 \\ 0 & 1 & 0 & b & \frac{-a'}{c' - a'b'} \\ 0 & 0 & 1 & b' & \frac{-a}{c - ab} \\ 0 & 0 & a' & c' & 0 \end{array} \right\| \sim \\ &\sim \left\| \begin{array}{cccc|c} 1 & b' & b & bb' & 1 \\ 0 & 1 & 0 & b & \frac{-a'}{c' - a'b'} \\ 0 & 0 & 1 & b' & \frac{-a}{c - ab} \\ 0 & 0 & 0 & c' - a'b' & \frac{aa'}{c - ab} \end{array} \right\| \sim \left\| \begin{array}{cccc|c} 1 & b' & b & bb' & 1 \\ 0 & 1 & 0 & b & \frac{-a'}{c' - a'b'} \\ 0 & 0 & 1 & b' & \frac{-a}{c - ab} \\ 0 & 0 & 0 & 1 & \frac{aa'}{(c - ab)(c' - a'b')} \end{array} \right\| \end{aligned} \tag{21}$$

Заметим, что ранг матрицы системы равен рангу расширенной матрицы системы, а значит, по теореме Кронекера-Капелли система уравнений (20) совместна. Отсюда следует, что найдутся такие C_j, $j \in \{1, ..., 4\}$, что выполнено равенство (19), а значит mm' выражается линейной комбинацией α_j, $j \in \{1, ..., 4\}$.

4. Для восстановления открытого текста будем использовать свойство билинейности спаривания e. Сперва заметим, что (вновь используем аддитивную нотацию):

$$\begin{aligned} w_1^{cc'} &= e\left((m + ra)g, (m' + r'a')h\right)^{cc'} = e\left((mc + rac)g, (m'c' + r'a'c')h\right) \\ w_4^{aa'} &= e\left((mb + rc)g, (m'b' + r'c')h\right)^{aa'} = e\left((mab + rac)g, (m'a'b' + r'a'c')h\right) \end{aligned} \tag{22}$$

При этом в силу того, что $e(a, b) = e(a, b)^{(-1)(-1)} = e(-a, -b)$, верно следующее:

$$w_4^{aa'} = e\left(-(mab + rac)g, -(m'a'b' + r'a'c')h\right) \tag{23}$$

Вновь воспользуемся свойством билинейности e, а также (22) и (23):

$$w_1^{cc'} w_4^{aa'} = e\left((m(c-ab)g, m'(c'-a'b')h\right) = e(g,h)^{mm'(c-ab)(c'-a'b')}, \tag{24}$$

где $e(g,h)$ — генератор группы $\mathbb{G}_T$.

Так как m и m' лежат в малых интервалах, то можно перебором их произведение и найти.

Приложение А. Исходный код для задачи 3.3

```
def get_max_2_power(x):
    power = 0

    while x % 2 == 0:
        x //= 2
        power += 1

    return 2 ** power


def is_power_of_two(x):
    result = False

    while x % 2 == 0:
        x //= 2

    return x == 1


def get_path(k):
    long_hops  = 0
    short_hops = 0
    path       = []

    position   = k - 1

    while position != 0:
        path.append(position)

        if position % 2 != 0 or is_power_of_two(position):
            position -= 1
            short_hops += 1
        else:
            position -= get_max_2_power(position)
            long_hops += 1

    return long_hops, short_hops, path
```