NuCypher: Сеть прокси ре-шифрования для обеспечения конфиденциальности в децентрализованных системах

Michael Egorov,[*] David Nuñez,[†] и MacLane Wilkison[‡]
NuCypher

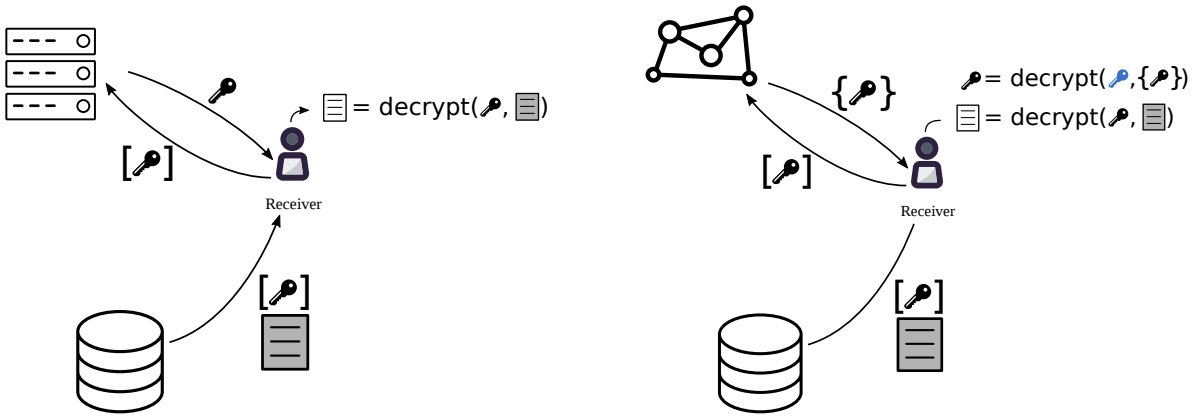
Перевод выполнен: Шадов Сергей[§] и Игонина Наталия[¶]

(Дата: 1 сентября 2018 г.)

NuCypher - это децентрализованная система управления ключами (KMS), которая устраняет ограничения использования консенсусных сетей для безопасного хранения и управления приватными зашифрованными данными [1]. Она обеспечивает шифрование и управление доступом на основе криптоалгоритмов, выполняемое децентрализованной сетью, используя прокси ре-шифрование [2]. В отличие от централизованных KMS, она не требует доверия поставщику услуг. NuCypher позволяет обмениваться конфиденциальными данными как для децентрализованных, так и для централизованных приложений, обеспечивая инфраструктуру безопасности для приложений от здравоохранения и управления идентификацией до децентрализованных площадок по продаже контента. NuCypher будет неотъемлемой частью децентрализованных приложений, так же как SSL/TLS является неотъемлемой частью каждого безопасного веб-приложения.

[*] michael@nucypher.com
[†] david@nucypher.com
[‡] maclane@nucypher.com
[§] shadovserg@gmail.com
[¶] nataligonina@gmail.com

Содержание



(a) Централизованная KMS (b) KMS на основе PRE

Рис. 1: Различие между централизованной системой управления ключами (KMS) и системой, использующей прокси ре-шифрование (PRE)

I. ВВЕДЕНИЕ

NuCypher - децентрализованная система управления ключами (KMS), шифрования и служба контроля доступа. Она позволит обмениваться частными данными между произвольным числом участников в открытой консенсусной сети (блокчейне), используя прокси ре-шифрование (повторное шифрование) для делегирования прав на расшифрование таким способом, которого невозможно достичь ни с помощью симметричных криптоалгоритмов, ни с помощью схем с открытом ключом. Токены используются для поощрения участников сети, которые управляют ключами и делегируют/отменяют доступ.

A. Общие сведения

Система управления ключами (KMS) представляет собой комплексный подход для создания, распространения и управления криптографическими ключами для устройств и приложений (Рис. 1). KMS на бэкэнде реализует создание, распространение и смену ключей, а на клиенте – внедрение, хранение и управление ключами на устройствах. [3].

Самое важное, чтобы KMS надлежащим образом управлялась, была настроена и защищена. На практике, это означает размещение KMS в аппаратных модулях безопасности (HSM) [4] или использование утилит подобных Vault от HashiCorp [5]. Однако, это требует повышенной технической подготовки, а также предварительных капиталовложений. Для облегчения технических сложностей и предложения более конкурентоспособных цен, такие поставщики, как Amazon CloudHSM [6], Google Cloud KMS [7], Azure Key Vault [8] и TrueVault [9] начали предоставлять KMS как услугу. Однако, подобный подход требует установления чрезмерного уровня доверия к поставщику услуг, что может быть неприемлемо для критически важных приложений.

Публичные блокчейны, такие как Bitcoin и Ethereum, являются перспективным решением этой проблемы централизации. Но ограничения публичных блокчейнов при выполнении криптографических операций, связанных с управлением секретными данными, хорошо известны [1]. Блокчейн использует сеть узлов, которая подвержена постоянной текучке и не так надежна, как центральная инфраструктура, когда дело доходит до

доступности и соблюдения правил управления доступом.

NuCypher использует децентрализованную сеть для устранения зависимости от центральных поставщиков услуг, прокси ре-шифрование для защищенного управления доступом и механизм стимулирования токенами для обеспечения надежности, доступности и корректности. Благодаря использованию прокси ре-шифрования, зашифрованный симметричный ключ (который дает возможность расшифровать частные данные) никогда не передается на «сервер» (Рис. 1), и нет единой точки отказа безопасности. Даже если кто-то будет скомпрометирован, хакеры получат только ключи ре-шифрования, но доступ к файлу по-прежнему останется защищен.

II. АРХИТЕКТУРА

A. Криптографические основы

1. Симметричное шифрование

Симметричное шифрование подразумевает, что пользователи знают общий секретный ключ. Для удобства мы называем этот общий секретный ключ DEK (ключ шифрования данных).

Для симметричных шифров определены две операции:

$$c = \text{encrypt}_{sym}(dek, d); \tag{1}$$

$$d = \text{decrypt}_{sym}(dek, c); \tag{2}$$

где d – это открытый текст, а c – шифрованный текст.

Наиболее полезными алгоритмами симметричного шифрования для наших целей являются AES (поскольку обычно это аппаратное ускорение) [10] и Salsa20 [11].

Симметричные блочные шифры могут работать в различных режимах. Мы используем режимы работы, которые позволяют вероятностное шифрование (например, GCM в AES), чтобы гарантировать сильную безопасность. Для простоты мы опускаем подробности о конкретных режимах работы, рассматривая значение nonce, связанное с режимом работы, как часть шифрованного текста c.

2. Шифрование с открытым ключом

Шифрование с открытым ключом (PKE) – это вид шифрования, при котором два участника (отправитель и получатель) обмениваются информацией без необходимости иметь общий ключ. Каждый участник имеет ключевую пару (открытый ключ pk и секретный ключ sk). Пусть ключевая пара отправителя sk_s/pk_s и ключевая пара получателя sk_r/pk_r, тогда отправитель может зашифровать сообщение с помощью открытого ключа получателя, а получатель может расшифровать его своим закрытым ключом.

Гибридная криптосистема может быть создана путем объединения эффективности симметричного шифрования с удобством PKE. Гибридное шифрование определяется следующими функциями:

$$dek = \text{random}(); \tag{3}$$

$$c = \text{encrypt}_{sym}(dek, d); \tag{4}$$

$$edek = \text{encrypt}_{pke}(pk_r, dek). \tag{5}$$

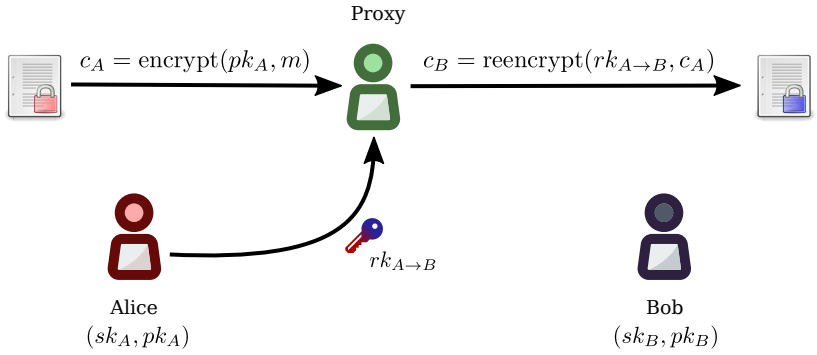


Рис. 2: Основные действующие лица в PRE

Пара $(edek, c)$ используется для преобразования шифрованных данных. Расшифрование получатель производит следующим образом:

$$dek = \text{decrypt}_{pke}(sk_r, edek); \tag{6}$$

$$d = \text{decrypt}_{sym}(dek, c). \tag{7}$$

3. Прокси ре-шифрование

Прокси ре-шифрование (PRE) [2, 12] – это вид шифрования с открытым ключом (PKE), который позволяет прокси-серверу заменить один открытый ключ на другой, не узнавая ничего о первоначальном сообщении (Рис. 2).

