概念 · 2026年5月18日

← 上一页·下一页 →

端到端加密，真正的原理解析

供应商在提到 E2EE 时说了什么，又隐瞒了什么。抛开广告包装，对该机制及其局限性的启发式讲解。

直截了当地说： WhatsApp 说您的消息是端到端加密的。这是真的——但这还不够。如果备份在没有额外加密的情况下上传到 iCloud 或 Google Drive，加密在您自己的手机上就被打破了。运营上的问题不在于是否加密，而在于密钥存放在哪里。

加密的真正含义

加密一条消息，意味着将其转换成一种对于任何不持有被称为“密钥”的特定信息的人来说，看起来像噪声的东西。加密操作在发送者的设备上完成，而使用正确的密钥，则在接收者的设备上撤销该操作，使消息恢复原状。在这之间，消息作为一串没有明显含义的字节进行传输。这就是简单的原理。本文的其余部分将探讨那些根据情况将其转变为真实保证或仅仅是市场标签的细微差别。

“端到端”这个形容词——英文为 *end-to-end*，缩写为 E2EE——增加了一个精确的定义。加密并不是为了让中间服务器能够读取并投递消息。加密是为了让只有两端——即发送者的设备和接收者的设备——拥有密钥。消息经过的任何服务器看到的都是噪声，而不是消息内容。这就是它与“传输中加密”（*in transit*）的技术区别，在传输中加密的情况下，内容在服务器之间加密传输，但经过的每个服务器都会对其进行解密以便转发，从而暂时恢复明文。

共享密钥的悖论

这里有一个显而易见的问题。为了让两个人能相互加解密消息，双方都需要同一个密钥。但是，如果他们发送给对方的所有内容，根据定义，都要经过一个可能有人监听的渠道，那他们如何商定这个密钥呢？在稍后将要使用的同一个渠道商定密钥似乎是不可能的：如果攻击者在他们商定时听到了密钥，就能解密随后的一切。几十年来，古典密码学以一种笨拙的方式解决了这个问题：密钥在开始使用前，通过物理会面亲自交付。外交官们随身携带缝在衣里衬中的密钥包。

在当代的电子邮件中，这种解决方案无法扩展。如果我们必须亲自去每个打算进行加密通信的人家里，那我们就没法和任何人说话了。五十年前密码学界提出的问题是：是否可能让两个互不认识且只共享一个公开渠道的人，在同一个公开渠道商定一个该渠道的任何监听者都无法获知的秘密？

Diffie-Hellman 的优雅之处

1976 年，两位名叫 Whitfield Diffie 和 Martin Hellman 的数学家证明了一件看似不可能的事：两个只通过公开渠道——任何人都可听到他们所说一切的渠道——交谈的人，可以在任何听众都无法发现的情况下，商定一个秘密密码。这听起来像魔术，但它不是：它是数学。自那时起被称为 Diffie-Hellman 密钥交换的技术，是几乎所有互联网加密通信的基础。半个世纪的高强度使用和全球学术界的严密审视证实了其稳固性。想了解直观感受或数学原理的人可以继续阅读。宁愿相信它有效的人也可以继续阅读，而不会丢失文章的思路。

对于想通过图像直观感受的人来说，有一个著名的颜色类比。想象 Alicia 和 Bruno 在监听他们的 Eva 面前，公开商定一种基础颜色——假设是黄色。每个人私下选择第二种秘密颜色，并将自己的秘密颜色与黄色混合。Alicia 得到一种特定的橙色；Bruno 得到一种特定的绿色。他们在 Eva 面前交换混合结果。现在，每个人将收到的混合色与自己的秘密颜色再次混合，两人都会得到相同的最终颜色，因为混合的顺序并不重要。Eva 看到了黄色和两种中间混合色，但没看到秘密颜色；没有秘密颜色，她无法得到最终颜色。真实的数学用模群或椭圆曲线中的幂运算代替了颜色，但原理是一样的：共享秘密是在公开场合构建的，而渠道中的任何人都无法重建它。

在算术中，对于更喜欢看机制的人来说： Alicia 选择一个秘密数字 a，Bruno 选择 b。他们在渠道上公开交换 g^a 和 g^b。Alicia 计算 $(g^b)^a$，Bruno 计算 $(g^a)^b$；两人都得到相同的 g^{ab}。Eva 看到 g、g^a 和 g^b 在渠道中经过，但在 g 是在合适的数学群中选择的情况下，从 g^a

