NuCypher KMS: 탈중앙화 키 관리 시스템

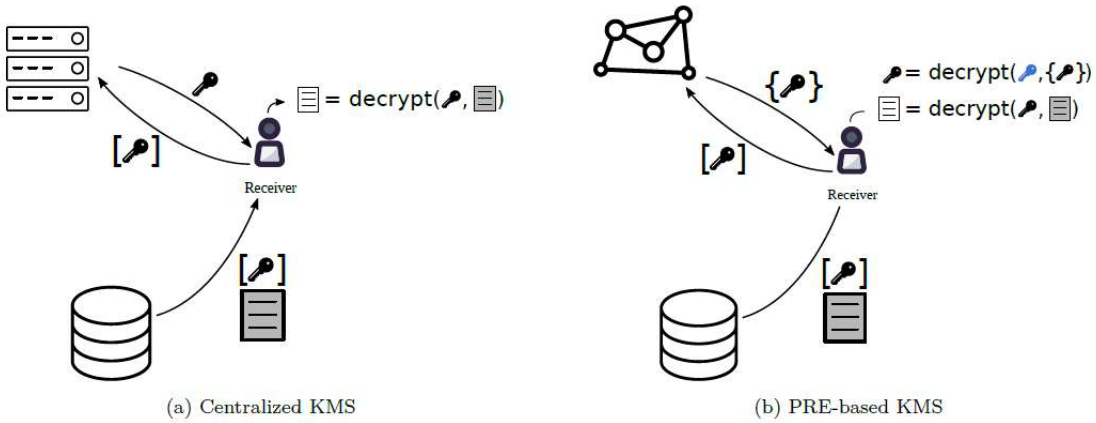
Michael Egorov[1] and MacLane Wilkison[2]

NuCypher

David Nuñez‡
NICS Lab, Universidad de Málaga, Spain

(Dated: December 21, 2017)

NuCypher KMS는 합의 네트워크(consensus networks)로 개인적이고 암호화된 데이터를 안전하게 저장하고 처리할 때 생기는 제한사항을 해결하는 탈중앙화 키 관리 시스템(KMS) 이다[1]. 이 시스템은 프록시 재암호화를 이용(leveraging)하여 탈중앙화된 네트워크가 수행하는 암호화와 암호화 접근 제어를 제공한다[2]. 서비스 솔루션으로 제공되는 중앙화된 키 관리 시스템과 다르게 탈중앙화 키 관리 시스템은 신뢰 가는 서비스 제공자를 필요로하지 않는다. NuCypher KMS를 사용하면 분산 형 및 중앙 집중식 응용 프로그램 모두에서 중요한 데이터를 공유할 수 있으므로 의료 관리부터 ID 관리, 탈중앙화 콘텐츠 시장까지 다양한 응용 프로그램에 보안 인프라를 제공할 수 있다. SSL / TLS가 모든 보안 웹 응용 프로그램의 필수 요소인 것처럼 NuCypher KMS는 분산 응용 프로그램의 필수 요소가 될 것이다.



(a) Centralized KMS (b) PRE-based KMS

그림 1: 중앙화된 키 관리 시스템(KMS)과 프록시 재암호화(PRE)를 사용하는 키 관리 시스템의 차이점

I. 개요

NuCypher KMS는 탈중앙화 키 관리 시스템(KMS), 암호화 및 접근 권한(access) 제어 서비스다. NuCypher는 프록시 재암호화로 복호화 권한을 위임하는 방법으로 대칭키 또는 공개 키 암호화 같은 전통적인 방식으로는 할 수 없던 공개 합의 네트워크상에 임의의 수의 참가자 간에 개인정보 공유를 가능하게 한다. 네이티브 토큰은 네트워크 참여자에게 키 관리 및 접근 권한의 위임/철회 작업을 수행하도록 인센티브를 부여하기 위해 사용된다.

A. 배경

키 관리 시스템(KMS)은 응용 프로그램과 장치를 위한 암호화 키 생성, 배포 그리고 관리를 위한 통합적 접근 방식이다(그림. 1). 키 관리 시스템은 키 생성, 배포 및 순환 같은 백엔드 기능뿐만 아니라 장치에서의 키 주입, 저장, 관리하는 사용자 측 기능도 포함된다[3].

신뢰의 근간으로서 키 관리 시스템은 적절하게 구성되고, 관리되고, 보호되어야만 한다. 여태까지 이것은 키 관리 시스템을 하드웨어 보안 모듈(HSM) 또는 헤시콥(HashiCorp)의 Vault[5] 같은 도구를 사용하여 직접 구성하여 배포하는 것을 의미했다. 그러나 이것

은 고도의 기술적 정교함뿐만 아니라 선행자본 투자가 필요하다. 기술적 부담과 가격 부담을 줄이기 위해서 Amazon CloudHSM [6], Google Cloud KMS [7], Azure Key Vault [8] 및 TrueVault [9]와 같은 공급업체가 키 관리 시스템을 서비스로 제공하기 시작했다. 하지만 키 관리 시스템은 서비스 제공자의 과도한 신뢰가 필요하기에 보안이 중요한 응용 프로그램에는 적합하지 않다.

비트코인이나 이더리움 같은 공개 합의 네트워크는 이러한 중앙화 문제에 대한 획기적인 해결책이다. 그러나 비밀 데이터의 조작을 포함하는 암호 연산을 수행할 때 공개 합의 네트워크의 한계는 너무나도 명확하다[1]. 합의 네트워크는 자발적 노드의 네트워크로 이루어져 있어 지속적으로 변화하며 가용성 및 액세스 관리 규칙을 적용할 때 중앙 기반만큼 견고하게 신뢰할 수 없다.

NuCypher KMS는 탈중앙화 네트워크를 사용하여 중앙 서비스 공급자에 대한 의존성을 제거하고, 프록시 재암호화를 사용하여 암호화 접근을 제어하며, 토큰 인센티브 메커니즘으로 안정성, 가용성 및 정확성을 보장한다. 프록시 재암호화를 사용하기 때문에 암호화되지 않은 대칭 키(개인 정보 암호 해독 기능 제공)는 절대로 서버 측에 노출되지 않으며(그림 1) 단일 보안 장애점 또한 없다. 해킹 당한 경우에도 해커는 재암호화 키만 가져갈 수 있으며 파일에 대한 접근은 여전히 보호된다.

II. 구성

A. 암호 프리미티브

1. 대칭 암호화

대칭 키 또는 비밀 키 암호화는 사용자가 공유하는 공통의 비밀 키를 사용한다. 편의상 이 공통 비밀 키를 DEK(데이터 암호화 키, data encryption key)라고 하겠다.

대칭 암호에 대해 다음 두 개의 연산이 정의된다:

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

$$d = decrypt_{sym}(dek,c); \quad (2)$$

여기서 d는 평문 데이터이고 c는 암호문(암호화된 데이터)이다.
우리의 목적에 잘 부합하는 대칭 키 암호화 알고리즘은 AES(일반적으로 하드웨어 가속되기 때문에)[10]와 Salsa20[11]이다.
대칭 블록 암호는 다른 모드에서 작동할 수 있다. 우리는 강력한 의미론적인 보안을 보장하기 위해, 확률적 암호화(예: AES의 GCM)를 생성하는 작동 모드를 사용한다. 단순화를 위해서, 우리는 암호문 c의 일부로써 조작 모드와 관련된 넌스 값을 다루는 특정 연산 모드에 대한 세부 사항을 생략한다.

2. 공개 키 암호화

공개 키 암호화(PKE, public-key encryption)는 두 당사자(발신자와 수신자)가 비밀을 공유하지 않고도 정보를 교환할 수 있는 암호화의 일종이다. 모든 참가자는 한 쌍의 키(공개 키 pk 및 비밀/개인 키 sk)를 가진다. 송신자가 키 쌍 sk_s /pk_s 를 갖고 수신자가 키 쌍 sk_r /pk_r을갖는 경우, 송신자는 수신자의 공개 키를 사용하여 메시지를 암호화할 수 있으며 수신자는 자신의 비밀 키를 사용하여 암호를 해독할 수 있다.
대칭 암호화의 효율성과 공개 키 암호화의 편리성을 결합한 하이브리드 암호화 시스템을 만들 수 있다. 하이브리드 암호 시스템에서의 암호화 과정은 다음과 같은 함수를 정의한다:

$$dek = random(); \quad (3)$$

$$c = encrypt_{sym}(dek,d\ ); \quad (4)$$

$$edek = encrypt_{pke}(dek,pk_r). \quad (5)$$

이 쌍(edek,c)은 암호화된 데이터를 변환하는 데 사용된다. 수신기가 수행하는 해독은 다음 함수를 정의한다:

$$dek = decrypt_{pke}(edek, sk_r); \quad (6)$$

$$d = decrypt_{sym}(dek, c). \quad (7)$$

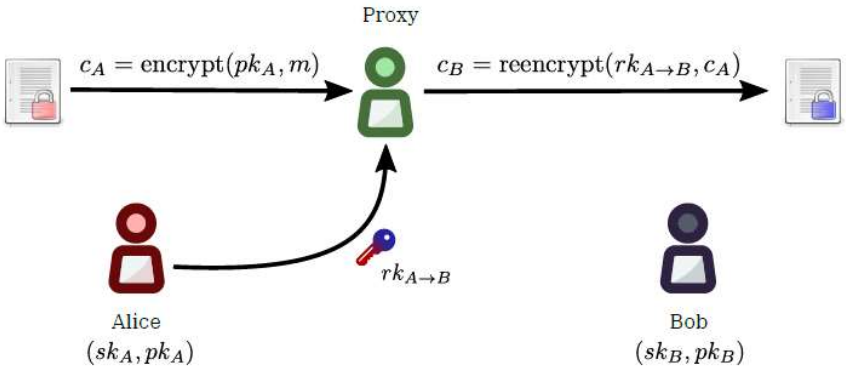


그림 2: PRE 환경에서의 주된 행위자와 상호작용

3. 프록시 재암호화

프록시 재암호화(PRE, proxy re-encryption)[2, 12]는, 프록시 개체(entity)가 기본 메시지에 대해 아무것도 배우지 않고도 하나의 공개 키에서 다른 공개 키로 암호문을 변환할 수 있게 하는 일종의 공개 키 암호화(PKE)이다 (그림 2).

전형적인 프록시 재암호화 시나리오에는 여러 행위자가 있다. 데이터 소유자인 앨리스는 공개 키 pk_A를 가진다. 이 키를 알고 있는 모든 개체는, 오직 앨리스만 그의 비밀 키 sk_A를 사용해서 해독할 수 있는 암호화된 데이터를 생성할 수 있다. 데이터 생성자가 앨리스의 공개 키 pk_A를 사용하여 메시지 m을 암호화하여 암호문 c_A를 생성한다고 가정하자. 앨리스는 밥에게 메시지 m에 대한 접근 권한을 위임하기로 결정한다. 밥 또한 자신의 키 쌍(pk_B, sk_B)을 가지고 있을 때, 이를 위해 앨리스가 다시 암호화 키를 만든다:

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

일회용의 단방향 프록시 재암호화 방식일 때, 재암호화 함수는 비가역적이고, $rk_{A\to B}$가 구성 요소로 분해될 수 없다는 점이 중요하다(최소한 sk_A또는 sk_B를 알지 못한다면). c_A가 c_B로 변형되도록 c_A를 다시 암호화하는 것만이 가능하다:

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

그런 다음 밥은 자신의 비밀 키 sk_B를 사용하여 c_B를 해독할 수 있다.

PRE는 1:1 통신에 이상적인 기존 PKE 프로토콜과 비교할 때, 임의의 수의 데이터 생성자 및 소비자간의 N:N 통신에 대해 더 확장성이 좋다. 재암호화 토큰은 언제든지 작성하고 적용할 수 있으므로 미리 메시지 수신자를 알 필요가 없다. 이것은 블록체인, IoT, 빅 데이터[13]와 같은 분산 시스템에 적합하다.

다른 속성을 가진 많은 프록시 재암호화 알고리즘이 있다. NuCypher KMS의 첫 번째 버전에서는 우리가 프록시 재암호화 알고리즘을 작성한[14] ECIES를 선택한다. 이것은 가장 간단한 (그리고 성능이 좋은) 알고리즘인 BBS98[15]과 매우 유사하지만 더 나은 보안을 보장한다. 그러나 때로 우리는 재암호화를 여러 노드에 위임하여 이들 간의 신뢰를 분리하고 시간 기반 또는 조건부 재암호화 정책을 적용하기를 원한다. 이 경우 우리는 AFGH 방식[16]을 사용한다. 필요하다면 양자내성(Quantum-resistant) NTRU도 사용될 수 있다[17, 18].

BBS98과 마찬가지로, ECIES에 대한 프록시 재암호화는 비밀+공개 키가 아닌 두 개의 비밀 키에서 재암호화 키를 만든다. 그러나 우리는 발신자가 수신자의 비밀 키를 알게 되지 않기를 원한다. 이 문제를 해결하기 위해 임시 키 쌍인 sk_e /pk_e 를 무작위로 생성한다. 그 후 접근 권한 위임은 다음과 같다:

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

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

$$sk_e^0 = \text{encrypt}_{pke}(pk_B, sk_e); \quad (12)$$

$$rk_{A\to B} = (rkA\to e, ske0\ ); \quad (13)$$

재암호화 노드는 $rk_{A\to e}$ 를 사용하여 (기본 메시지가 m인) 임의의 암호문 c_A를 다시 암호화하여 sk_e에 의해 해독될 수 있도록 한다. 수신기는 재암호화된 암호문을 해독하기 위해 sk_e^0 를 필요로 하기 때문에, 그것을 재암호화 결과에 첨부한다. 따라서 재암호화 프로세스는 다음과 같다:

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

$$c_B = (c_e, sk_e^0); \quad (15)$$

(16)

그러면 수신기의 암호 해독은 다음과 같다:

$$sk_e = decrypt_{pke}(sk_B, sk`_e); \quad (17)$$

$$m = decrypt_{pke}(sk_e, c_e); \quad (18)$$

(19)

이 접근법은 이전에 Ateniese 등[16]에 의해 언급되었고, 프록시 재암호화 작업을 넘어서는 것이다. 이 방법은 "핵심적인 최적화"로 간주되지 않지만, 우리 사례에서는 매우 경쟁력 있는 성과를 보여주고 있다.

B. 재암호화 방식에 대한 간단한 검토

재암호화 방식이 사용할 수 있는 다양한 속성이 있다. 이 하위 절에서는 우리의 사례와 관련된 몇가지 것들을 고려한다. 알려진 모든 프록시 재암호화 알고리즘과 그 속성을 고려한 전체 조사가 최근에 발표되었다[12].

고려해야 할 중요한 속성 중 하나는 알고리즘이 상호작용식인지 여부이다. "상호작용식"은 두 개의 비밀 키로 재암호화 키가 계산된다는 것을 의미한다:

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

"비상호작용식"은 소유자의 개인 키와 대리인의 공개 키만 알아도 된다는 것을 의미한다:

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

상호작용식 알고리즘의 예로는 BBS98[15], 우리의 ECIES 재암호화(BBS98 기반) 및 LWE 기반 재암호화[19]가 있다. "비상호작용식" 알고리즘의 예는 AFGH[16]이다. 초기에 우리가 비상호작용식 알고리즘에만 관심이 있는 것처럼 보였을지라도, 임시(ephemeral) 키(식 10-12)를 사용하여 개인 키가 아닌 공개 키와 공유하기 위해 프로토콜 레벨에서 상호작용식 알고리즘을 조정할 수 있다.

또 다른 흥미로운 속성은 알고리즘이 단방향인지 또는 양방향인지 여부이다. 양방향성은 바로 re_{ab} 에서 re_{ba} 를 계산할 수 있음을 의미한다. 단방향 알고리즘에서는 그것이 불

가능하다. 임시 키를 사용할 때 양방향 알고리즘은 효과적으로 단방향이다(식 10-12). 그럼에도 불구하고, V-C장은 단방향이 복잡한 계층적 데이터를 확장 가능한 방식으로 공유하기에 매우 편리하다는 것을 보여준다. 양방향 알고리즘의 예는 BBS98[15]이다.

프록시 재암호화 알고리즘은 싱글홉 또는 멀티홉일 수 있다. "멀티홉"은 re_{ab}와 re_{bc}가 있는 경우, b의 참여없이, 암호문 c_a를 암호문 c_C로 변환하기 위해 이 두 개의 재암호화 키를 연속적으로 적용할 수 있음을 의미한다:

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

때로는 BBS98에서 re_{ac}을 계산할 수도 있다:

$$re_{ac} = reab \cdot rebc. \quad (23)$$

"싱글홉"은 c_b가 다시 암호화를 통해 얻어지면 더 이상 다시 암호화할 수 없음을 의미한다. 멀티홉 방식은 키 순환(V-G장)과 계층적 데이터 공유(V-C장)에 유용하다. 임시적인 키 트릭(EQ 10-12)은 효과적으로 단일 홉이다(임시 키는 다른 상대방과 데이터를 공유할 때만 생성되므로 키 순환은 여전히 가능하지만). 다중 홉 알고리즘의 예로는 BBS98[15]과 LWE 기반 알고리즘 [19]이 있다. AFGH 알고리즘[16]은 단일 홉이다.