В стандартном процессе прокси ре-шифрования участвуют несколько сторон. Алиса, владелец данных, имеет открытый ключ pk_A. Каждый, кто знает этот ключ может зашифровать данные, но лишь она сможет их расшифровать, используя секретный ключ sk_A. Предположим, что владелец данных шифрует сообщение m, с открытым ключом Алисы pk_A, в результате чего получается шифрованный текст c_A. Алиса решает делегировать доступ к сообщению m Бобу, у которого есть пара ключей (pk_B, sk_B). Для этого Алиса создает ключ ре-шифрования:

$$rk_{A\to B} = \text{rekey}(sk_A, pk_B). \tag{8}$$

Важно отметить, что в одноразовых однонаправленных схемах прокси ре-шифрования данная функция ре-шифрования однонаправленная, а $rk_{A\to B}$ нельзя разложить на составные части (по крайней мере, без знания sk_A или sk_B). Все, что можно сделать, это ре-шифровать c_A так, чтобы он был преобразован в c_B:

$$c_B = \text{reencrypt}(rk_{A\to B}, c_A). \tag{9}$$

Боб теперь сможет расшифровать c_B, используя его секретный ключ sk_B.

По сравнению с существующими протоколами PKE, которые идеально подходят для связи 1-к-1, PRE более пригодна для связи N-к-N с произвольным числом пользователей данных. Для этого не требуется знать получателя сообщения заранее, так как ключ ре-шифрования может быть создан и применен в любой момент. Это делает данный подход идеальным для распределенных систем, таких как блокчейн, IoT и большие данные [13].

Существует множество алгоритмов прокси ре-шифрования с различными свойствами. Для первой версии NuCypher мы выбрали ECIES, для которого создали алгоритм прокси ре-шифрования [14]. Он очень похож

на самый простой (и производительный) алгоритм BBS98[15], но обеспечивает лучшие гарантии безопасности. Однако, иногда требуется делегировать ре-шифрование нескольким узлам, чтобы разделить доверие между ними и применить политики ре-шифрования на основе времени или условий. В этом случае используется схема AFGH [16]. Квантово-устойчивый NTRU может также использоваться при необходимости [17, 18].

Так же, как и BBS98, наш алгоритм прокси ре-шифрования ECIES создает ключ ре-шифрования из двух секретных ключей, а не секретного и открытого ключей. Однако, мы не хотим, чтобы отправитель знал секретный ключ получателя. Чтобы обойти эту проблему, мы случайным образом генерируем пару ключей sk_e/pk_e. Затем делегирование доступа выглядит следующим образом:

$$sk_e = \text{random}(); \tag{10}$$

$$rk_{A\to e} = \text{rekey}(sk_A, sk_e); \tag{11}$$

$$sk'_e = \text{encrypt}_{pke}(pk_B, sk_e); \tag{12}$$

$$rk_{A\to B} = (rk_{A\to e}, sk'_e); \tag{13}$$

Узел ре-шифрования использует $rk_{A\to e}$ для повторного шифрования любого шифрованного текста c_A (первоначальным сообщением является m), поэтому его можно расшифровать с помощью sk_e. Поскольку получателю требуется sk'_e для расшифровки конечного сообщения, он привязан к результату ре-шифрования. Следовательно, процесс ре-шифрования выглядит следующим образом:

$$c_e = \text{reencrypt}(rk_{A\to e}, c_A); \tag{14}$$

$$c_B = (c_e, sk'_e); \tag{15}$$

$$\tag{16}$$

Затем получатель расшифровывает первоначальное сообщение:

$$sk_e = \text{decrypt}_{pke}(sk_B, sk'_e); \tag{17}$$

$$m = \text{decrypt}_{pke}(sk_e, c_e); \tag{18}$$

$$\tag{19}$$

Этот подход был упомянут Ateniese et al. [16], а также в работе о прокси ре-шифрованию. Эта схема не считается «оптимальной», однако, в нашем случае она имеет очень конкурентоспособную производительность.

B. Краткий обзор схем ре-шифрования

Схемы ре-шифрования имеют различные свойства. В этом подразделе мы рассмотрим несколько из них, которые используются нами. Недавно было опубликовано полное исследование с учетом всех известных алгоритмов прокси ре-шифрования и их свойств [12].

Одним из важных свойств, которое следует учитывать, является ли алгоритм интерактивным. «Интерактивный» означает, что ключ ре-шифрования вычисляется из двух секретных ключей:

$$re_{ab} = \text{rekey}(sk_a, sk_b). \tag{20}$$

«Неинтерактивный» означает, что ключ ре-шифрования вычисляется из секретного ключа владельца и открытого ключа получателя:

$$re_{ab} = \text{rekey}(sk_a, pk_b). \tag{21}$$

Примерами интерактивных алгоритмов являются: BBS98 [15], наш ECIES (на основе BBS98) и ре-шифрование на основе LWE [19]. Примером «неинтерактивного» алгоритма является AFGH [16]. Несмотря на то, что изначально кажется, что нас будут интересовать неинтерактивные алгоритмы, мы можем настроить интерактивные алгоритмы на уровне протокола для совместного использования с публичным, а не с закрытым ключом, используя эфемерные ключи (Рав. 10-12).

Другое интересное свойство алгоритма - является ли он однонаправленным или двунаправленным. Двунаправленность означает, что можно вычислить re_{ba} с помощью re_{ab}. В однонаправленных алгоритмах это невозможно. Двунаправленные алгоритмы эффективнее однонаправленых при использовании эфемерных ключей (Рав. 10-12). Тем не менее, в Гл. V C мы показываем, что однонаправленность может быть очень удобной для масштабирования сложных иерархических данных. Примером двунаправленного алгоритма является BBS98 [15].

Алгоритмы прокси ре-шифрования также различаются по количеству посредников и могут быть с одним (single-hop) или несколькими (multi-hop) посредниками. «Multi-hop» означает, что если у нас есть re_{ab} и re_{bc}, эти два ключа ре-шифрования могут применяться последовательно для преобразования шифрованного текста c_a в шифрованный текст c_c без участия b:

$$c_c = \text{reencrypt}(re_{bc}, \text{reencrypt}(re_{ab}, c_a)). \tag{22}$$

Иногда, даже возможно вычислить re_{ac}, например, для BBS98:

$$re_{ac} = re_{ab} \cdot re_{bc}. \tag{23}$$

«Single-hop» означает, что если c_b получается посредством ре-шифрования, невозможно перешифровать его дальше. Multi-hop схемы полезны для смены ключей (Гл. V G) и для нашей иерархической схемы обмена данными (Гл. V C). Подход с эфемерным ключом (Рав. 10-12) это, действительно, single-hop (хотя смена ключа по-прежнему возможна, потому что эфемерные ключи создаются только тогда, когда данные передаются другим сторонам). Примерами multi-hop алгоритмов являются BBS98 [15] и алгоритм на основе LWE [19]. Алгоритм AFGH [16] является single-hop.

