종단간 암호화, 진짜 설명

제공업체가 E2EE라고 말할 때 하는 말과 하지 않는 말. 광고 문구를 걷어낸 메커니즘과 그 한계에 대한 교육적 설명.

확실히 해둡시다: WhatsApp은 귀하의 메시지가 종단간 암호화되어 있다고 말합니다. 사실이지만, 그것만으로는 충분하지 않습니다. 추가 암호화 없이 백업이 iCloud나 Google Drive로 이동한다면, 암호화는 귀하의 휴대폰에서 깨지는 것입니다. 실무적인 질문은 암호화 여부가 아니라 키가 어디에 있느냐입니다.

암호화가 진짜 의미하는 것

메시지를 암호화한다는 것은 키라고 불리는 특정 정보를 가지고 있지 않은 누구에게나 노이즈처럼 보이는 것으로 변환하는 것을 의미합니다. 이 작업은 발신자의 장치에서 수행되며, 올바른 키를 사용하면 수신자의 장치에서 원래대로 복구됩니다. 그 사이에서 메시지는 겉으로 드러나는 의미가 없는 바이트의 연속으로 전송됩니다. 이것이 단순한 개념입니다. 기사의 나머지 부분에서는 경우에 따라 이것이 실제 보증이 될지 아니면 단순한 마케팅 라벨이 될지를 결정 짓는 뉘앙스에 대해 다룹니다.

종단간 — 영어로 *end-to-end*, 줄여서 E2EE — 이라는 형용사는 정밀함을 더합니다. 암호화는 중간 서버가 이를 읽고 전달할 수 있도록 하기 위해 수행되는 것이 아닙니다. 발신자의 장치와 수신자의 장치라는 두 끝단만이 키를 소유하도록 수행됩니다. 메시지가 통과하는 모든 서버는 메시지가 아닌 노이즈를 보게 됩니다. 이것이 콘텐츠가 한 서버에서 다음 서버로 암호화되어 이동하지만, 통과하는 각 서버가 전달을 위해 이를 복호화하여 일시적으로 평문을 복구하는 *전송 중(in transit)* 암호화와의 기술적 차이점입니다.

공유 비밀의 역설

명백한 문제가 있습니다. 두 사람이 서로 메시지를 암호화하고 복호화할 수 있으려면 두 사람 모두 동일한 키가 필요합니다. 하지만 정의상 누군가 엿들을 수 있는 채널을 통해 서로 보내는 모든 것이 전달되는데, 어떻게 이 키에 합의할 수 있을까요? 나중에 키를 사용할 동일한 채널에서 키를 합의하는 것은 불가능해 보입니다. 공격자가 합의하는 시점에 이를 듣게 된다면 이후의 모든 것을 복호화할 수 있기 때문입니다. 수십 년 동안 고전 암호학은 이를 어려운 방식으로 해결했습니다. 키는 사용을 시작하기 전에 물리적인 만남을 통해 직접 전달되었습니다. 대사들은 코트 안감에 꿰매어 놓은 키 가방을 가지고 다녔습니다.

현대의 이메일 환경에서 그 솔루션은 확장성이 없습니다. 우리가 암호화된 통신을 하려는 모든 사람의 집을 물리적으로 방문해야 한다면, 누구와도 대화할 수 없을 것입니다. 50년 전 암호학 커뮤니티가 던진 질문은 이것이었습니다. 서로를 모르고 공개 채널만 공유하는 두 사람이 그 동일한 공개 채널을 통해 채널을 듣고 있는 누구도 알 수 없는 비밀에 합의하는 것이 가능한가?

Diffie-Hellman의 우아함

1976년 Whitfield Diffie와 Martin Hellman이라는 두 수학자는 겉보기에 불가능해 보이는 것을 증명했습니다. 공개 채널(누구나 그들이 말하는 모든 것을 들을 수 있는 채널)만을 통해 대화하는 두 사람이 청취자가 이를 알아내지 못하게 하면서 비밀번호에 합의할 수 있다는 것입니다. 마법처럼 들리지만 그렇지 않습니다. 그것은 수학입니다. 그 이후로 알려진 Diffie-Hellman 키 교환은 사실상 인터넷의 모든 암호화 통신의 기초가 되었으며, 반세기에 걸친 집중적인 사용과 전 세계적인 학술적 조사가 그 견고함을 확인해 줍니다. 시각적 직관이나 수학을 보고 싶은 분은 계속 읽어주시기 바랍니다. 그것이 작동한다는 것을 믿고 싶은 분은 기사의 흐름을 잃지 않고 계속 진행하셔도 좋습니다.