외견상 중요한 또 다른 속성은 결탁 저항(collusion resistance)이다. 비공식적으로, 결탁 저항은 re_{ab}와 sk_b를 사용해서 sk_a를 유도하는 것이 불가능하다는 뜻이다. 반대로, 결탁 저항이 부족하면 re_{ab} 및 sk_b에서 sk_a를 얻을 수 있다. 언뜻 보기에 이것은 보안상 매우 바람직하다. 그러나 결탁 저항 알고리즘을 사용하는 경우에도 공격자가 re_{ab}와 sk_b를 모두 가지고 있다면 그는 원래 sk_a로 해독할 수 있는 모든 데이터를 다시 암호화하고 읽을 수 있다. 따라서, 동일한 키 쌍이 다른 목적(예: 서명, 키 유도 등)에 사용될 때만 결탁 저항이 중요해진다. 그럼에도 불구하고 두 기능에 대해 별도의 키 쌍을 사용하는 것이 좋다. 우리의 목적을 위해서는, 암호화 이외에 위임자의 키 쌍을 다른 용도로 사용하지 않기 때문에 결탁 저항이 우선 순위는 아니다. 결탁 저항 알고리즘은 AFGH[16]와 LWE 기반 재암호화[19]를 포함한다. 비 결탁 저항 알고리즘은 BBS98[15] 및 ECIES 재암호화이다.

우리는 재암호화 키(II-A장)에서 데이터의 생산자 와/또는 소비자를 식별 불가능하게 해야 할 수도 있다. 종종 시스템의 모든 공개 키를 알고 있는 프록시가 이 정보를 추론할 수 있습니다. 많은 PRE 체계는 재암호화 관점에서 익명이 아니다[15,16]. 그러나 LWE 기반의 재암호화 기법[19]은 익명성을 띄고 있다.

마지막으로 CPA보안[20] (선택된 평문 공격에 대한 보안)과 CCA보안(선택된 암호문 공

격에 대한 보안)의 개념은 프록시 재암호화에 적용 할 수 있다. 지금까지 언급한 모든 알고리즘은 ECIES 재암호화 및 LWE를 제외하고는 CCA보안 뿐이며[18], 이들은 CCA 보안이다.

C. 암호화 된 메시지 서명

공개 키 암호화 알고리즘에서 누구든지 pub_a를 사용하여 암호화할 수 있다. 이것이 가능하므로 네트워크의 악의적 사용자가 A인 것처럼 데이터를 암호화할 수 있다. 따라서 데이터는 송신자의 정체를 수신자에게 증명하기 위해서 서명되어야 한다.
하지만 우리는 프로토콜을 익명화할 수 있도록 만들고자 한다. 왜냐하면 데이터의 소유주를 인증하는 공개적인 디지털 서명은 재암호화 노드가 소유자로부터 돈을 갈취할 가능성을 높이기 때문이다.

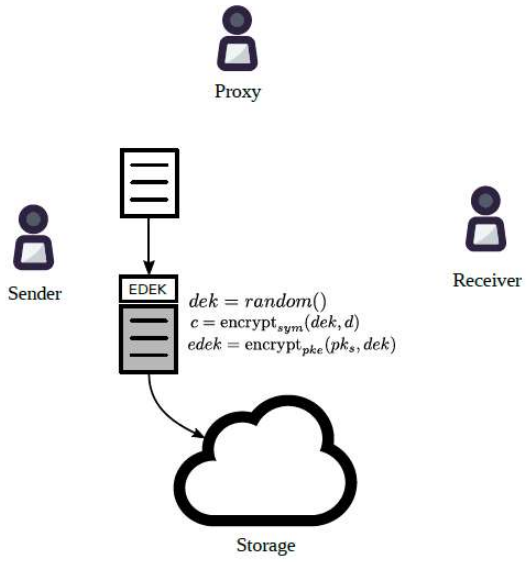


그림 3: 구조: 암호화

- 암호문이 해독되기 전까지 서명을 확인하지 못하도록 디지털 서명을 평문에 포함시킨다.

- 서명과 암호화에 다른 키 쌍을 사용한다(특히 일부 재암호화 암호시스템의 결탁 저항이 부족하기 때문에).

D. 재암호화 노드

클라우드나 탈중앙화 저장소에 데이터가 저장되어 있을 때, 그것은 데이터 소유자의 키 pks 로 암호화 된다(그림 3). 데이터 자체는 랜덤한 대칭 키인 dek로 암호화 되고, 파일 하나당 한 개의 대칭키가 있다. pks 로 암호화된 dek는 암호화된 데이터에 부착된다. 이 조합(edek, c)은 IPFS, Swarm, S3 등 어떤 탈중앙화 되거나 중앙화된 저장소 어디든 저장할 수 있다.

데이터를 저장할 때, 우리가 접근 권한을 위임한 사용자를 사전에 반드시 알아야 할 필요가 없다. 수신자는 자신의 공개 키를 보내는 사람에게 반드시 공개하여야 한다(그림 4). 공개 키가 이더리움 네트워크의 주소와 일치하는 것은 의미가 있을 수 있다(예를 들어 디지털 콘텐츠 구독을 위해 해당 주소에서 지불이 이루어졌음을 증명할 때). 전송자는 재암호화키 $re_{s\rightarrow r}$(필요에 따라 암호화된 랜덤 임시 키를 포함)를 생성하고, 탈중앙화 네트워크에서 활성 노드 중 지분 증명방식에 의해 선택된 랜덤한 재암호화 노드에 보낸다. 이중화 또는 보안을 위해 다수의 노드를 선택한 경우는 추후 논의하겠다. 수신자와 송신자가 공유하는 데이터를 가지고 있는 노드는 이 정보를 네트워크에 등록한다.

수신자가 전송자 공유한 데이터를 해독하고자 하면 먼저 암호화 스트림 또는 저장소에서 데이터를 다운받아야 한다(그림 5). 그는 메시지에서 edek를 분리하고 edek를 재암호화 노드 네트워크로 보내고 수신자와 송신자 사이 데이터를 공유할 수 있는 활성 재암호화 노드를 찾는다(재암호화 키(들) $re_{s\rightarrow r}$를 가지고 있는 노드). 수신자는 노드에게 재암호화 키 edek를 $edek_0$로 변환할 것을 요청하고 이후 자신의 비밀키 sk_r 를 사용하여 해독하여 DEK를 획득한다. 이제 그는 DEK를 이용하여 데이터 덩어리를 해독할 수 있다.

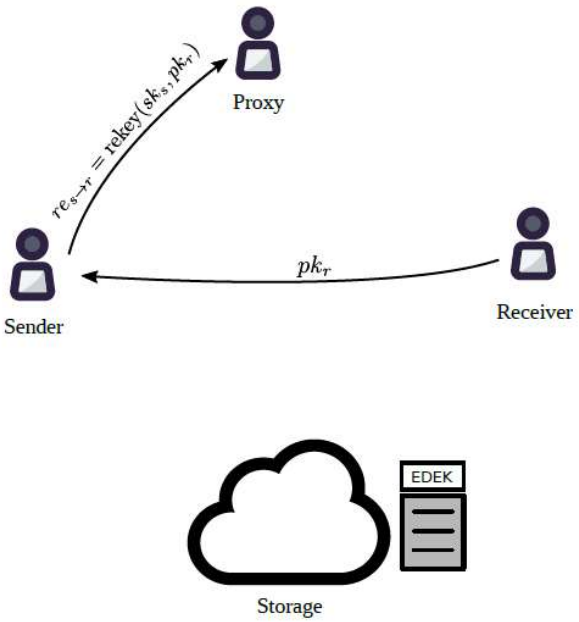


그림 4: 구조: 접근 권한 위임

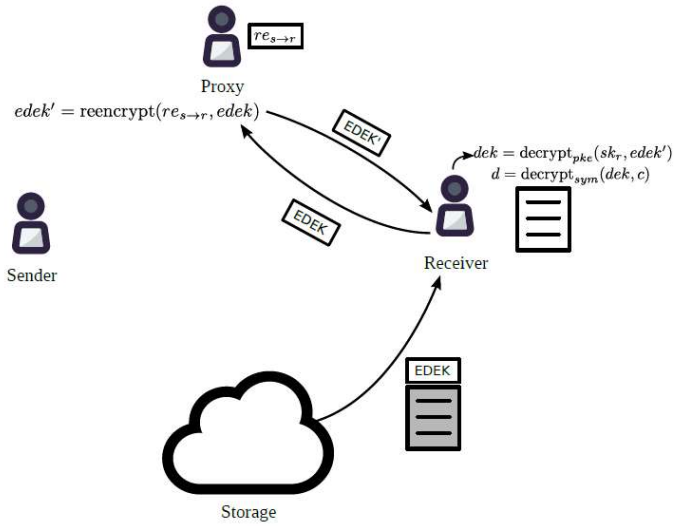


그림 5: 구조: 해독

Ⅲ. 네트워크 보안

네트워크에는 액세스 관리 정책을 적용하는 다수의 재암호화 노드가 있다.