Еще одним, казалось бы, важным свойством является стойкость к коллизиям. На самом деле, стойкость к коллизиям означает, что имея re_{ab} и sk_b невозможно получить sk_a. И наоборот, отсутствие стойкости к коллизиям позволяет получить sk_a из re_{ab} и sk_b. На первый взгляд это кажется очень важным для обеспечения безопасности. Однако, даже для алгоритмов стойких к коллизиям, если злоумышленник имеет re_{ab} и sk_b, он вполне способен решифровать и читать любые данные, которые должны расшифровываться с помощью sk_a. Таким образом, стойкость к коллизиям становится важным только тогда, когда одна и та же пара ключей используется в различных целях (например, подпись, деривация ключа и т. д.). Тем не менее, рекомендуется использовать отдельные пары ключей для обеих функций. Для наших целей стойкость к коллизиям не является приоритетом, потому что у нас нет другого целевого использования пары ключей, кроме шифрования. Алгоритмами, стойкими к коллизиям, являются AFGH [16] и ре-шифрование на основе LWE [19]. Алгоритмы, не стойкие к коллизиям - BBS98 [15] и ECIES.

Возможно, потребуется сделать невозможной идентификацию владельцев и/или потребителей данных с помощью ключей ре-шифрования (Гл. III B). Часто прокси-сервер, который знает все открытые ключи в системе, может получить эту информацию. Многие PRE схемы не являются анонимными с точки зрения ре-шифрования [15, 16]. Однако, основанная на LWE схема ре-шифрования [19] обладает свойством анонимности.

Наконец, как и для любой криптосистемы, понятие CPA-безопасности [20] (защита от атак на основе подобранного открытого текста) и понятие CCA-безопасности [21] (защита от атак на основе подобранного шифрованного текста) применимы к прокси ре-шифрованию. Все алгоритмы, которые мы упомянули до сих пор, являются только CPA-безопасными, за исключением ECIES и LWE, которые также являются CCA-безопасными [19].

C. Подписание шифрованных сообщений

В алгоритмах шифрования с открытым ключом любой может зашифровать с помощью pub_a. Это очень удобно, но также позволяет злоумышленникам шифровать данные, как будто они A. Таким образом, данные должны быть подписаны, чтобы доказать идентичность отправителя получателю.

Тем не менее, мы хотим сделать возможным анонимизацию протокола (Гл. III B), потому что публичная цифровая подпись, которая аутентифицирует владельца данных, также повышает вероятность того, что узел ре-шифрования попытается вымогать деньги у владельца. Таким образом, имеет смысл:

- включить цифровую подпись в открытый текст так, чтобы было невозможно ее проверить, пока текст не расшифруют,
- использовать различные пары ключей для подписи и шифрования (особенно учитывая отсутствие стойкости к коллизиям некоторых криптосистем ре-шифрования).

D. Узлы ре-шифрования

Когда данные хранятся в облаке или децентрализованном хранилище, они зашифрованы ключом владельца (отправителя) pk_s (Рис. 3). Сами данные шифруются случайным симметричным ключом dek, один ключ на каждый файл. Ключ dek, зашифрованный с помощью pk_s прикрепляется к зашифрованным данным. Эта пара $(edek, c)$ может храниться где угодно - в IPFS, Swarm, S3 или любом децентрализованном или централизованном хранилище.

При хранении данных пользователь, которому мы делегируем доступ, не обязательно известен заранее. Во-первых, получатель должен показать отправителю свой открытый ключ (Рис. 4). Часто имеет смысл, чтобы открытый ключ соответствовал адресу в сети Ethereum (например, чтобы доказать, что с этого адреса был произведен платеж за подписку на цифровой контент). Отправитель генерирует ключ ре-шифрования $re_{s \to r}$ (включая зашифрованный случайный эфемерный ключ, когда это необходимо) и отправляет его на случайный узел ре-шифрования, выбранный в соответствии с его долей (принцип proof-of-stake) из активных узлов в децентрализованной сети. Случай, когда несколько узлов выбраны для избыточности или безопасности, будет обсужден позже. Узлы, которые используют данные отправителя, регистрируют эту информацию в сети.

Когда получатель хочет расшифровать данные, которыми поделились с ним, он сначала выгружает эти данные из хранилища (Рис. 5). Он извлекает $edek$ из сообщения и отправляет $edek$ в сеть узлов ре-шифрования и находит активные узлы, которые могут обмениваться данными отправителя с получателем (те, у которых есть

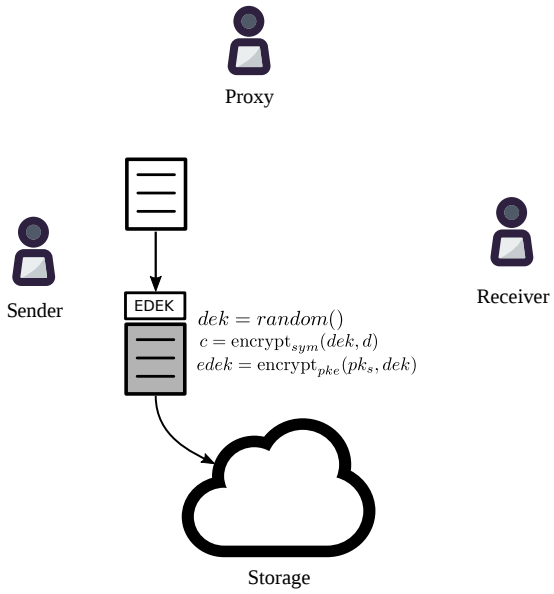


Рис. 3: Архитектура: шифрование

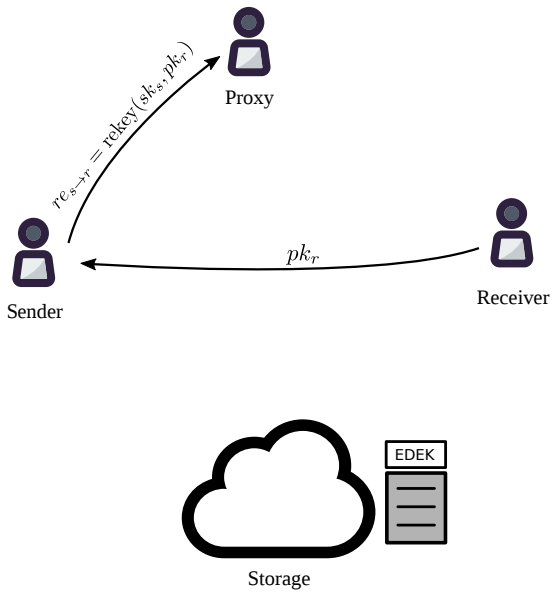


Рис. 4: Архитектура: делегирование доступа

ключ(и) ре-шифрования $re_{s\to r}$). Получатель запрашивает узел (узлы) с ключом ре-шифрования для преобразования $edek$ в $edek'$ и использует свой собственный секретный ключ sk_r для расшифровки и получения DEK. Теперь он может использовать DEK для расшифровки данных.

III. БЕЗОПАСНОСТЬ СЕТИ

В сети существует несколько узлов ре-шифрования, которые применяют политики управления доступом.

Прокси ре-шифрование позволяет NuCypher разделить доверие между управлением доступом и правами расшифрования, не вводя абсолютно доверенную сущность (как в традиционной системе управления ключами).

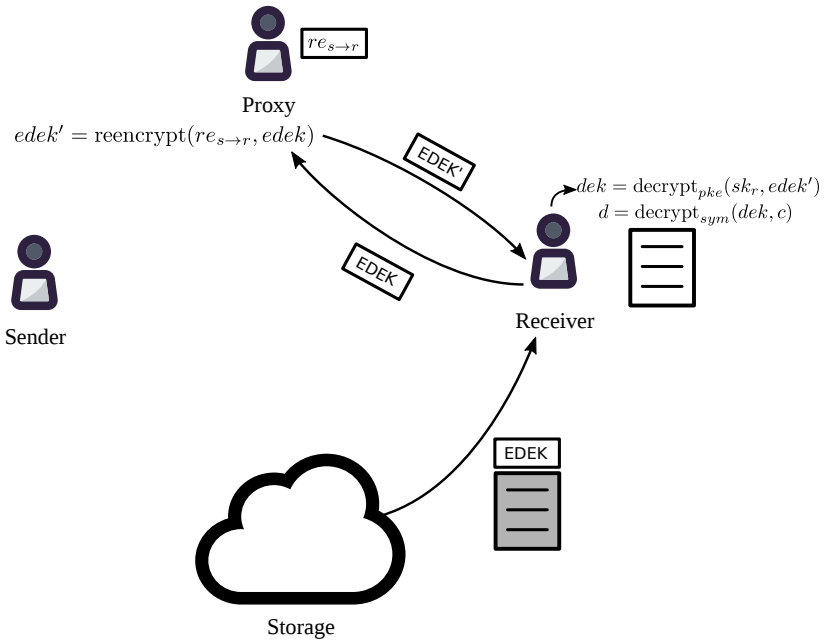


Рис. 5: Архитектура: расшифрование

Майнеры никогда не видят открытых данных или чего-то, что позволяет им расшифровать данные. Они несут полную ответственность за хранение ключей ре-шифрования и применение функций ре-шифрования.