이미지로 이해하고 싶은 분들을 위해 색상을 이용한 유명한 비유가 있습니다. 앨리스와 브루노가 그들을 엿듣는 에바 앞에서 공개적으로 기본 색상(예: 노란색)에 합의한다고 상상해 보십시오. 각자 비공개로 두 번째 비밀 색상을 선택하고 자신의 비밀을 노란색과 섞습니다. 앨리스는 특정한 주황색을 얻고, 브루노는 특정한 초록색을 얻습니다. 그들은 에바 앞에서 결과를 교환합니다. 이제 각자 받은 색상을 자신의 비밀과 섞으면, 섞는 순서는 상관없기 때문에 두 사람 모두 동일한 최종 색상에 도달하게 됩니다. 에바는 노란색과 두 개의 중간 혼합물을 보았지만 비밀은 보지 못했

습니다. 비밀 중 하나라도 없으면 최종 색상에 도달할 수 없습니다. 실제 수학은 색상을 모듈러 군 또는 타원 곡선에서의 거듭제곱으로 대체하지만, 개념은 동일합니다. 공유 비밀은 채널에 있는 누구도 재구성할 수 없는 상태에서 공개적으로 구축됩니다.

산술에서, 메커니즘을 보고 싶은 분들을 위해: 앨리스는 비밀 숫자 a를 선택하고, 브루노는 b를 선택합니다. 그들은 채널을 통해 공개적으로 g^a와 g^b를 교환합니다. 앨리스는 $(g^b)^a$를 계산하고 브루노는 $(g^a)^b$를 계산합니다. 두 사람 모두 동일한 g^{ab}에 도달합니다. 에바는 g, g^a, g^b가 채널을 통과하는 것을 보지만, g^a에서 a를 복구하는 것(이른바 이산 로그 문제)은 g가 적절한 수학적 군에서 선택되었을 때 우주의 나이보다 훨씬 더 긴 천문학적인 계산 시간을 필요로 합니다.

Para quien quiera comprobarlo con números pequeños. El intercambio Diffie-Hellman se puede recorrer entero con cifras lo bastante reducidas como para hacer las cuentas a mano. Quien prefiera no entrar en aritmética puede saltarse este bloque sin perder el hilo del artículo; quien quiera ver el mecanismo funcionando paso a paso lo encontrará aquí. **Las reglas públicas**, que cualquiera puede leer: un primo $p = 11$ (en el Diffie-Hellman real es de unas trescientas cifras; usamos once para que las cuentas quepan en una página), una base $g = 2$, y la convención de que toda la aritmética se hace *módulo p* — se calcula, se divide entre p, y se conserva el resto, como un reloj de once posiciones que vuelve al cero al rebasar el diez. **Las elecciones privadas**, una cada uno y jamás compartidas: Alicia elige $a = 4$. Bruno elige $b = 7$.

Paso 1. Alicia calcula $2^4 = 16$, luego $16 \bmod 11 =$ **5**. Envía el cinco. Eva lo anota.

Paso 2. Bruno calcula $2^7 = 128$, luego $128 \bmod 11 =$ **7**. Envía el siete. Eva también lo anota. Tras los dos envíos, la libreta de Eva contiene cuatro datos: $p = 11$, $g = 2$, A = 5, B = 7. Le falta el número compartido que Alicia y Bruno están a punto de derivar — y que Eva no podrá reconstruir.

Paso 3. Alicia toma el siete que Bruno le envió y lo eleva a su exponente privado $a = 4$. Para evitar manejar $7^4 = 2401$, se calcula por partes aplicando el módulo en cada paso:

$7^2 = 49$

$49 \bmod 11 = 5$

$7^4 = (7^2)^2 = 5^2 = 25$

$25 \bmod 11 = 3$

Alicia obtiene el número **3**.

Paso 4. Bruno toma el cinco que Alicia le envió y lo eleva a su exponente privado $b = 7$. De nuevo por partes:

$5^2 = 25 \bmod 11 = 3$

$5^4 = (5^2)^2 = 3^2 = 9 \bmod 11 = 9$

$5^6 = 5^4 \times 5^2 = 9 \times 3 = 27 \bmod 11 = 5$

Finalmente $5^7 = 5^6 \times 5 = 5 \times 5 = 25 \bmod 11 =$ **3**.