프록시 재암호화를 통해 NuCypher KMS는 상시 온라인 상태이며 항상 신뢰할 수 있는 독립체(예: 전통적인 키 관리 시스템) 없이 액세스 관리와 암호 해독 권한 사이의 신뢰를 분리 할 수 있게 되었다. 채굴자들은 평문(plaintext) 데이터 자체는 물론, 데이터를 해독할 수 있는 어떤 것도 보지 못한다. 그들은 재암호화된 키의 저장과 재암호화 함수의 적용에만 전적인 책임이 있다.

이 모델의 첫번째 위험성은 채굴자와 데이터 수신자와의 결탁이다. 만약 채굴자가 데이터 수신자에게 자료에 대한 재암호화 키를 준다면, 데이터는 언제든지(상황이나 시간적 제약을 피해서) 데이터 수신자에 의해서 해독될 수 있다. 우리는 이러한 위협에 여러 방법으로 대응한다: 재암호화 키의 반익명성(pseudo-anonymity), 분할 키 프록시 재암호화, 그리고 챌린지 프로토콜 (challenge protocol). 또한, 우리는 7장에서 설명할 공정한 작업(operation)에 따른 경제적 인센티브도 적용한다.

두번째 위험성은 노드 오작동(재암호화 수행 대신 가짜 데이터를 반환하는 것)이다. 우리는 이것을 챌린지 프로토콜(challenge protocol)을 통해서 해결한다.

세번째 위험성은 50% 공격을 위한 노드들 사이의 결탁이다. 이 위험성은 MPC(Multi-Party Computation) (예: 에니그마) 에게는 대부분 치명적이다. 그러나 우리의 경우 공격자는 재암호화 정책을 잘못 적용시키는데 그치며, 데이터를 해독할 수 있거나 데이터에 접근 권한이 없는 사용자에게 접근 권한을 허용할 수 있는 것이 아니다. 이상적으로 시스템은 가능한 한 분산되어야만 한다. 하지만 50 %의 공격은 마치 작업 증명 암호화폐에서 50 % 공격에서 공격자가 자금을 이동시킬 수 있는 능력을 갖지 못하는것과 같이, 데이터의 기밀성을 손상시키지 않는다.

A. 반익명성

재암호화 노드가 자신이 무엇을 재암호화하는지 알지 못하는 것은 시스템의 보안에 매우 유용하다. 이렇게 하면 그들(*재암호화 노드들)이 결탁 공격(Collusion attack) 할 재암호화 키를 알지 못하게 한다(그리고 네트워크가 분산되어 있을 때 모든 네트워크 참여자와 결탁하는 것은 불가능하다). 그러나 재암호화 키의 반익명성(pseudo-anonymity)은 챌린지 프로토콜(III-C장: challenge protocol)의 작동을 가능하도록 해준다.

재암호화를 위한 익명 프로토콜의 설계는 미래의 기술로서 남기지만, 이것(*재암호화를 위한 익명 프로토콜)이 다음과 같은 특성은 반드시 가져야 함을(또는 가지지 말아야 함을) 짚고 넘어가겠다. 첫째, 재암호화 구조는 키-개인[19, 24](key-private)의 구조여야 한다. 그렇지 않을 경우, 알려진 모든 공개 키 쌍을 반복하여 키 소유권을 확인할 수 있다.

둘째, 재암호화 노드와 데이터 수신자는 같은 재암호화 키에 대하여 동일한 식별자(identifier)를 가져서는 안 된다. 이는 수신자가 키를 찾을 수 있도록 키-값 저장소에 재암호화 키를 저장하는 간단한 방법을 배제한다.

B. 분할-키 재암호화

재암호화 노드가 지시 된 조건 정책을 적용하는 대신 데이터를 즉시 재암호화하기로 결정하는 상황이 있을 수 있다. 이 문제를 해결하기 위해 분할 키 프록시 재암호화 체계를 사용할 수 있다.

하나의 재암호화 키 대신 m-of-m방식의 재암호화 키를 사용하여 "재암호화 지분"을 생성할 수 있다. 이러한 지분은 사용자 측과 결합될 수도 있다. 이러한 m-of-m방식은 AFGH[16]방식 암호화를 위해 존재한다. 이 방식에서 결탁 공격을 위해서는 m만큼의 채굴자와 데이터 수신자를 필요로 한다.

말라가 대학(University of Mlaga)의 NICS 연구소와 함께 개발 한 임계 값 기반 m-of-m 기법(Umbral)은 이 작업에 더 적합한 것처럼 보인다. 이 방식은 또한 서드파티가 재암호화를 검증할 수 있게 하여 노드를 신뢰도 있게 유지하는 데 중요한 역할을 한다.

C. 챌린지 프로토콜

채굴자가 데이터를 올바르게 재암호화하는 대신 임의의 숫자를 반환할 위험이 있다. 데이터가 비공개(private)이므로 시스템 사용자는 이 데이터와 사용자들의 키를 채굴자가 속였다는 증거로 제시할 수 없다.

채굴자가 "진정한 재암호화"와 무작위 데이터의 재암호화를 구별하는 것은 불가능하다. 그래서 우리는 채굴자에 챌린지하도록 특별히 고안된 여러 가지 "가짜" 재암호화 키를 생성할 수 있다. 만약 채굴자가 속임수를 썻다면, 이 챌린지을 위해서 만들어진 키와 데이터는 어떤 개인 키와도 관련되지 않을 것이다.

채굴자는 네트워크에 재암호화하기 전과 후의 데이터의 해시를 표시해야 한다. 만약 이 재암호화가 챌린지가 있고, 채굴자가 속였었다면, 챌린지를 제기한 사람은 이 챌린지에 연관된 민감하지 않은(*non-sensitive: 실제 암호화에 사용되지 않은) 키가 실제로 다른 재암호화 결과를 산출한다는 것을 증명할 수 있고, 이 경우 채굴자의 담보 보증금을 챌린지 제기자에게 수여할 수 있다. 또한 시스템은 의도적으로 많은 수의 "잘못된 재암호화"를 만들어서 Truebit[25]가 지적한대로 챌린지 제기자에게 인센티브를 부여해야 한

다.
챌린지 프로토콜을 설계하는 것은 "공정한 교환"프로토콜 [26-28]과 관련된 복잡한 문제이다. 이는 신중한 설계와 테스트가 필요하며 Ethereum의 지분증명방식 (Proof-of-Stake)(Casper) 프로토콜은 이러한 복잡성에 직면 해 있다. 암호화 알고리즘의 수준에서 정확성을 검사하는 것만으로도 가능할 수도 있다 [29].
재암호화 키가 유출되지 않도록 특별한 주의를 기울여야 한다. 다음 챌린지 프로토콜이 제안한다.
재암호화 키에 대한 책임을 수락하면 채굴자는 시간 가 지남에 따라 수수료 f를 받을 것으로 예상하므로 데이터 소유자는 코인을 예금한다. 채굴자는 또한 재암호화 키가 누설될 경우 몰수 될 담보 c를 맡겨야 한다.
챌린지 제기자가 채굴자가 재암호화 키를 유출했다는 것을 입증하면 챌린지 제기자가 보상을 받아야 한다. 그러나 데이터 소유자는 챌린지 제기에 따른 보상을 부정하게 수집하기 위해 채굴자에게 챌린지를 제기 할 수 있다. 우리는 이러한 "self challenge"을 불가능하게 만든다. 만약 챌린지 제기가 t시간 이후에 일어났다면, 챌린지 제기자는 αft/T코인을 얻게 된다. 여기서 α<1이다. 이 경우 데이터 소유자는 (1−t/T)f 코인을 돌려받는다. 담보물과 나머지 비용은 c+(1−α)t/T의 총 금액과 함께 네트워크의 다른 참가자들의 이익을 위해 압수당한다.
또한 정책의 철회가 아니라 채굴자에게 "가짜 챌린지"을 하는 데이터 소유자에 대한 인센티브는 없어야 한다. 따라서, "올바른" 철회시, 데이터 소유자는 (1−t/T)f 의 코인을 얻고, 채굴자는 c+ft/T 의 코인을 얻는다. 여기서 c는 담보로 제공된 것이다.

D. 다양한 사용 사례에 대한 가능한 위협 적합성

모바일 장치 관리 사용 사례 (VIIII-I장)에서 가장 중요한 것은 데이터가 위태롭기 전에 분실되거나 도난당한 장치에서의 접근을 취소하는 것이다. 누군가가 장치를 훔쳐 관련 채굴자와 결탁 하는 가능한 공격을 상상해보자. 따라서 채굴자가 사용자를 식별 할 수 있는 방법이 없어야 하며 그 반대 또한 마찬가지이다. 또 다른 가능한 공격은 채굴자 집단이 접근 권한을 취소하고 재암호화를 위한 추가 요금 지불을 요구하는 것이다. 그러나 데이터 소유자가 해당 모바일 장치에 대한 액세스 권한을 쉽게 다시 부여 할 수 있기 때문에 그렇게 할 동기적 이점이 없다. 또 다른 가능한 위협으로는 채굴 노드가 재암호화 키들을 정책의 수명을 넘어서 보관하여 최종 사용자 장치를 공격한 어떤 공격자와 결탁하기를 기다리는 것이다. 이러한 위협을 방지하려면 채굴 노드가 데이터가 가치가 있는

지 없는지 파악할 수 없게 하는 것이 중요하며, 이를 위한 가장 좋은 방법은 데이터 소유자나 데이터 그 자체를 익명으로 처리하는 것이다. 즉, 채굴 노드가 'edek'가 어느 데이터에 대응하는지를 알 수 없게 만드는 것이다.