Первый риск этой модели - сговор между майнером и получателем данных. Если майнер предоставляет получателю ключи ре-шифрования данных, данные могут быть расшифрованы в любое время, обходя любые условные или временные ограничения. Мы противодействуем этой угрозе несколькими способами: используем схему ре-шифрования с разделенным ключом Umbral [14] для децентрализации доверия между несколькими майнерами, позволяя получателям данных доказать правильность или некорректность ре- шифрования в другой сети, включив протокол проверки в Umbral; делаем протокол псевдо-анонимным. Кроме того, мы применяем экономические стимулы для справедливой работы, описанные в Гл. VII.

Второй риск - неисправность узлов (возврат поддельных данных вместо перешифрованных). Мы решаем эту проблему, используя протокол вызова.

Третий риск – это возможность узлов вступать в сговор друг с другом для выполнения атаки 50%. Это обычно чревато для многопотоковых вычислений [22] (таких как Enigma [23]). Однако, в нашем случае злоумышленник получает только возможность неправомерно применять политики ре-шифрования, не расшифровывая данные и не предоставляя доступ неправомерным пользователям. В идеале, система должна быть максимально децентрализована, однако атака 50% не ставит под угрозу конфиденциальность данных, так же, как атака 50% в POW криптовалютах не дает злоумышленнику возможность перемещать средства.

A. Разделение ключа ре-шифрования

Представьте, что узел ре-шифрования решает немедленно перешифровать данные, а не применять условные политики в соответствии с инструкциями. Для решения этой проблемы можно использовать схему прокси ре-шифрования с разделением ключа.

Вместо одного ключа ре-шифрования могут быть использованы m-of-m ключей ре-шифрования для созда-

ния "общих ресурсов". Для шифрования AFGH [16] существует схема m-of-m. Для атаки на данную схему потребуется m майнеров и получатель данных. Однако, AFGH-схема уязвима к DOS атаке.

Пороговая схема m-of-n (Umbral), которую мы разрабатываем вместе с NICS Labs из University of Málaga кажется еще более подходящей для этой задачи [14]. Эта схема также может позволить третьей стороне проверить правильность ре-шифрования, что важно для обеспечения честности узлов. Значения m и n могут быть выбраны клиентом на основе компромисса между производительностью и безопасностью, в то время как узлы ре-шифрования не применяют никаких конкретных значений m и n.

B. Псевдоанонимность

Для безопасности системы очень полезно, что узлы ре-шифрования не знают, что они перешифруют. Это не позволяет им выполнения атак по сговору (попытка сговора со всеми участниками сети невозможна, когда сеть децентрализована).

Наш протокол изначально является псевдоанонимным, т. е. не хранит личности участников. Мы постараемся спроектировать полностью анонимный протокол ре-шифрования в будущем. Однако, отметим следующие его свойства, которыми он должен (и не должен) обладать. Во-первых, схема ре-шифрования должна быть ключ-приватной [19, 24]. В противном случае, можно будет определить владельца ключа перебором всех известных пар открытых ключей. Во-вторых, узел ре-шифрования и получатель данных не должны иметь один и тот же идентификатор для одного и того же ключа. Это исключает простой способ хранения ключа шифрования в хранилище ключ-значение, когда получатель может придумать свой ключ.

C. Протокол вызова

Существует риск того, что майнеры возвратят случайную последовательность вместо корректных шифрованных данных. Поскольку данные являются приватными, пользователи системы не могут публиковать эти данные и их ключ в качестве доказательства того, что майнер совершил подлог.

Майнер не может различить “истинное ре-шифрование” и ре-шифрование случайных данных. Таким образом, мы можем создать несколько “поддельных” ключей ре-шифрования, которые разработаны специально для того, чтобы проверить майнеров. Если майнер жульничает, данные и ключ для этой проверки не будут связаны с личными данными.

Майнеры должны показывать хэши данных до и после ре-шифрования. Если перешифрование было некорректным, и майнер обманул, пользователи могут представить доказательство того, что ключи, связанные с ними, должны фактически привести к другому результату ре-шифрования, и залоговый депозит майнера может быть присужден пользователю.

Система также должна преднамеренно производить ряд “неправильных ре-шифрований”, чтобы стимулировать пользователей к работе, как указано в Truebit [25].

Разработка протокола вызова является сложной проблемой, связанной с протоколами “справедливого обмена” [26–28]. Это требует тщательного проектирования и тестирования, и протокол POS Ethereum (Casper) сталкивается с подобной проблемой сейчас. Можно просто проверить корректность с помощью алгоритма шифрования [29].

Особое внимание следует уделить защите ключей ре-шифрования от утечки. Предлагается следующий про-

токол вызова.

Принимая на себя ответственность за ре-шифрование ключа, майнер рассчитывает получить комиссию f в течение времени T, поэтому владелец данных депонирует f монет. Майнер должен также оставить залог c, который будет конфискован, если произойдёт утечка ключа ре-шифрования.

Если пользователь докажет, что майнер скомпрометировал ключ ре-шифрования, пользователь должен быть вознагражден. Тем не менее, владелец данных может проверить майнера, чтобы обманным путем получить награду за вызов. Мы делаем этот "вызов самому себе" невыполнимым. Если вызов произошел после времени t, то претендент получит $\alpha ft/T$ монет, где $\alpha < 1$. Владелец данных в этом случае получает $(1 - t/T)f$ монет. Залоговое обеспечение и остальная часть гонорара изымаются в пользу других участников сети, с общей суммой $c + (1 - \alpha)t/T$.

Также не должно быть никакого стимула для владельца данных фальсифицировать - проверить майнера вместо отмены политики. Таким образом, при "правильном" отзыве владелец данных получает $(1-t/T)f$ монет назад, а майнер получает $c + ft/T$ монеты, где c - залог, который был поставлен.

D. Актуальность возможных угроз для различных вариантов использования

В случае управления мобильными устройствами (Гл. VIII I), самое главное, чтобы можно было отозвать доступ для потерянного или украденного устройства до того, как данные будут скомпрометированы. Представьте себе возможную атаку, когда кто-то крадет устройство и вступает в сговор с соответствующими майнерами. Таким образом, майнеры не должны идентифицировать пользователя, и наоборот. Другая возможная атака-это группа майнеров, которые отказывают в доступе и требуют дополнительную плату за ре-шифрование. Однако, это не актуально, так как владелец данных может легко повторно предоставить доступ этому мобильному устройству. Еще одна возможная угроза - майнинговый узел, хранящий ключи ре-шифрования в течение длительного времени после срока действия политики, ожидая, пока кто-то атакует устройство конечного пользователя и вступит с ним в сговор. Для предотвращения этой угрозы важно, чтобы майнинговый узел не мог определить, являются ли данные ценными или нет, и хороший способ сделать это-анонимизировать владельца данных и сами данные. Другими словами, мы предотвращаем обнаружение майнинговым узлом данных, которые соответствуют *edek*.

Децентрализованный DRM (Гл. VIII D) предполагает, что расшифрованный контент (файл или фрагмент видео) был куплен, поэтому отзыв доступа на самом деле не является проблемой. Однако, если узел знает, что контент имеет высокую стоимость, он может попытаться предложить покупателю более низкую цену, в обход первоначального продавца. Чтобы предотвратить это, мы должны анонимизировать получателя данных. Это также поможет скрыть точную информацию о ценах от майнингового узла, позволяя в то же время ему проверить, была ли выплачена необходимая сумма. Это достигается с помощью zk-SNARKs [30].