Bruno obtiene también **3**.

Los dos han llegado al mismo número, 3, trabajando en paralelo. Ninguno envió su exponente privado en ningún momento. Alicia no sabe que $b = 7$; Bruno no sabe que $a = 4$. Cada cual usó el valor público que el otro envió combinado con su propio exponente privado, y se encontraron en el mismo destino. **¿Por qué llegan al mismo número?** Lo que calculó cada uno: Alicia, $(g^b)^a = 2^{7\times4} = 2^{28} \bmod 11$. Bruno, $(g^a)^b = 2^{4\times7} = 2^{28} \bmod 11$. Es la misma cantidad porque el orden de multiplicación de exponentes no importa ($7 \times 4 = 4 \times 7$). Cada cual llegó por un camino distinto al mismo destino.

¿Y Eva? Tiene en su libreta $p = 11$, $g = 2$, A = 5, B = 7, y quisiera el 3. Para calcularlo necesitaría conocer a o b — pero ninguno ha viajado por el canal. Su única vía es preguntarse: «¿para qué exponente a se cumple $2^a \bmod 11 = 5$?». Con p tan pequeño puede probar 0, 1, 2, 3, 4… y encontrarlo en menos de un minuto. Pero al sustituir 11 por un primo de trescientas cifras, el espacio de exponentes posibles tiene más elementos que átomos hay en el universo observable. **No existe a día de hoy ningún algoritmo conocido por la humanidad que pueda recorrer ese espacio en menos de miles de millones de años.** Es el llamado *problema del logaritmo discreto*: fácil hacia adelante, computacionalmente imposible hacia atrás. Y es la razón por la que el cifrado resiste aunque Eva haya seguido toda la conversación letra por letra.

Tres ingredientes simples —aritmética sobre un reloj, exponenciación, y conmutatividad de la multiplicación ($a \cdot b = b \cdot a$)— combinados producen un protocolo del que media humanidad depende cada día para sus comunicaciones privadas. Ninguna de las tres piezas, por separado, parece especial. Lo decisivo es el ensamblaje.

Diffie-Hellman에서 Signal 프로토콜까지

오늘날 전문 메시징 애플리케이션이 사용하는 종단간 암호화는 거의 예외 없이 Diffie-Hellman 교환의 우아하고 강화된 버전에 의존합니다. 2013년에서 2016년 사이에 Trevor Perrin과 Moxie Marlinspike가 설계한 Signal 프로토콜이 참조 기준입니다. 이는 두 가지 핵심 아이디어를 결합합니다. 첫째, 두 장치 사이의 초기 공유 비밀을 생성하는 타원 곡선(X25519)에서의 키 교환입니다. 둘째, 메시지마다 자동으로 키를 갱신하는

Double Ratchet(더블 라쳇)이라 불리는 것입니다. 이를 통해 오늘 장치가 침해되더라도 과거의 메시지를 복호화할 수 없으며, 라쳇이 회전한 후의 미래 메시지도 복호화할 수 없습니다.

Zig에서 두 장치 사이의 공유 비밀을 생성하는 X25519 교환은 표준 라이브러리를 사용하여 6줄 안에 들어갑니다.

```
const std = @import("std");
const X25519 = std.crypto.dh.X25519;

// Alicia y Bruno generan cada uno un par (privada, pública).
const par_alicia = X25519.KeyPair.generate(io);
const par_bruno  = X25519.KeyPair.generate(io);

// Cada parte recibe la clave pública de la otra y deriva el mismo secreto.
const secreto_alicia = X25519.scalarmult(par_alicia.secret_key, par_bruno.public_key) catch unreachable;
const secreto_bruno  = X25519.scalarmult(par_bruno.secret_key,  par_alicia.public_key) catch unreachable;
// secreto_alicia == secreto_bruno  (32 bytes)
```

그 6줄에서 일어나는 일: 공개 키는 공개적으로 전송됩니다. 개인 키는 각각의 장치를 절대 떠나지 않습니다. 각 당사자는 자신의 개인 키와 상대방의 공개 키로부터 채널의 누구도 복구할 수 없는 동일한 32바이트 비밀을 도출합니다. 그 비밀은 나중에 교환되는 메시지를 암호화하기 위한 시드로 기능합니다. Signal 프로토콜의 Double Ratchet은 해당 재료에 지속적인 회전을 추가하여 한 순간의 침해가 대화의 나머지 부분을 침해하지 않도록 합니다.