中恢复 a——即所谓的离散对数问题——所需的计算时间天文数字般地超过了宇宙的年龄。

写给想用小数字验证一下的人。 Diffie-Hellman 交换可以用小到能够手工计算的数字完整地演示一遍。不想涉及算术的读者可以跳过这一块，这不会影响文章的连贯性；想一步步看清机制运行的人，可以在这里找到答案。**公开的规则**，任何人都能看到：一个素数 $p = 11$（在真实的 Diffie-Hellman 中它大约有三百位；我们用十一是为了让计算过程能写在一页纸上），一个底数 $g = 2$，以及所有算术都在 *模 p (modulo p)* 下进行的约定——你进行计算，除以 p，然后保留余数，就像一个十一刻度的时钟，过了十就会回到零。**私人的选择**，每人一个且绝不共享：Alicia 选择 $a = 4$。Bruno 选择 $b = 7$。

步骤 1。 Alicia 计算 $2^4 = 16$，然后 16 mod 11 = **5**。她把 5 发送出去。Eva 把它记了下来。

步骤 2。 Bruno 计算 $2^7 = 128$，然后 128 mod 11 = **7**。他把 7 发送出去。Eva 也把它记了下来。在两次发送之后，Eva 的笔记本上有了四个数据：$p = 11$, $g = 2$, A = 5, B = 7。她缺少的是 Alicia 和 Bruno 即将推导出的共享数字——而 Eva 是无法重构这个数字的。

步骤 3。 Alicia 拿着 Bruno 发给她的 7，并用她的私有指数 $a = 4$ 对其进行乘方。为了避免处理 $7^4 = 2401$ 这样的大数，计算是分步进行的，每一步都取模：

$7^2 = 49$

49 mod 11 = 5

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

25 mod 11 = 3

Alicia 得到了数字 **3**。

步骤 4。 Bruno 拿着 Alicia 发给他的 5，并用他的私有指数 $b = 7$ 对其进行乘方。同样分步进行：

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

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

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

最后 $5^7 = 5^6 \times 5 = 5 \times 5 = 25 \bmod 11 = \mathbf{3}$。

Bruno 也得到了 **3**。

两人并行计算，都得到了同一个数字，3。 两人在任何时候都没有发送过自己的私有指数。Alicia 不知道 $b = 7$；Bruno 也不知 $a = 4$。每个人都把对方发送的公开值与自己的私有指数结合使用，然后他们在同一个目的地相遇了。**为什么他们能得出相同的数字？** 看看每个人计算了什么：Alicia 计算的是 $(g^b)^a = 2^{7\times4} = 2^{28} \bmod 11$。Bruno 计算的是 $(g^a)^b = 2^{4\times7} = 2^{28} \bmod 11$。这两个量是相等的，因为指数乘法的顺序无关紧要 $(7 \times 4 = 4 \times 7)$。每个人通过不同的路径到达了同一个终点。

那么 Eva 呢？ 她笔记本上有 $p = 11$, $g = 2$, A = 5, B = 7，而她想要得到 3。为了算出 3，她需要知道 a 或 b——但这俩数字都没在渠道中传输过。她唯一的办法就是问自己：“对于哪个指数 a，能使 $2^a \bmod 11 = 5$ 成立？” 因为 p 这么小，她可以尝试 0, 1, 2, 3, 4... 然后在不到一分钟内找到答案。但当把 11 换成一个三百位的素数时，可能指数的空间元素数量比可观测宇宙中的原子数量还要多。**目前，人类还没有发现任何算法能在几十亿年内遍历这个空间。** 这就是所谓的*离散对数问题*：正向计算很容易，逆向计算在计算能力上是不可能的。这就是为什么即使 Eva 逐字逐句地跟踪了整个对话，加密依然坚不可摧的原因。

三个简单的成分——时钟算术、乘方运算以及乘法的交换律 $(a \cdot b = b \cdot a)$——结合在一起，就产生了一个半数人类每天私人通信都依赖的协议。这三个部分中的任何一个，单独拿出来看似乎都并不特别。决定性的是它们的组合方式。

从 Diffie-Hellman 到 Signal 协议

专业即时通讯应用如今使用的端到端加密，几乎无一例外地基于 Diffie-Hellman 交换的一个优雅且经过强化的版本。Trevor Perrin 和 Moxie Marlinspike 在 2013 年至 2016 年间设计的 Signal 协议是这一领域的基准。它结合了两个核心想法：第一，椭圆曲线密钥交换（X25519），用于在两台设备之间生成初始共享秘密；第二，所谓的 Double Ratchet（双棘轮算法），它随每条消息自动更新密钥，因此即便今天设备被入侵，也无法解密过去的消息，而在棘轮转动后也无法解密未来的消息。

在 Zig 中，利用标准库，在两台设备间生成共享秘密的 X25519 交换仅需六行代码即可实现：

```
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)
```

这六行代码中发生了什么： 公钥公开传输。私钥永远不离开各自的设备。每一方根据自己的私钥和对方的公钥，派生出相同的 32 字节秘密，渠道中没有任何人能恢复该秘密。该秘密随后作为加密交换消息的种子。Signal 协议的 Double Ratchet 为该材料增加了持续的轮换，因此一瞬间的泄露不会危及后续的对话。