분산 형 탈중앙화 디지털 저작권 관리(DRM VIII-D장)은 콘텐츠(파일 또는 동영상)가 해독되면 구매 된 것으로 간주하므로 액세스 취소는 실제로 중요한 문제가 되지는 않는다. 그러나 노드가 콘텐츠가 매우 비싸다는 것을 안다면, 구매자에게 접근하여 원 판매자를 배제하면서 콘텐츠에 대한 더 저렴한 가격을 제시할 수 있다. 이를 방지하려면 데이터 수신자의 이름을 익명으로 처리해야 한다. 채굴 노드에서 정확한 가격 정보를 숨기면서 zk-SNARK [30]를 사용하여 필요한 금액을 지불했는지 확인하도록 하는 것 또한 도움이 된다.

파일(VIII-A장) 또는 메시지에 안전한 접근으로 NuCypher KMS를 사용 할 때, 접근 권한 부여와 해지가 중요하다. 따라서 완전한 익명화가 가장 바람직하다. 가능한 공격은 데이터를 데이터의 수신자가 채굴자를 매수하여 해지되었어야 하는 접근을 계속하고, 소유자로부터 채굴자가 접근 권한 해지를 위한 수수료를 갈취하는 것을 포함한다. 익명화는 그러한 공격을 실행 불가능하게 만드는 중요한 부분이다.

E. 하드웨어 적용 보안

채굴자가 잘못 할 경우, 그들은 그들의 담보금을 잃을 위험이 있다. 그러나, 악의적인 목적보다는, 채굴자 노드들 또한 서드 파티 공격의 희생양이 될 수 있다.

노드가 손상되는 것을 방지하기 위해 채굴자들은 SGX또는 NVidia GPU를 지원하는 최신 세대 Intel CPU (Skylake+)와 같이 일반적으로 접근 할 수 있는 보안 하드웨어[31]에서 신뢰할 수 있는 컴퓨팅을 사용하는 것이 한 가지 방법이다.

Intel SGX 기술[32]은 안전한 환경에서 연산을 수행 할 것을 보장한다. 이전에는 채굴자들의 공정성이나 재암호화(re-encryption)[33] 보는 SGX 기술에 의존하는 비밀 관리용 분산 네트워크를 갖는 것이 제안되었다.

그 대신에, GPU 내부에서만 재암호화 키를 가지고 적용 할 수 있었다. GPU가 제한된 기능을 가지면서 신뢰할 수 있는 플랫폼 모듈로 작동 할 수 있음을 보여주었다 (확실하게 재암호화를 실행할 수는 있지만) [34].

IV. 성능 고려

V. 기능

A. 로컬 암호화 라이브러리와 데몬

NuCypher KMS는 기존의 중앙집중식 응용 프로그램과 호환이 가능하다. 파이선 IPFS에서 파일을 공유하는 것은 다음과 같다.

import nkms nkms.connect() ***# Using default config***

path= 'ipfs://QmTkzDw.../to_the_moon.avi'

nkms.share('0xab12...', path, time=86400)

사용할 수 있는 라이브러리 클라이언트가 없으면, 게스(geth)를 사용하여 이더리움 네트워크와 상호작용하는 것과 유사한 로컬 API 서버를 사용하면 된다.
파일을 읽는 것은 다음과 같다:

import nkms

client = ipfsapi.Client(...)

nkms.connect() *# Using default config*

path = 'ipfs://QmTkzDw.../to_the_moon.avi'

edata = client.cat(path)

data = nkms.decrypt(edata, path)

print(data)

"decrypt"함수는 edata를 edek와 실제 암호화된 데이터로 분리하고, 키 관리 시스템(KMS) 네트워크에게 edek를 $edek_0$으로 변환해줄 것을 요청하며, $edek_0$을 받는 사람의 개인 키로 해독한 후 획득한 dek로 데이터를 해독한다.
API의 예비 버전에는 다음과 같은 함수가 들어갈 것이다:

- connect — 구성 파일 또는 인자로부터 환경 설정을 가져오고 탈중앙화 네트워크에 연결시킨다;

- write — 파일 소유자의 공개키로 데이터를 암호화하고 백엔드 저장소에 저장한다;

- read — 백엔드 저장소로 부터 데이터를 다운받고, 탈중앙화 네트워크에게 재암호화할 것을 요청하며, 개인 키로 해석을 한다;

- delete — 파일과 그와 관련된 재암호화 키를 삭제한다;

- decrypt — 이미 읽은 데이터를 해독한다;

- split-edek — 로우 레벨 함수로서 데이터에서 암호화된 대칭 암호 키를 분리한다;

- share/renew/revoke — 파일 경로 또는 정책에 따라 자신이 가진 데이터 또는 파일의 하위집합을 읽을 권한을 생성한다. 정책은 시간 제한이나 다른 조건들을 포함하고 있을 수 있다;

- read-policies/update-policies/delete-policies — 우리가 만든 모든 접근 권한 정책을 읽고 변경한다;

B. 짧은 비밀 공유

로우 레벨의 기능은 바이너리 비밀(예: 데이터베이스 자격 증명) 또는 이러한 비밀들의 집합을 개별 파일에 저장하지 않고 암호화하고 접근 권한을 위임 가능하게 한다. 이런 간단한 비밀을 저장할 백엔드로는 IPFS에 저장된 계층적 구성파일(YAML 또는 JSON 형식으로 이루어진)이거나 더 간단하게는 같은 인스턴스 이미지로 가능하다. 클라이언트 측 소프트웨어는 이 파일을 분석하여 필요한 비밀만을 재암호화하도록 요청할 수 있다. 필드 별 또는 하위 분야별 세분화된 접근 권한을 위한 재암호화 키가 존재한다.

C. 파일 및 계층적 데이터 공유

파일을 다룰 경우, 각 파일 그리고 각 디렉토리는 해당 비밀 키 아래 암호화 된다. 각 파일마다 자체의 암호키, 해당 디렉토리 부터 사용자의 루트폴더까지 각 디렉토리의 암호키들로 암호화 된 DEK(Data Encryption Key)를 가지고 있다. 디렉토리가 공유되었을 때 공유된 디렉토리에 한정된 재암호화 키가 생성된다.
위 방법은 트리구조의 레벨만큼의 EDEK(Encrypted Data Encryption Key)를 저장을 필요로 한다. 만약 우리가 멀티-홉, 유니-다이렉셔널 알고리즘(LWE가 오직 하나인)을 사용한다면, 우리는 바닥 레벨부터 탑 레벨까지 재암호화 키를 가질 수 있다. 디렉토리로의 접근이 허락될 때, 해당 디렉토리의 모든 파일과 하위 디렉토리는 디렉토리의 키로 재암호화 될 경로를 가지고, 그리고 나서 수신자를 위해 재암호화 된다.

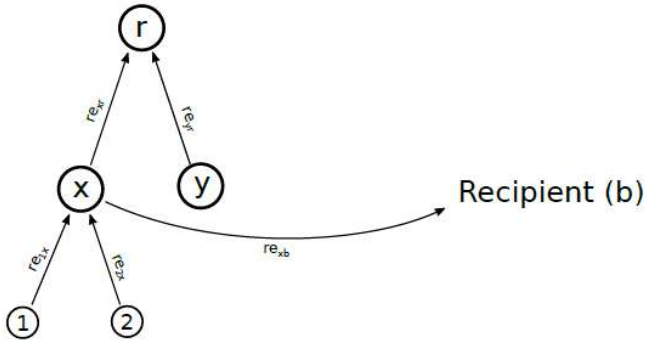


그림 6: 계층적 구조 공유(파일 또는 폴더)

D. 대량의 데이터 암호화

파일별 대칭 암호 키를 가지고 있을 때, 접근 권한이 있는 동안 악의적 수신자는 지나치게 많은 데이터를 다운 받지 않으면서도 모든 암호화된 대칭 키를 다운 받아 해독할 수 있다. 악의적 공격자는 나중에 이런 (미리 다운 받아 둔) 대칭키를 사용하여 대량의 데이터를 해독할 수 있다. 이것을 "키 스크래핑 공격"이라 한다.
이것이 문제되는 응용 프로그램에서 우리는 데이터에 "all-or-noting" 변환을 적용할 수 있고 따라서 공격자는 해독하기 위해서는 모든 파일을 다운받아야만 한다. 이 방법은 이미 발표되었고[35] 우리의 키 관리 시스템에 쉽게 사용될 수 있다.