그렇다면 `std.crypto.dh.X25519` 내부에는 정확히 무엇이 있을까요? 숨겨진 마법은 없습니다. Zig의 자체 표준 라이브러리에서 전체를 읽을 수 있는 두 개의 짧은 함수입니다. 첫 번째는 개인 키에서 공개 키 — 교환의 «g^a» — 를 파생시킵니다.

```
pub fn recoverPublicKey(secret_key: [secret_length]u8) IdentityElementError![public_length]u8 {
    const q = try Curve.basePoint.clampedMul(secret_key);
    return q.toBytes();
}
```

기사의 표현을 빌리자면, 개인 키는 기초 산술적인 의미가 아니라 타원적인 의미에서 Curve25519 곡선의 기본점(base point)과 «곱해지며», 그 결과는 32바이트로 직렬화(serialize)됩니다. `clampedMul` 연산은 해당 스칼라 곱셈의 강화된 버전입니다. 알려진 공격 제품군에 저항하기 위해 암호화 커뮤니티가 수년에 걸쳐 추가한 보호 장치들을 통합하고 있습니다. 함수 본문은 2줄입니다.

두 번째 함수는 귀하의 개인 키를 상대방이 보낸 공개 키와 결합합니다. 이것이 교환의 «$(g^b)^a$»이며, 두 사람 모두 결코 전송하지 않은 32바이트의 공유 비밀을 생성합니다.

```
pub fn scalarmult(secret_key: [secret_length]u8, public_key: [public_length]u8) IdentityElementError![shared_length]u8 {
    const q = try Curve.fromBytes(public_key).clampedMul(secret_key);
    return q.toBytes();
}
```

2줄이 더 있습니다. 수신된 공개 키는 곡선 위의 점으로 해석되며, 자신의 개인 키와 «곱해집니다». 곡선 연산의 교환 법칙(숫자 예시에서 보았던 지수 곱셈의 교환 법칙과 유사)에 의해 양 당사자는 직렬화된 동일한 점, 즉 이 기사에서 이야기하는 공유 비밀에 도달하게 됩니다.

이게 전부입니다. 애플리케이션에서 마법처럼 보이는 것은 실제로는 각각 3줄짜리 함수 2개일 뿐입니다. 기술적 복잡성은 `clampedMul`이라는 단일 연산에 집중되어 있습니다. 이 연산은 같은 표준 라이브러리 아래쪽에 작성되어 있으며, 국제 암호화 커뮤니티에서 수십 년 동안 검토해 왔고, 글자 하나하나 읽어보고 싶은 사람이라면 누구나 이용할 수 있습니다. 우리 애플리케이션이나 Zig의 표준 라이브러리에는 블랙박스가 없습니다. 사람이 원할 때 파고들 속도를 선택하여 이해할 수 있는 오픈 소스 코드가 존재할 뿐입니다.

종단간 암호화가 보호하는 것

올바른 구현을 가정할 때 E2EE가 잘 보호하는 것은 전송 중인 메시지의 내용입니다. 암호화된 데이터를 수신하고 전달하는 중간 서버에는 이해할 수 없는 바이트의 연속이 보일 것입니다. 케이블, 라우터, Wi-Fi 액세스 포인트에 접근할 수 있는 공격자도 똑같은 것을 보게 될 것입니다. 통신 사본을 보관하는 서비스 제공업체도 나중에 이를 읽을 수 없습니다. 서비스 운영자에게 콘텐츠 제출을 명령하는 정부도 서버가 처음에 가지고 있던 것과 동일한 이해할 수 없는 바이트를 받게 될 것입니다.

이는 실무적인 관점에서 매우 큰 의미를 갖습니다. 불투명한 봉투 안에 편지를 쓰는 것과 엽서에 쓰는 것의 차이입니다. 둘 다 도착합니다. 하지만 우체부로부터 내용을 보호하는 것은 하나뿐입니다.