Когда NuCypher используется для защиты доступа к файлам (Гл. VIII A) или сообщениям, как предоставление, так и аннулирование доступа играет важную роль. Поэтому полная анонимизация крайне желательна. Возможные атаки включают получателей данных, которые могут подкупить майнеров, чтобы продолжить иметь доступ после того, как он был отозван; и майнеры, которые вымогают плату у владельца, когда крайне важно отменить доступ. Анонимизация, по-видимому, является важной частью того, чтобы сделать такие атаки неосуществимыми.

E. Аппаратная защита

Если майнеры ведут себя некорректно, они рискуют потерять свой залоговый депозит. Однако, вместо целенаправленного злого умысла, майнерские узлы могут стать жертвой атаки третьей стороны. Для того, чтобы майнеры могли предотвратить компрометацию своих узлов, они могут использовать доверенные вычисления на общедоступном безопасном оборудовании [31], таком как процессоры Intel последнего поколения (Skylake+), которые поддерживают графические процессоры SGX или NVidia.

Технология Intel SGX [32] обещает выполнять любые вычисления в безопасной среде. Ранее предлагалось создать децентрализованную сеть для управления секретами, основанную на технологии SGX, а не на справедливости майнеров или ре-шифровании [33].

В качестве альтернативы, хранение и применение ключей ре-шифрования может быть сделано внутри графических процессоров. Было показано, что графические процессоры могут служить доверенными платформенными модулями с ограниченными возможностями (хотя они, безусловно, могут выполнять ре-шифрование) [34].

IV. АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ

V. ФУНКЦИОНАЛЬНОСТЬ

A. Локальная библиотека шифрования и демон

NuCypher можно связать с традиционным централизованным приложением. В Python совместное использование файла в IPFS будет выглядеть так:

```
import nucypher
nucypher.connect() # Using default config
path = 'ipfs://QmTkzDw.../to_the_moon.avi'
nucypher.share('0xab12...', path, time=86400)
```

Если клиент библиотеки недоступен, может существовать локальный API сервер, аналогичный тому, как geth используется для взаимодействия с сетью Ethereum.

Чтение этого файла будет:

```
import nucypher
client = ipfsapi.Client(...)
nucypher.connect() # Using default config
path = 'ipfs://QmTkzDw.../to_the_moon.avi'
edata = client.cat(path)
data = nucypher.decrypt(edata, path)
print(data)
```

Функция “decrypt” разделяет edata на *edek* и фактические зашифрованные данные, запрашивает сеть для преобразования *edek* в $edek'$, расшифровывает $edek'$ с помощью закрытого ключа получателя и расшифровывает данные полученным *dek*.

Предварительная версия API будет включать следующие функции:

- connect — подключиться к децентрализованной сети, взяв конфигурацию из аргументов или конфигурационного файла;
- write — шифрование данных открытым ключом, соответствующим владельцу файла, и сохранение в серверной части хранилища;
- read — загрузка данных из хранилища, запрос сети на ре-шифрование и расшифровка с помощью нашего закрытого ключа;
- delete — удалить файл и связанные с ним ключи ре-шифрования;
- decrypt — расшифровать данные, которые уже прочитаны;
- split-edek — низкоуровневая функция для отделения зашифрованного симметричного ключа шифрования от данных;
- share/renew/revoke — создать разрешение на чтение всех данных, которыми мы владеем, или их частью, на основе пути к файлу или политики. Политика может включать ограничения по времени и другие условия;
- read-policies/update-policies/delete-policies — чтение и изменение всех политик доступа, которые были созданы.

B. Совместное использование маленьких данных

Функциональность системы позволяет шифровать и делегировать доступ к двоичным данным (таким как учетки баз данных) или группам этих данных, не сохраняя их в отдельных файлах. Возможным бэкэндом для хранения таких простых данных может быть конфигурационный файл (в формате YAML или JSON), хранящийся в IPFS, или даже просто в том же образе экземпляра. Клиентское программное обеспечение может проанализировать этот файл и попросить повторно зашифровать только необходимые ему данные. Для более детальных разрешений могут быть созданы ключи ре-шифрования для каждого поля и для каждого подполя.

C. Совместное использование файлов и иерархических данных

При работе с файлами каждый файл и каждая папка шифруется своим секретным ключом. Каждый файл имеет свой ключ DEK, зашифрованный его секретным ключом, секретным ключом папки и так далее, вплоть до корневой папки пользователя. Если папка используется совместно, ключ ре-шифрования создается только для общих папок.

Для вышеуказанного метода требуется хранить столько Edek, сколько уровней в древовидной структуре. Если мы используем multi-hop, однонаправленный алгоритм (такой как LWE [19]), мы можем использовать ключи ре-шифрования от нижнего до верхнего уровня (Рис. 6). При предоставлении доступа к папке, все ее файлы и подкаталоги зашифрованы с помощью ключа папки, а затем перешифрованы для получателя:

$$edek_b = \text{reencrypt}(re_{xb}, \text{reencrypt}(re_{1x}, edek_1)). \tag{24}$$

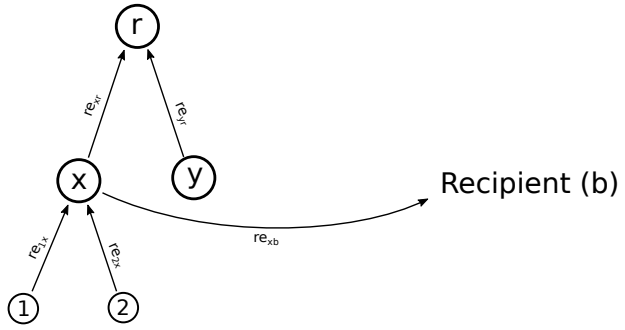


Рис. 6: Совместное использование иерархических структур (таких как файлы и папки)

D. Шифрование данных большого размера

Когда у нас есть симметричный ключ шифрования для каждого файла, злоумышленник может загрузить все зашифрованные симметричные ключи и расшифровать их, пока у него есть доступ, без необходимости загружать слишком много данных. Впоследствии он сможет использовать эти симметричные ключи, чтобы расшифровать данные. Это называется “атакой набора ключей”.

В приложениях, где присутствует эта проблема, мы можем использовать подход “все или ничего” к данным, так что нужно загрузить весь файл, чтобы расшифровать его. Этот подход был опубликован [35] и легко доступен для использования в нашей системе управления ключами.

E. Совместное использование зашифрованных потоковых данных

Совместное использование зашифрованных потоков с несколькими пользователями идеально подходит для таких приложений, как децентрализованный Netflix, где третьей стороне, направляющей трафик, не разрешается просматривать содержимое.

Принцип прост: каждый блок данных в потоке шифруется случайным ключом DEK, а EDEK создается с использованием открытого ключа для каждого канала. Однако, требуется время, чтобы сделать обращение к NuCypher. Поэтому имеет смысл также включить EDEK следующего блока данных вместе с предыдущим блоком.

Кроме того, для случая использования потокового мультимедийного контента особенно проблематична атака набора ключей. Таким образом, мультимедиа файлы следует рассматривать как данные большого размера и шифровать соответственно [35].

Как обычно, получатели потоковых данных имеют свои пары ключей и запрашивают у узлов EDEK. Майнеры также могут гарантировать, что потребитель фактически заплатил за подписку и отказаться от ре-шифрования, в обратном случае.

F. Политики, основанные на времени и условиях

Хотя мы не доверяем майнерам в отношении зашифрованных данных или ключей, мы по-прежнему доверяем им контролировать продолжительность хранения ключа ре-шифрования. Простейшая политика основная на времени: ре-шифрование разрешено только в течение указанного интервала времени, и по его окончании ключ

ре-шифрования должен быть уничтожен.

Можно создать более сложные политики. Ре-шифрование может быть разрешено, например, при условии завершения определенной транзакции. Это необходимо приложениям, таким как DRM с оплатой за контент, или предоставляющим услуги по депонированию секретных данных.