E. 암호화된 스트림 공유

탈중앙화 된 넷플릭스 같은 응용 프로그램의 경우 서드파티의 라우팅 트레픽이 내용을 볼 권한이 없으면서 다수의 사용자에게 암호화된 스트림을 공유하는 것이 이상적이다. 원칙은 간단하다: 스트림의 각 데이터의 블록이 랜덤한 DEK로 암호화 되고, 채널별 공개키를 이용하여 EDEK가 생성되는 것이다. 하지만 NuCypher KMS로 왕복에 시간이 걸린다. 따라서 이전 블록에 다음 블록 데이터의 EDEK를 포함시키는 것이 합리적이다.
또한 미디어 콘텐츠 스트리밍의 실례로서 "키 스크래핑 공격"이 특히 문제가 된다. 따라서 우리는 영화 한 블록을 하나의 대량 데이터로 여기고 그에 따라 암호화해야 한다[35]. 보편적으로 스트림의 소비자들은 그들의 키 쌍을 가지고 있으며 채굴자 노드에게 EDEKs를 소비자들이 해독 가능하도록 요청한다. 채굴자들은 소비자가 구독을 위해 실제로 지불하였는지 확인할 수 있으며, 지불하지 않았다면 재암호화하기를 거부한다.

F. 시간 기반 그리고 조건 기반 정책들

암호화된 데이터 또는 키를 가지고 있는 채굴자를 신뢰하지 않기 때문에 재암호화 키의 저장 기간을 조절함으로써 우리는 그들을 신뢰할 수 있다. 가장 간단한 방법은 시간기반 정책이다. 재암호화는 주어진 일정 시간 동안 가능하도록 하며, 미래에 재암호화가 허용되지 않는다면 재암호화 키를 제거하는 것이다.
보다 복잡한 정책 역시 생성 가능하다. 조건 하에 재암호화를 허용하는 것이다. 예를 들어 특정 트랜잭션의 완성을 보류한다. 그럼으로써 콘텐츠 별 디지털 저작권 관리(DRM: Digital Rights Management) 또는 조건부 날인 증서(escrow)로써 비밀 데이터를 저장하는 응용 프로그램이 가능하도록 한다.

G. 키 회전

BBS98[15] 알고리즘을 포함한 많은 프록시 재암호화 알고리즘은 여러 번 적용될 수 있다. 따라서 프록시 재암호화는 키 회전에 사용할 수 있다. 키 회전은 오래된 키로 암호화된 모든 EDEKs을 새로운 키로 암호화 되는 것을 가능하게 한다.
데이터의 소유자는 두 가지 버전의 비밀 키 사이에서 재암호화 키 $re_{v1 \to v2}$ 를 생성해야하며, 이것의 핵심은 미래의 자신과 데이터를 공유하는 것이다. 이재 재암호화 노드는 (암

호화된 저장소는 IPFS같이 공개되어 있다는 가정하에) EDEKs를 다운로드하고 앞서 이야기 한 변환을 적용한다. 이 작업에서는 데이터를 보내는 사람과 받는 사람이 근본적으로 같기 때문에 결탁 위험이 없다.

Ⅵ. 스마트 컨트랙트 설계

재암호화 서비스를 제공하기 위해서, 노드는 자신의 보증금을 스마트 컨트랙트(락타임을 명시하여)로 보내야 한다. 시간이 만료되면, 노드는 자신의 지분을 빼낼 수 있다. 목적은 그들이 정확하게 재암호화 서비스를 했으면, 최근에 생겨난(minted) 보상을 스테이킹 노드가 가질 수 있도록 하기 위해서다.

초기 릴리즈에 한해서, 익명성을 요구하지 않고(III-A장) 노드가 온라인 상태를 유지하고 정확하게 데이터를 재암호화하는 것을 보장한다. 우리는 다른 네트워크에서 일어난 것처럼 노드가 전문화되고, 보안을 위해서 크게 투자할 것으로 기대한다. 따라서, 노드가 재암호화 서비스를 제공하기 위해서 s_{min}만큼의 최소 지분 요건을 만족해야한다. 이것은 DASH 암호화폐에서 마스터 노드의 지분을 가지는 메커니즘과 유사하게 노드의 수를 자연스럽게 제한할 것이다.

지분을 가지는 총량은 공개되어 있으므로, 네트워크의 클라이언트들은 노드에 재암호화 키를 배포할지 여부를 스스로 결정할 수 있다. 클라이언트인 앨리스는 재암호화 키를 무작위로 배포할 책임이 있다. 우리는 블록체인에서 앨리스의 무작위성에 대한 선택을 강요하지는 않는다.

i번째 채굴자의 활성(잠긴) 보증금을 d_i라고 하면 보증금의 총량은 다음과 같다.

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

여기서 k는 채굴자의 총 수다. i번째 채굴자가 한 번의 정책에서 재암호화 키를 얻을 수 있을 가능성은 다음과 같다.

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

여기서 n 은 우리가 재암호화 키를 나누기 위해 분할한 개수를 말한다(III-B장).

재암호화 키를 관리하기 위해서 임의의 노드를 선택하는 것은 다음과 같은 방식으로 진행된다(그림 7). 모든 노드는 랜덤하게 정렬되고, 좌표 x=0부터 길이가 D인 직선 구획에 배치된다. 따라서 i 번째 노드의 좌표는 다음과 같다.

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

이 방식대로 배치할 경우, 앨리스(Alice)는 임의의 수 n 을 선택할 수 있다.

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

여기서 수 r_i 은 보편성을 잃지 않고 정렬된 것으로 가정한다. 그렇다면 선택된 채굴자의 색인인 i_j 은 단지 해당하는 구획을 선택한 광부의 색인일 뿐이다.

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

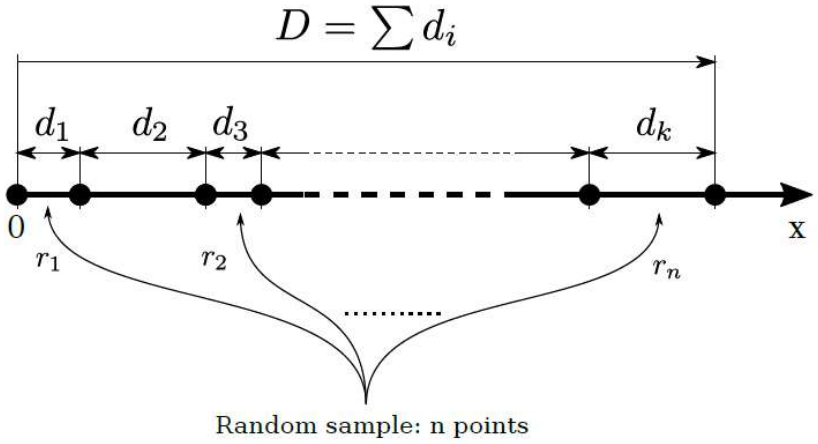


그림7. 지분 증명에 따라서 임의의 노드를 선택한다. 각 노드 di 의 지분은 직선 구획으로 표시된다. 각 구획은 길이가 D인 큰 구획으로 함께 쌓여 있다. 무작위 노드는 x=0 에서 x=D 사이의 임의의 지점에 선택된다.

일부는 구획에 해당하는 동일한 채굴자에게 떨어질(갈) 수도 있습니다. 이 경우, 앨리스는 i_j 의 반복을 제거하고, 원하는 채굴자의 수 n이 될 때까지 더 많이 선택 된 채굴자 색인 i_j 를 같은 방식으로 만든다.
가능한 테스트의 단위를 일반화 시켜 "핑(Ping)" 이라고 하자. 이 테스트는 다음을 포함한다.

- 실제 핑(아직 노드가 네트워크 상에서 보여지는가?)
- 오래되었지만 활성 상태인 재암호화 방식이 여전히 작동하는지 확인
- 생성 - 재암호화 - 취소 과정이 작동하는지 확인
- 채굴 노드와 관련이 없는 다른 주소에서 요청이 온 경우, 재암호화 정책이 앞의 작업(실제 핑, 재암호화 방식 작동 확인, 생성 - 재암호화 - 취소 작업 작동)이 작동하는지 확인

이 "핑"을 수행할 때, 가용성 판별에 필요한 테스트만 수행하도록 선택할 수 있다. 예를 들면, 노드가 오프라인 상태가 되면, (굉장히 가벼운) 실제 핑만으로 노드의 상태를 확인할 수 있다.