종단간 암호화가 보호하지 않는 것

마찬가지로 알아두어야 할 사항이 있습니다. E2EE는 메타데이터를 보호하지 않습니다. 서버는 사용자 A가 사용자 B에게 몇 시에, 얼마나 자주, 어디서 데이터를 보내는지에 대해 무엇을 말하는지는 모르더라도 여전히 알고 있습니다. 이러한 메타데이터는 이전에 *「암호화하는 것」이 「프라이버시를 지키는 것」은 아니다*에서 논의했듯이 종종 내용보다 더 많은 것을 드러냅니다. 누군가 금요일 밤 22:00에 이혼 전문 법률 사무소에 30분 동안 전화했다는 사실을 아는 것은 통화 내용이 결코 말하지 않은 이야기를 들려줍니다. 이것은 어떤 사람이 종양 클리닉을 여러 번 드나드는 것을 보는 것과 같은 상황입니다. 안에서 어떤 대화가 오가는지 듣지 않아도 무슨 일이 일어나고 있는지 상상하기에 충분합니다. 단일한 고립된 메타데이터는 아무런 의미가 없을 수 있지만, 여러 데이터가 교차하면 진실과 너무나 흡사한 무언가가 그려집니다. E2EE는 끝단(endpoints)을 보호하지 않습니다. 수신자의 장치가 악성 프로그램에 의해 침해되었다면 메시지는 해당 수신자에게 정상적으로 복호화되고 악성 프로그램이 이를 읽습니다. E2EE는 대화 상대방의 신원 자체를 보호하지 않습니다. 앨리스가 브루노와 대화하고 있다고 믿더라도 공격자가 처음에 개입했다면(*중간자(man in the middle) 공격*), 그리고 프로토콜에 독립적인 검증이 포함되어 있지 않다면 두 당사자는 서로 대화하고 있다고 생각하면서 침입자와 대화하게 됩니다.

모호함 없이 명확히 해두어야 할 네 번째 사항이 있습니다. E2EE는 이를 제공한다고 주장하는 제공업체가 자신의 시스템에 암호화되지 않은 메시지의 사본을 추가로 보관하는 것을 막지 않습니다. 「내 메시지는 종단간 암호화되어 있다」는 주장과 「제공업체는 내 콘텐츠를 보관하지 않는다」는 주장은 동일하지 않습니다. 애플리케이션은 두 번째 주장을 위반하면서 첫 번째 주장을 충족할 수 있습니다. 우리는 2018년 이후 뉴스 헤드라인에서 이를 반복해서 보아왔습니다. 사용자는 클라이언트 코드를 검증할 수 없는 한 전문가의 조사 없이는 한 경우와 다른 경우를 기술적으로 구별할 방법이 없습니다. 일반 대중에게 가장 잘 알려진 사례: WhatsApp은 전송 중에 메시지를 종단간 암호화하지만, 사용자가 추가 암호화 없이 iCloud나 Google Drive에서 백업을 활성화하면 해당 사본은 제3자의 인프라에 읽을 수 있는 상태로 저장되며, 암호화는 사용자 자신의 끝단에서 깨지게 됩니다.

운영자가 듣고 싶어 하지 않는 질문

종단간 암호화를 한다고 주장하는 애플리케이션은 기술적으로 키와 관련하여 다음 세 가지 중 하나를 수행할 수 있습니다.

1. **키는 장치에만 존재합니다.** 키는 오직 사용자의 장치에서만 생성되고 존재합니다. 운영자는 키를 알지 못하며 저장하지도 않습니다. 이것이 최적의 사례입니다.
2. **운영자가 원하면 액세스할 수 있습니다.** 운영자가 사용자의 키를 보유하고 있거나(원하는 대로 생성할 수 있거나) 자신의 데이터베이스에 저장합니다. 원하거나 강제될 경우 내용을 읽을 수 있습니다. 대부분의 '클라우드' 서비스가 이 경우에 해당합니다.
3. **설계상 액세스할 수 없지만, 액세스 권한을 제어합니다.** 운영자가 키를 가지고 있지는 않지만, 키를 생성하는 애플리케이션을 제어합니다. 강제될 경우 암호화 전의 키나 내용을 캡처하는 악성 업데이트를 보낼 수 있습니다. 많은 상용 E2EE 서비스가 이 경우에 해당합니다.

따라서 실무적인 질문은 암호화 여부가 아니라, 장치와 키를 관리하는 소프트웨어를 누가 제어하느냐입니다. Solo2에서 키는 오직 귀하의 '금고'(귀하의 비밀번호로 암호화된 IndexedDB)에만 존재하며 소프트웨어는 검증 가능한 오픈 소스 코드입니다.