G. Смена ключей

Многие алгоритмы прокси ре-шифрования могут быть применены несколько раз, в том числе BBS98 [15]. Таким образом, для смены ключей можно использовать прокси ре-шифрование. Смена ключей позволяет всем EDEK, зашифрованным старым ключом, перешифроваться новым.

Владельцу данных необходимо создать ключи ре-шифрования $re_{v1\to v2}$ между двумя секретными ключами. Теперь узлы ре-шифрования (при условии, что зашифрованное хранилище является общедоступным, например IPFS) будут загружать EDEK и применять преобразование. Для этой операции не существует риска сговора, поскольку “отправитель” и “получатель” данных являются, по сути, одним и тем же лицом.

VI. ПРОЕКТИРОВАНИЕ СМАРТ КОНТРАКТОВ

Для предоставления услуг ре-шифрования узлу необходимо отправить депозит на смарт контракт (указав при этом время блокировки). По истечении времени узел может вывести его. Цель-получить вознаграждение за создание узлов, предоставляющих службы ре-шифрования.

При первоначальном запуске мы гарантируем, что все узлы будут находиться в сети и правильно ре-шифровать данные, не требуя анонимизации (Гл. III B). Мы ожидаем, что узлы станут профессиональными и инвестируют значительные средства в свою безопасность, подобно тому, что произошло в других сетях. Таким образом, у нас будет минимальное требование к доле $s_{\min}$ узлов для предоставления услуг ре-шифрования. Это, естественно, ограничивает число узлов, аналогично механизму формирования мастернод криптовалюты DASH [36].

Хранимое на узле количество токенов является открытым, поэтому клиенты сети могут решить для себя, следует ли пользоваться конкретным узлом или нет. Клиент (Alice) отвечает за случайное распределение ключей ре-шифрования.

Назовем активный (заблокированный) депозит i-го майнера d_i. Общая сумма заблокированных депозитов составляет:

$$D = \sum_{1}^{k} d_i, \tag{25}$$

где k общее количество майнеров. Вероятность того, что i-й майнер получит ключ ре-шифрования от одной политики:

$$p_i = n\frac{d_i}{D}, \qquad \text{в пределе} \quad \forall i : np_i \ll 1, \tag{26}$$

где n количество частей, на которые мы разделяем ключ ре-шифрования (Гл. III A).

Выбор случайных узлов для обработки ключей ре-шифрования происходит следующим образом (Рис. 7). Все узлы упорядочивается случайным образом и размещаются на отрезке длины D, начиная с координаты $x = 0$.

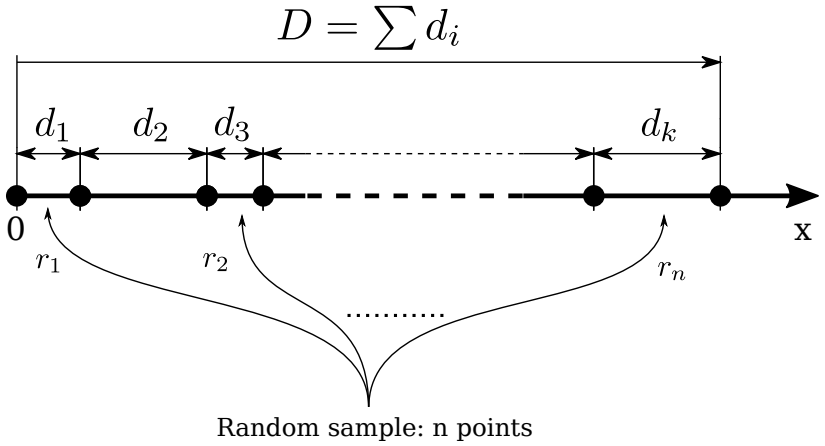


Рис. 7: Выбор случайных узлов в соответствии с POS. Доля каждого узла d_i представляется в виде отрезка. Отрезки складываются вместе в один более крупный отрезок длиной D. Узлы выбираются вбросом случайных точек, распределенных от $x = 0$ до $x = D$.

Таким образом, координата i-го узла будет:

$$x_i = \sum_{j=1}^{i-1} d_j. \tag{27}$$

При создании политики Алиса выбирает n случайных чисел:

$$\forall j = 1..n : r_j \in [0, D). \tag{28}$$

Не теряя общности, предположим, что числа r_j отсортированы. Тогда индексы выбранных майнеров i_j это просто индексы, которым соответствуют выбранные отрезки:

$$\forall i_j : x_{i_j} \leq r_j, x_{i_j+1} > r_j. \tag{29}$$

Некоторые числа могут попадать на отрезки, соответствующие одному и тому же майнеру. В этом случае Алиса удаляет повторы i_j и генерирует новые индексы i_j таким же образом, пока не будет выбрано нужное количество майнеров (n).

Назовем набор возможных тестов обобщенным “ping”. Он включает:

- фактический ping (отображается ли узел в сети?);
- проверка, работает ли старая, но активная политика ре-шифрования;
- проверка, работает ли последовательность create-reencrypt-revoke;
- проверка, работают ли вышеперечисленные политики ре-шифрования по запросу с другого адреса, не связанного с узлом.

При выполнении этого “ping”, можно выбрать только тесты, необходимые для определения доступности. Например, если узел переходит в автономный режим, фактического ping будет достаточно, чтобы определить состояние узла.

Теперь мы хотим спроектировать систему узлов, которая делает самопроверку каждые h блоков (где $h = 10$, например). Идея заключается в том, что узлы сами определяют некорректно работающие и награждаются

в зависимости от того, правильно ли их предположение или нет. “Правильным” предположением считается предположение, подтвержденное большинством узлов, вычисленное с помощью смарт контракта.

Протокол не будет хорошо масштабируем в системе с N узлами, если мы будем проверять каждый узел раз в h блоков. Таким образом, только часть узлов (например, $k = \sqrt{N}$) будут проверены. Например, если $100,000$ узлов находятся в сети и время блока Ethereum составляет 24 с (текущее значение), то работоспособность каждого узла будет проверяться примерно каждый 21 час. Выбор узлов, подлежащих проверке, может выполняться текущими проверяющими узлами, обменивающимися случайными байтами, и вычислением хэша этих случайных байтов. Узлы, которые не участвуют в проверке, не смогут отличить, проверяются ли они или получают реальный запрос клиента.

Невозможно для всех узлов выполнить проверку работоспособности. Таким образом, часть узлов (например, $\sqrt{N}$ узлов, ближайших к текущему хэшу блока по расстоянию Хэмминга) сможет получить вознаграждение за проверку работоспособности в текущем раунде. Если узлы считаются “нездоровыми”, они наказываются, теряя некоторое количество бонусных вознаграждений, полученных ими с момента последней проверки работоспособности.

Можно проверить узел на корректность ре-шифрования. Для этого владелец данных может подготовить “challenge pack” - данные, которые не соответствуют каким-либо конфиденциальным данным, но используются специально для проверки узлов ре-шифрования. “Challenge pack” состоит из текстов, продаваемых на вход, и ожидаемых выходов после ре-шифрования (этот метод работает только для алгоритмов прокси ре-шифрования, которые не имеют вероятностных выходов). Получатель данных может расшифровать пакет и продемонстрировать, что узел ре-шифрования не работает или не может правильно перешифровывать данные.

Узлы, которые проверяют другие узлы ре-шифрования, по существу голосуют за то, какие узлы некорректно себя ведут и делают ставку на результат этого голосования. Проверяющие узлы должны быть неспособны вычислить голоса других до того, пока каждый не отдаст свой голос. Таким образом, они сначала совершают голосование, демонстрируя хэши “подсоленных голосов” (голоса изменяют специальной добавкой называемой солью), а затем доказывают, что у них есть фактические голоса и “соль”, чтобы вычислить эти хэши.

VII. ТОКЕНОМИКА

Экономика протокола состоит из сети майнеров, которые способствуют работе, чтобы обеспечить дефицитный ресурс, и это вознаграждается, когда указанный ресурс потребляется. В NuCypher майнерами являются узлы ре-шифрования. Любой может стать майнером, и их награды дифференцируются в зависимости от количества операций ре-шифрования. Доступ к дефицитным службам ре-шифрования должен контролироваться и распределяться в соответствии с наивысшей ценностью. Механизм, с помощью которого мы достигаем этого, - это токен NuCypher (NU). NU является наградой для майнеров, которую они получают за свою работу, а владельцы данных платят за доступ к услугам шифрования. Особенно важно, что токен также стимулирует правильность вычислений и безопасность системы.