이제 우리는 모든 h 블록마다(예: h=10) 자체 검사를 수행하는 노드 시스템을 설계하고자 한다. 아이디어는 노드가 오작동하는 노드에 베팅하고, 추측이 맞는지 아닌지에 따라 채굴 보상을 받는 것이다. "올바른" 추측은 모든 노드의 투표에서 스마트 컨트랙트에 의해 계산 된 노드의 지분을 가지고 다수결로 결정되는 하나의 결과로 간주한다.

이 프로토콜이 각 h블록마다 모든 개별 노드를 검사한다면, 시스템 N의 모든 노드를 제대로 평가할 수 없을 것이다. 그래서 우리는 노드의 일부만 (예를 들면 k=N 중 일부) 만 확인한다. 예를 들면, 100,000개의 노드가 온라인 상태이고, 이더리움 블록 시간이 24초(현재 값)라면 모든 노드의 상태가 21시간마다 확인된다. 유효성을 검사 할 노드의 선택은 현재 유효한 노드가 임의의 바이트를 오프체인(off-chain)으로 교환하고, 이러한 임의 바이트의 해시를 정렬 및 결합하여 수행할 수 있다. 유효성 확인에 참여하지 않는 노드는 그들이 챌린지를 받고 있는지 실제 고객 요청을 받는지를 구분할 수 없다.

모든 노드가 상태 검사를 하는 것 또한 가능하지 않다. 따라서 노드의 일부(예를 들어 해밍 거리로 현재 블록의 해시에 가장 가까운 노드의) 현재 라운드에서 상태 확인에 대한 보상을 받을 수 있다. 노드가 정상적이지 않은 것으로 판명되면, 마지(막 상태 확인 이후 얻은 보상의 평균값을 처벌로 잃게 된다.

노드의 출력에서 재암호화의 정확성을 확인할 수 있다. 그러기 위해서 데이터 소유자는 민감한 데이터에는 해당하지 않지만, 특별히 재암호화 노드에 챌린지하기 위한 "챌린지 팩"을 준비할 수 있다. "챌린지 팩"은 입력 암호문과 예상 재암호화 출력값으로 구성된다(이 방법은 확률적 출력값을 가지지 않는 프록시 재암호화 알고리즘에서만 작동합니다). 데이터 수신자는 챌린지 팩을 해독하여 재암호화 노드가 실패하거나 정상적으로 데이터를 재암호화할 수 없음을 입증할 수 있다.

다른 재암호화 노드에 챌린지하는 노드는 필수적으로 어떤 노드가 잘못 작동하고 있는지를 투표에 붙이고 그 투표의 결과에 배팅해야 한다. 챌린지를 제기한 노드는 모든 사람이 투표를 하기 전에 다른 사람의 득표를 파악할 수 없어야 한다. 따라서 그들은 먼저 Salted 된 투표를 해시화하여 투표를 먼저 제시하고 추후 그들이 실제로 투표한 것을 Salt으로부터 그 해시를 만듦으로써 증명한다.

VII. 토큰 경제

프로토콜 경제는 희소한 자원 제공 작업에 기여하고 자원이 소비 될 때 보상받는 채굴

자들의 네트워크로 구성된다. NuCypher KMS에서 채굴자들은 재암호화 노드다. 누구나 채굴자가 될 수 있으며 보상은 제공된 재암호화 작업의 양에 따라 다르다. 희소한 재암호화 서비스에 대한 접근은 최고 가치 용도로서 할당되고 제어되어야 한다. 이를 달성하기 위한 방법으로 NuCypher KMS Token (NKMS) 토큰이 있다. NKMS는 작업에 공헌한 채굴자들에게는 보상이자 소비자(데이터 소유자)는 재암호화 서비스의 접근하기위한 비용이다. 필수적으로(Vitally), 토큰은 시스템의 연산 및 보안의 정확성을 장려한다.

A. 토큰 분배

채굴자는 스마트 컨트랙트로 잠긴 보증금으로서 가지고 있는 토큰의 수에 비례하는 재암호화 키를 가진다. 채굴자는 재암호화 서비스를 제공하고 데이터 재암호화에 가용하도록 함으로써 지불(*토큰)을 받는다. 채굴자는 데이터 소유자뿐만 아니라 데이터 사용자에게도 지불받을 수 있다. 후자의 경우는 탈중앙화 자율 조직(DAO)가 분산형 디지털 저작권 관리(DRM)에서 콘텐츠를 배포하는데 사용할 수 있다.

또한, 채굴자들은 그들이 항상 온라인 상태에 있으면서 서비스를 제공하기 위한 준비가 되어 있고, 스마트 컨트랙트(VI장)로 증명이 확인될 경우 생성되는 "가용 보상(availability rewards)"을 받을 수 있다. 만약 채굴자가 가용상태가 아닌 경우, 그 시간 동안의 "가용" 보상을 잃게 된다.

채굴자가 부정을 저지르거나 잘못된 재암호화를 제공할 경우, 보안 예치금의 일부를 잃게 된다(IIIC장).

우리는 재암호화 키를 유출하는 채굴자에게 불이익을 줄 수 있다. 누구나 재암호화 키의 해시로 챌린지를 제기할 수 있고, 재암호화 키가 누출되었다는 것으로 밝혀질 경우 채굴자는 NKMS(*토큰단위)로 매겨진 담보금을 상실한다. 재암호화 키가 암호화 단계에서 누출된 경우 그것을 신속하게 확인하기 위해서 재암호화 키와 특정 프록시와 연결할 수 있다[37].

시스템에 사용자가 많지 않은 경우, 초기에 재암호화 작업에 인센티브를 부여하는 것은 중요하다. 그것이 우리는 시간이 지남에 따라 점진적으로 감소하는 (0으로 가는) "채굴되는" 보상 계획을 도입하는 이유이다. 채굴자 i가 얻는 보상 r_i 은 다음과 같이 표현될 수 있다.

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

여기서 s는 현재 보상률, d_i는 i번째 채굴자의 예치금이며, f_i 는 이 채굴자에 대한 총 트

랜잭션 수수료이다. 보상률은 지수적으로 감소하여 무한한 시간이 지났을 때와 평형상태에 이르면, 수수료 전용 보상으로 전환된다.

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

여기서 T는 보상이 e의 계수로 떨어지는(*지수적으로 감소하는) 시간이고, s_0는 초기 보상률이다. 이 지수는 미분방정식의 해로써 값 비싼 유동소수점 계산 없이 스마트 컨트랙트 내에서 쉽게 얻을 수 있다.
여기서 S_{max} 는 t = ∞일 때의 최대 코인 제공량이고, S(t = 0)은 초기 코인 제공량이다. 여기서 시간 t초단위가 아닌 단위채굴시간으로 표현된다.

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

NuCypher KMS의 흥미로운 속성은 네트워크 참여자의 수가 증가함에 따라 보안이 향상된다는 것이다. 추가적인 재암호화 노드가 네트워크에 진입하면 결탁의 가능성이 낮아진다. 이를 통해 시스템의 보안 및 검열에 대한 저항력이 향상되어 강력한 네트워크 효과와 최초 참가자의 이득이 유의미하게 제공될 수 있다.
마지막으로, 네트워크의 보안을 테스트하기 위해 우리는 지갑에 개인 키를 암호 화폐와 함께 넣을 수도 있다. 누구든지 시스템을 해킹하여 현상금을 가져갈 수 있다. 현상금의 지급 사실은 보안취약의 증거가 될 것이다. 이와 동일한 메커니즘을 네트워크의 개별 사용자가 데이터에 키를 노출했음을 알리는 신호를 보내 잠재적인 네트워크 내 개별 사용자에 의한 데이터 유출을 경고 할 수 있다.

VIII. 사용 사례

NuCypher KMS는 기본적으로 민감한 데이터를 공유하는 것을 요구하는 여러 애플리케이션들을 위한 인프라를 제공한다. 특정 메시지 게시, 특정 당사자 간의 지불 및 기타 이벤트와 같이 합의 네트워크에서 공개 작업에 대한 해독 작업을 규정 할 수 있는 기능을 사용하면 다음과 같은 다양한 애플리케이션을 구현할 수 있다:

A. 암호화된 파일 공유("탈중앙화 Dropbox")

파일들을 클라이언트 사이드에서 암호화 시키고, Swarm [38], IPFS [39], Sia [40], or Storj [41]같은 탈중앙 파일시스템 또는 S3같은 중앙형 서버에 저장을 할 수 있다. 이 파

일들은 자기들의 공유 키를 기반으로 한 재암호화 토큰을 제공하는 공인된 서드파티들을 이용해서 쉽게 공유할 수 있다. 이 서드파티의 접근 권한은 네트워크에서 재암호화 토큰을 제거함으로써 쉽게 폐기할 수 있다.