전문 독자를 위하여

종단간 암호화는 디지털 주권을 위한 도구입니다. 하지만 모든 도구와 마찬가지로 그 효과는 그것을 휘두르는 손과 그것이 딛고 선 땅에 달려 있습니다.

1. 암호화 키는 어디서 생성되며 물리적으로 어디에 존재합니까? 운영자가 (심지어 일시적으로라도, 복구를 구실로라도) 키에 접근할 수 있다면 E2EE는 명목상일 뿐입니다.
2. 대화를 설정하는 동안 중간자 공격을 방지하는 대화 상대에 대한 독립적인 검증(보안 번호, QR 코드, 대역 외 비교)이 있습니까?
3. 클라이언트 코드는 감사 가능(공개, 게시, 재현 가능)합니까, 아니면 클라이언트가 실제로 수행하는 작업에 대해 제공업체의 말을 신뢰해야 합니까?
4. 서비스는 어떤 메타데이터를 생성 및 유지하며, 기간은 얼마나 됩니까? 콘텐츠가 불투명하더라도 메타데이터는 민감한 정보의 상당 부분을 재구성할 수 있습니다.

이 네 가지 질문은 고급 기술 정보를 요구하지 않습니다. 정직한 운영자라면 공개 문서에서 답할 수 있는 정보를 요구합니다. 답변의 질과 정확성은 답변 자체만큼이나 제품에 대해 많은 것을 말해줍니다.

종단간 암호화는 제대로 구현될 경우 현대 암호학이 일상의 실천에 선사한 가장 정교한 구조 중 하나입니다. 공개된 채널을 통해 두 사람이 비밀에 합의할 수 있다는 독창적인 아이디어는 1976년 Whitfield Diffie와 Martin Hellman에게서 나왔습니다. 반세기가 지난 지금도 우리는 그 결과 속에 살고 있습니다. 그러나 모든 기술적 약속이 그러하듯, 그 가치는 라벨이 아니라 실제 이행 여부에 달려 있습니다. 정직한 전문가의 질문은

「암호화되어 있는가?」가 아니라 「누가 키를 가지고 있는가?」입니다. 그 답에 따라 결과는 판이하게 달라집니다. 이를 아는 것이 중요합니다.

참고 문헌 및 관련 자료

- Diffie, W.; Hellman, M. — *New Directions in Cryptography*, IEEE Transactions on Information Theory, 1976년 11월. 공개 키 암호화의 기초 논문.
- Perrin, T.; Marlinspike, M. — *The Double Ratchet Algorithm*, Open Whisper Systems의 공개 사양, 2016년 개정. Signal 프로토콜 및 그 산업적 파생물의 기초.
- RFC 7748 — Elliptic Curves for Security (IETF, 2016년 1월). 최신 키 교환에 사용되는 X25519 및 X448 곡선의 규범적 사양.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). 키 교환 및 인증된 암호화 프로토콜에 관한 장.
- 유럽 디지털 신원 프레임워크에 관한 규정 (EU) 2024/1183 (eIDAS 2) — 대화 상대의 독립적인 검증이 제도적 지원을 얻고, 명목상 암호화와 실제 암호화의 구분이 서로 다른 법적 결과를 갖는 프레임워크를 설정합니다.

← 이전킬 스위치와 제도적 포획다음 →신뢰의 신호로서의 비즈니스 모델

최근 읽은 글

- 분석 · 2026년 5월 18일 진정한 개인정보 보호 vs 겉모습뿐인 보호: 스스로 던져야 할 질문
- 분석 · 2026년 5월 18일 전문적 실무로서의 셀프 호스팅
- 개념 · 2026년 5월 18일 24개의 단어: 암호학적 정체성이란 무엇인가

이 기사를 다운로드하여 필요한 곳에서 활용하십시오.

↓ 마크다운 ↓ 텍스트 형식 ↓ PDF

파일이 기기에 다운로드됩니다. 해당 위치에서 저장하거나 Solo2로 가져오거나 원하는 곳에 공유할 수 있습니다. Cuadernos는 전송 대상을 결정하지 않습니다.

봉인 · SHA-256 b5f0933a09ff0ba0d5ecf82cb00fb543ea9ef8bd9092a238fdd2da5b1e2ff506

Cuadernos Lacre · Menzuri Gestión S.L.의 간행물 ·
저자: R.Eugenio · Solo2 팀 편집