A. Предназначение токена

Основной целью токена является распределение доверия в системе. Количество доверия к каждому узлу пропорционально количеству токенов, которые хранит узел. Это означает, что вероятность получения узлом

ключа ре-шифрования равна отношению токенов, которые заблокированы на узле, к общему количеству заблокированных токенов. Кроме того, узел гарантирует доступность на время блокировки токенов. Токен можно рассматривать как часть майнинг оборудования, которое резервирует определенную квоту всех заданий для майнера.

Майнер все еще должен предоставить достаточно вычислительных ресурсов для выполнения этой квоты. Важно отметить, что токен должен быть труднодоступным, а это значит, что ни один майнер не может приблизиться к 50% доверия, сохраняя хорошую децентрализацию системы.

Также токен может быть использован в качестве залога против неправильного поведения майнера при работе с блокчейном, который поддерживает смарт контракты (например, Ethereum).

B. Зачем нужен отдельный токен, а не просто ETH

В принципе, можно было бы использовать другие токены вместо NU. Однако это открывает возможность для следующей атаки.

Злоумышленник может одолжить достаточно денег, чтобы купить больше ETH, чем заблокировано в настоящее время. Затем этот злоумышленник может поставить эти ETH, и тогда у него будет более 50% доверия. Это, конечно, сигнализирует о том, что система менее безопасна, и ее ценность будет падать. Но это не влияет на стоимость ETH злоумышленников, и после совершения атаки он может легко продать их за ту же цену, за которую купил, и вернуть взятые в долг деньги.

Если NU токены использовать для хранения, такая атака уменьшит стоимость токенов вместе с обесцениванием всей сети. Таким образом, если атакующий продает свою долю после совершения атаки, он получает от этого достаточно меньше средств и не может выплатить кредит, который он взял для совершения атаки.

C. Распределение токенов

Майнеры могут хранить несколько ключей ре-шифрования, пропорционально количеству токенов, которые они держат в качестве залогового депозита, заблокированного смарт контрактом. Майнерам платят как за предоставление услуг ре-шифрования, так и за предоставление доступа к ре-шифрованию данных. Майнеры могут быть оплачены либо владельцем данных, либо пользователями данных. Последнее более актуально в случае, когда DAO распространяют контент в рамках децентрализованного DRM.

Кроме того, у майнеров может быть “вознаграждение за доступ”, если они остаются в сети и готовы предоставить свои услуги, подтверждение которых проверяется с помощью смарт контракта (Гл. VI). Если майнеры становятся недоступными, они теряют компенсацию за свою “доступность”.

Если майнеры обманывают и некорректно перешифровывают, они теряют часть своего депозита (Гл. III C).

Мы можем сдерживать майнеров от утечки ключей ре-шифрования. Любой может проверить майнера с помощью хэша ключа ре-шифрования, и, если будет доказана утечка ключа ре-шифрования, майнер теряет свой залог. Можно сделать ключи ре-шифрования, связанные с определенными прокси-серверами, чтобы быстро их идентифицировать, если они были скомпрометированы [37].

Важно стимулировать операции ре-шифрования на ранней стадии, когда в системе ещё мало пользователей. Вот почему мы введем график вознаграждений, где некоторые вознаграждения “майнятся”, асимптотически приближаясь к нулевой инфляции с течением времени. Вознаграждение r_i, которое получает майнер i, может

быть подсчитано как:

$$r_i = \frac{d_i \cdot s}{\sum_{i=1}^{k} d_i} + f_i, \tag{30}$$

где s - текущая ставка вознаграждения, d_i - депозит i-го майнера, а f_i общая сумма комиссий за транзакцию для этого майнера. Ставка вознаграждения экспоненциально уменьшается таким образом, что достигает равновесия в будущем, переключаясь на плату только комиссий:

$$s = s_0 e^{-t/T}, \tag{31}$$

где T - время, по истечении которого вознаграждение падает в e раз, а s_0 - начальная ставка вознаграждения. Эту экспоненту удобно вычислять в смарт контракте без дорогостоящих вычислений с плавающей запятой как решение дифференциального уравнения:

$$s = \frac{dS}{dt} = s_0 \frac{S_{\max} - S}{S_{\max} - S(t=0)}, \tag{32}$$

где $S_{\max}$ - максимальный объем монет при $t = \infty$, $S(t=0)$ - начальный объем монет. Время t выражается в периодах майнинга, а не в секундах.

Интересным свойством NuCypher является то, что безопасность повышается по мере роста числа участников сети. Поскольку дополнительные узлы ре- шифрования входят в сеть, следовательно уменьшается вероятность сговора. Это улучшает обеспеченность и цензуроустойчивость системы, обеспечивая мощный сетевой эффект и неоспоримые преимущества проекта-новатора.

Наконец, для того, чтобы проверить безопасность сети, мы можем создать bounty secrets — приватные ключи для кошельков с некоторым количеством токенов. Любой может взломать систему и получить награду. Факт похищения награды будет доказательством компрометации безопасности. Этот же механизм может использоваться для предупреждения о потенциальной утечке данных отдельными пользователями сети, сигнализируя о том, что они, возможно, раскрыли свой ключ.

VIII. ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ

NuCypher предоставляет инфраструктуру для различных приложений, требующих обмена конфиденциальными данными в качестве основного функционала. Возможность использовать операции шифрования для публичных действий в сети, таких как публикация определенных сообщений, платежи между конкретными сторонами и другие события, позволяет использовать целый ряд приложений, включая:

A. Совместное использование зашифрованных файлов (“Децентрализованный Dropbox”)

Файлы могут быть зашифрованы на стороне клиента и сохранены в децентрализованных файловых системах, таких как Swarm [38], IPFS [39], Sia [40], или Storj [41], или централизованных, таких как S3. Файлами можно легко обмениваться с одобренными пользователями, предоставляя токен ре-шифрования на основе открытого ключа пользователя. Разрешение доступа можно легко отозвать, удалив данный токен из сети.

B. Шифрованный групповой чат (“Шифрованный Slack”)

PRE идеально подходит для сквозного шифрования сообщений, в котором нескольким участникам требуется доступ на чтение и запись в канал. Участники могут быть легко добавлены или удалены в чат путем выдачи или отзыва токена ре-шифрования. Это позволяет избежать накладных расходов на шифрование и отправку сообщений для каждого участника.

C. Электронные медицинские записи, контролируемые пациентом (EHR)

Контролируемая пациентом EHR, в которой пациент владеет своими данными и ключами шифрования, в отличие от централизованных систем, таких как Epic. Опять же, данные могут храниться как централизованно, так и децентрализованно. Когда пациент хочет поделиться своими зашифрованными данными с больницей или страховой компанией, они выдают токен ре-шифрования, который предоставляет временный доступ к данным.

D. Децентрализованное управление цифровыми правами (DDRM)

Криптографическое управление доступом может выступать в качестве децентрализованного DRM. Элементы управления доступом могут быть встроены в само шифрование, чтобы они следовали за данными, где бы они ни находились. Условные токены ре-шифрования могут управляться смарт контрактом и выпускаться только после оплаты. Такие сервисы, как децентрализованный Netflix или шифрованная площадка, продающая программное обеспечение, приложения, фотографии и другой цифровой контент, теперь могут быть построены с использованием NuCypher.

E. Независимое управление идентификацией

Независимая служба управления идентификационной информацией может быть создана с помощью NuCypher. Идентификационные данные могут быть зашифрованы на стороне клиента и сохранены у провайдера. Пользователи могут создавать ключи ре-шифрования для утвержденных приложений. Служба перешифрует идентификационные данные для указанных сторонних приложений, не имея доступ к провайдеру.