B. 양단 간 암호화 그룹 채팅("암호화 Slack")

프록시 재암호화(PRE)는 이상적인 양 끝 사이의 프리미티브 이며, 여러 사용자들이 채널에 접근하여 읽고 쓸 수 있도록 하는 요구사항을 충족한다. 재암호화 토큰을 발행하거나 폐기함으로써 대화 멤버를 쉽게 추가되거나 제거할 수 있다. 이것은 각 참가자들에게 여러 번 메시지를 암호화하고 보내야 하는 오버헤드를 줄여준다.

C. 환자가 관리하는 관리 전자건강기록(Electronic Health Records - EHR)

Epic과 같은 중앙 집중형 시스템과 정반대로 환자가 관리하는 전자건강기록을 작성하여 환자가 데이터 및 암호화 키를 소유 할 수 있다. 데이터는 다시 중앙 집중형 또는 탈중앙 백엔드 서버에 저장할 수 있다. 환자가 암호화 된 데이터를 병원이나 보험 회사와 공유하기를 원하면, 재암호화 토큰을 발급하여 제 3 자에게 임시 액세스 권한을 부여할 수 있다.

D. 분산 디지털 권한 관리 (Decentralized digital rights management - DDRM)

암호화 액세스 제어는 탈중앙 디지털 저작권관리(DRM)의 역할을 맡을 수 있다. 접근 권한 제어는 자체적으로 암호화된 채로 내장되어 어디서나 데이터가 가는 대로 따라간다. 조건부 재암호화 토큰은 스마트 컨트랙트에 의해 제어되고 지불시에만 릴리즈 할 수 있다. 이제 NuCypher KMS를 사용하여 탈중앙 Netflix 또는 소프트웨어, 응용 프로그램, 사진 및 기타 디지털 콘텐츠를 판매하는 암호화된 시장과 서비스를 구축할 수 있다.

E. Blind identity 관리

블라인드 아이덴티티 관리 서비스는 NuCypher KMS를 이용해서 구축 할 수 있다. 아이덴티티는 고객측에서 암호화 될 수 있고 아이덴티티 관리 제공자에 저장 된다. 사용자는

허용된 응용 프로그램을 위한 재암호화 키를 생성할 수 있다. 이 서비스는 아이덴티티 제공자가 접근 권한을 가지지 않고서도 서드파티 응용 프로그램에 대하여 아이덴티티 인증서를 재암호화할 수 있다.

F. 스크립트와 백엔드 응용 프로그램을 위한 비밀 인증서 관리

NuCypher KMS는 민감한 환경 변수, 인증서 데이터베이스 그리고 API키 같은 비밀을 저장하는데 이상적인 저장소다. 스크립트의 경우 스크립트 기간 동안 재암호화 토큰을 생성 한 다음 해지 할 수 있다. 한가지 예로써 개발자는 암호화 된 데이터베이스 인증서를 GitHub에 안전하게 저장할 수 있으므로 인스턴스가 배포 된 후 이러한 자격 증명에 임시 액세스 할 수 있다. GitHub 레포지토리가 퍼블릭이더라도 자격이 없는 사람은 인증서를 사용할 수 없다.

G. 공유된 인증서와 기업 비밀번호 관리

NuCypher KMS는 직원이 웹 서비스에 접근하기 위해 공유된 인증서 를 관리할 수 있다. 어떤 비밀에 누가 액세스 하는지를 모니터하기 위해 검사 로그를 구축 할 수 있다. 직원이 떠나면 액세스 권한을 취소하거나 키를 재사용하기 쉽다.

H. 강제적 접근 로깅

몇몇 기업과 회사의 설정에서 사용자는 민감한 파일의 접근 로그를 남겨야만 한다. 이는 각 파일에 대한 접근을 기록되어야 하며 조건적 재암호화가 이러한 로깅 규칙을 위임받아 작동할 수 있다.

I. 모바일 장비 관리 그리고 폐지

기업 모바일 장비 관리 설정에서 유효한 장비를 위해 재암호화 토큰을 생성할 수 있다. 장비를 잃어버리거나 수명이 다하거나 또는 직원이 조직을 떠날 경우 재암호화 토큰을 삭제함으로써 장비의 접근 권한을 폐지 할 수 있다.
이것은 계층적 키 트리를 재구성해야하는 문제를 피하게 한다.

J. NuCypher KMS의 프라이빗 사용

전통적인 기업들(금융, 헬스케어, 정부 또는 사물인터넷 기업들)에게 이익이 되도록 하기 위해선 프라이빗 으로도 배포 가능해야 한다. 프라이빗(또는 컨소시엄) 블록체인을 사용 할 수 있으며, 연합되거나(채굴자들이 기업의 컨소시엄에 의해 사전 선택된 노드들로 대체된) 심지어는 중앙화(오직 하나의 집단이 모든 재암호화 키를 관리하는)될 수 있다. 이런 사용법들은 프록시 재암호화의 이점을 살릴 수 있지만 검열 저항(*censorship resistance)은 그전처럼 효과적이지 않다.

IX. 요약

NuCypher KMS는 탈중앙화 키 관리 서비스이며, 블록체인 및 탈중앙 응용 프로그램용 권한 관리 레이어이다. 개발자와 기업 모두 의료, 금융 서비스 등에서 매우 안전한 응용 프로그램을 개발하는 데 활용할 수 있다. 개인의 데이터를 퍼블릭 블록체인에 공유하고 계산함으로써, NuCypher KMS는 암호화 된 콘텐츠 시장부터 비밀 자격 증명 관리, 환자가 제어하는 전자 건강 기록까지 모든 것을 가능하게 한다.

X. 감사의 말

Stevens 공대의 Giuseppe Ateniese와 Málaga 대학의 Isaac Agudo Ruiz 께서 프록시 재암호화 알고리즘에 대한 도움을 주신 것에 대해 감사의 말씀을 드린다. 또한 평생 동안 우리의 회사에 조언을 해 주고, 최신 다자간 계산 기술에 대한 도움을 주신 Virginia 대학의 Dave Evans 에게도 감사의 말씀을 드린다. 그리고 Stefano Bernardi, Tom Ding, 그 외에 토큰 경제에 도움을 준 많은 사람들에게 감사의 인사를 표한다.

- Gabriel Kaptchuk, Ian Miers, and Matthew Green, "Managing secrets with consensus networks: Fairness, ransomware and access control," Cryptology ePrint Archive, Report 2017/201 (2017).
- Wikipedia, "Proxy re-encryption — Wikipedia, the free encyclopedia," (2017).
- Wikipedia, "Key management — Wikipedia, the free encyclopedia," (2017).
- Wikipedia, "Hardware security module — Wikipedia, the free encyclopedia," (2017).
- HashiCorp Inc., "Vault by hashicorp," (2017).
- Amazon Inc., "Aws cloudhsm," (2017).
- Alphabet Inc., "Cloud key management service," (2017).
- Microsoft Inc., "Key vault," (2017).
- TrueVault Inc., "Hipaa compliant api & secure data store," (2017).
- Wikipedia, "Advanced encryption standard — Wikipedia, the free encyclopedia," (2017).
- Wikipedia, "Salsa20 — Wikipedia, the free encyclopedia," (2017).
- 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).
- NuCypher, "Modern data security for hadoop," (2017).
- "Umbral: a threshold proxy re-encryption scheme," .
- 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.
- 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).

- Wikipedia, "Ntruencrypt — Wikipedia, the free encyclopedia," (2017).
- 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.
- 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).
- Wikipedia, "Chosen-plaintext attack — wikipedia, the free encyclopedia," (2017).
- Wikipedia, "Chosen-ciphertext attack — wikipedia, the free encyclopedia," (2017).
- Vitalik Buterin, "Secret sharing daos: The other crypto 2.0," (2014).
- Guy Zyskind, Oz Nathan, and Alex Pentland, Enigma: Decentralized Computation Platform with Guaranteed Privacy, Tech. Rep. (2015).
- 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).
- Jason Teutsch and Christian Reitwießner, "A scalable verification solution for blockchains," (2017).
- Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and ₩Lukasz Mazurek, "Secure Multiparty Computations on Bitcoin," Commun. ACM **59**, 76–84 (2016).
- 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).
- Giuseppe Ateniese, "Accountable Storage," , 1–18.
- Xiaodong Lin and Rongxing Lu, "Proxy Re-encryption with Delegatable Verifiability," (2016) pp. 120–133.
- Consensys, "Introduction to zk-SNARKs with examples," (2017).
- 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).

- Wikipedia, "Software guard extensions — wikipedia, the free encyclopedia," (2017).
- 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).
- 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).
- 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.
- Evan Duffield and Daniel Diaz, "Dash: A privacy-centric crypto-currency," (2015).
- 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).
- Victor Trón, Aron Fischer, and Daniel Varga, "Smash-proof: auditable storage for swarm secured by masked audit secret hash," (2016).
- Juan Benet, "IPFS — content addressed, versioned, P2P file system," (2014).
- Nebulous Inc., "Sia: Simple decentralized storage," (2014).
- Storj Labs Inc., "A peer-to-peer cloud storage network," (2016).