F. Управление секретными учетными данными для скриптов и бэкэнд-приложений

NuCypher идеально подходит для хранения любых секретов, таких как конфиденциальные переменные среды, учетные данные баз данных и ключи API. Для сценариев токен ре-шифрования может быть создан на время выполнения сценария, а затем отозван. Например, разработчики могут безопасно хранить зашифрованные учетные данные на GitHub, предоставляя временный доступ к ним после развертывания экземпляра. Даже если репозиторий GitHub является открытым, учетные данные не могут быть использованы посторонними.

G. Управление общими учетными данными и паролями

NuCypher может управлять общими учетными данными, которые сотрудники используют для доступа к веб-службам. Журнал аудита может быть создан для отслеживания того, кто к чему обращается. Когда сотрудник уходит, легко отозвать доступ или даже свернуть ключи.

H. Обязательный журнал доступа

В некоторых компаниях клиенты должны публиковать журналы доступа к конфиденциальным файлам. Это требует, чтобы каждый доступ к файлам был записан, и условное ре-шифрование может использоваться для контроля этих правил и ведения журнала.

I. Управление мобильными устройствами (MDM)

При настройке MDM компании токены ре-шифрования можно создавать для определённых устройств. Когда устройство потеряно или выведено из эксплуатации, или сотрудник покидает организацию, токен ре-шифрования можно удалить, чтобы отменить доступ к устройству. Это позволяет избежать проблемы реорганизации иерархических деревьев ключей.

J. Частное использование NuCypher

Чтобы принести пользу традиционным компаниям (финансовым, медицинским, правительственным или IoT-компаниям), NuCypher можно развернуть в частном порядке. Он может использовать приватные (или консорциум) блокчейны, быть федеративным (где майнеры заменяются предварительно выбранными узлами, контролируемыми группой предприятий) или даже централизованным (где одна сторона контролирует все ключи ре-шифрования). Эти виды использования по-прежнему будут пользоваться преимуществами прокси ре-шифрования, однако стойкость к цензуре будет не столь эффективной.

IX. РЕЗЮМЕ

NuCypher - это децентрализованная служба управления ключами и криптографический контроль доступа для блокчейна и децентрализованных приложений. Разработчики и компании могут использовать его для создания высокозащищенных приложений в сфере здравоохранения, финансовых услуг и т.д. Используя совместный доступ к секретным данным с помощью публичного блокчейна, NuCypher позволяет создавать всевозможные приложения, от площадок по шифрованию контента и управления секретными учетными данными до контролируемых пациентом электронных медицинских записей.

X. БЛАГОДАРНОСТИ

Мы хотели бы поблагодарить Giuseppe Ateniese из Stevens Institute of Technology и Isaac Agudo Ruiz из University of Málaga за помощь в разработке алгоритмов прокси ре-шифрования. Мы также благодарим Dave

Evans из University of Virginia за консультирование нашей компании на протяжении всего ее существования и его помощи в отношении многопоточных вычислений. Stefano Bernardi, Tom Ding, и многих других за их помощь в создании токеномики.

---

[1] Gabriel Kaptchuk, Ian Miers, and Matthew Green, "Managing secrets with consensus networks: Fairness, ransomware and access control," Cryptology ePrint Archive, Report 2017/201 (2017).

[2] Wikipedia, "Proxy re-encryption — Wikipedia, the free encyclopedia," (2017).

[3] Wikipedia, "Key management — Wikipedia, the free encyclopedia," (2017).

[4] Wikipedia, "Hardware security module — Wikipedia, the free encyclopedia," (2017).

[5] HashiCorp Inc., "Vault by hashicorp," (2017).

[6] Amazon Inc., "Aws cloudhsm," (2017).

[7] Alphabet Inc., "Cloud key management service," (2017).

[8] Microsoft Inc., "Key vault," (2017).

[9] TrueVault Inc., "Hipaa compliant api & secure data store," (2017).

[10] Wikipedia, "Advanced encryption standard — Wikipedia, the free encyclopedia," (2017).

[11] Wikipedia, "Salsa20 — Wikipedia, the free encyclopedia," (2017).

[12] David Nuñez, Isaac Agudo, and Javier Lopez, "Proxy Re-Encryption: Analysis of Constructions and its Application to Secure Access Delegation," Journal of Network and Computer Applications 87, 193–209 (2017).

[13] NuCypher, "Nucypher hadoop: Delegated access control and encryption management system," (2017).

[14] "Umbral: a threshold proxy re-encryption scheme," .

[15] Matt Blaze, Gerrit Bleumer, and Martin Strauss, "Divertible protocols and atomic proxy cryptography," in Advances in Cryptology — EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31 – June 4, 1998 Proceedings, edited by Kaisa Nyberg (Springer Berlin Heidelberg, Berlin, Heidelberg, 1998) pp. 127–144.

[16] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," ACM Trans. Inf. Syst. Secur. 9, 1–30 (2006).

[17] Wikipedia, "Ntruencrypt — Wikipedia, the free encyclopedia," (2017).

[18] David Nuñez, Isaac Agudo, and Javier Lopez, "Ntrureencrypt: An efficient proxy re-encryption scheme based on ntru," in Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15 (ACM, New York, NY, USA, 2015) pp. 179–189.

[19] Le Trieu Phong, Lihua Wang, Yoshinori Aono, Manh Ha Nguyen, and Xavier Boyen, "Proxy Re-Encryption Schemes with Key Privacy from LWE." IACR Cryptology ePrint Archive 2016, 327 (2016).

[20] Wikipedia, "Chosen-plaintext attack — wikipedia, the free encyclopedia," (2017).

[21] Wikipedia, "Chosen-ciphertext attack — wikipedia, the free encyclopedia," (2017).

[22] Vitalik Buterin, "Secret sharing daos: The other crypto 2.0," (2014).

[23] Guy Zyskind, Oz Nathan, and Alex Pentland, Enigma: Decentralized Computation Platform with Guaranteed Privacy, Tech. Rep. (2015).

[24] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger, "Key-private proxy re-encryption," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5473, 279–294 (2009).

[25] Jason Teutsch and Christian Reitwießner, "A scalable verification solution for blockchains," (2017).

[26] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and \Lukasz Mazurek, "Secure Multiparty Computations on Bitcoin," Commun. ACM 59, 76–84 (2016).

[27] Iddo Bentov and Ranjit Kumaresan, "How to use Bitcoin to design fair protocols," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8617 LNCS, 421–439 (2014).

[28] Giuseppe Ateniese, "Accountable Storage," , 1–18.

[29] Xiaodong Lin and Rongxing Lu, "Proxy Re-encryption with Delegatable Verifiability," (2016) pp. 120–133.

[30] Consensys, "Introduction to zk-SNARKs with examples," (2017).

[31] Yanjiang Yang, Liang Gu, and Feng Bao, "Addressing leakage of re-encryption key in proxy re-encryption using trusted computing," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6802 LNCS, 189–199 (2011).

[32] Wikipedia, "Software guard extensions — wikipedia, the free encyclopedia," (2017).

[33] Gabriel Kaptchuk, Ian Miers, and Matthew Green 0001, "Managing Secrets with Consensus Networks: Fairness, Ransomware and Access Control." IACR Cryptology ePrint Archive 2017, 201 (2017).

[34] Giorgos Vasiliadis, Elias Athanasopoulos, Michalis Polychronakis, and Sotiris Ioannidis, "PixelVault: Using GPUs for Securing Cryptographic Operations," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security , 1131–1142 (2014).

[35] Steven Myers and Adam Shull, "Efficient hybrid proxy re-encryption for practical revocation and key rotation," Cryptology ePrint Archive, Report 2017/833 (2017), http://eprint.iacr.org/2017/833.

[36] Evan Duffield and Daniel Diaz, "Dash: A privacy-centric crypto-currency," (2015).

[37] Benoît Libert and Damien Vergnaud, "Tracing malicious proxies in proxy re-encryption," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5209 LNCS, 332–353 (2008).

[38] Victor Trón, Aron Fischer, and Daniel Varga, "Smash-proof: auditable storage for swarm secured by masked audit secret hash," (2016).

[39] Juan Benet, "IPFS — content addressed, versioned, P2P file system," (2014).

[40] Nebulous Inc., "Sia: Simple decentralized storage," (2014).

[41] Storj Labs Inc., "A peer-to-peer cloud storage network," (2016).