那么 `std.crypto.dh.X25519` 里面到底是什么呢？没有隐藏的魔法。它们是两个简短的函数，在 Zig 自己的标准库中可以完整阅读。第一个函数从私钥派生出公钥——即交换中的“g^a”：

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

用文章的话来说：私钥与 Curve25519 曲线的基点相“乘”——是在椭圆意义上，而不是初等算术意义上——结果被序列化为 32 字节。`clampedMul` 操作是该标量乘法的强化版本：它结合了密码学界多年来为抵抗已知攻击家族而添加的保护措施。只有两行函数体。

第二个函数将你的私钥与对方发送给你的公钥结合起来。它是交换中的“$(g^b)^a$”，它生成了你们双方都未曾传输过的 32 字节共享秘密：

```
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();
}
```

又是两行代码。接收到的公钥被解释为曲线上的一个点，并与自己的私钥相“乘”。通过曲线操作的交换律——类似于我们在数值示例中看到的指数乘法的交换律——双方最终得到相同的序列化点：正是文章所说的共享秘密。

仅此而已。 在应用程序中看起来像魔法的东西，实际上只是两个各占三行的函数。技术上的复杂性集中在单一的操作 `clampedMul` 中，该操作编写在同一标准库的靠后位置，数十年来经过国际密码学界的审查，并且可供任何想逐字阅读的人使用。无论是我们的应用程序还是 Zig 的标准库中，都没有黑匣子。这些都是人类能够理解的开源代码，你可以自行选择深入研究它的节奏。

端到端加密保护什么

在假设实现正确的情况下，E2EE 能很好保护的是传输中的消息内容。接收并转发加密数据的中间服务器看到的将是一串无法理解的字节。拥有光缆、路由器或 wifi 访问点权限的攻击者看到的也是同样的内容。保留流量副本的服务供应商事后也无法读取。命令服务运营商交付内容的政府收到的也将是服务器最初拥有的那些无法理解的字节。

从实用角度来看，这已经很多了。这就是在不透明信封里写信和在明信片上写信的区别。两者都能送达，但只有一种能在邮递员面前保护内容。

端到端加密不保护什么

同样值得清楚了解的是：E2EE 并不保护元数据（Metadata）。服务器依然知道用户 A 在什么时间、以什么频率、从哪里给用户 B 发送了数据，即便它不知道用户说了什么。正如我们已在《*加密不等于私密*》中论证过的，这些元数据往往比内容本身更具揭示性。知道某人在周五 22:00 给一家专门办理离婚业务的律师事务所打了 30 分钟电话，这所讲述的故事是通话内容从未讲述过的。这就像看到一个人多次进出肿瘤诊所：无需听到里面的谈话，就能想象发生了什么。单个孤立的元数据可能毫无意义，但多个交叉比对的元数据所描绘出的东西与事实太像了。此外，E2EE 并不保护端点：如果接收者的设备被恶意程序入侵，消息会照常为该接收者解密，而恶意程序就能读取它。E2EE 本身也并不防止通话对象身份的造假：如果 Alicia 相信自己正在与 Bruno 交谈，但攻击者在开始时就已介入（即“中间人”攻击 *man in the middle*），且协议不包含独立验证，那么双方最终都会在认为是在与对方交谈的情况下，实际上在与入侵者交谈。

还有第四点值得明确表述。E2EE 并不阻止自称提供该功能的供应商在其自身系统中额外保存一份未经加密的消息副本。“我的消息是端到端加密的”这一声明与“供应商不保存我的内容”这一声明并不是一回事。一个应用可以在违反后者的同时满足前者；自 2018 年以来，我们已在新闻头条中反复看到这种情况。除非客户端代码是可验证的，否则用户在没有专家调查的情况下，在技术上无法区分这两种情况。在普通大众中最著名的案例是：WhatsApp 在传输过程中对消息进行端到端加密，但如果用户在没有额外加密的情况下激活了 iCloud 或 Google Drive 的备份，该副本就会以可读形式存储在第三方基础设施中，加密在用户自己这一端就被打破了。

运营商最不想听到的问题

在技术上，一个声称进行端到端加密的应用在密钥方面可以做以下三件事之一：

1. **密钥仅保存在设备上。** 密钥仅在用户设备上生成并保存；运营商既不知道也不存储它们。这是最理想的情况。
2. **运营商如果想访问，就可以访问。** 运营商拥有用户的密钥（或可以随意生成密钥）并将其保存在其数据库中。如果他们想访问或被迫访问，就可以读取内容。大多数“云”服务都是这种情况。
3. **运营商在设计上无法访问，但控制着访问权限。** 运营商不持有密钥，但控制着生成密钥的应用。如果被迫，他们可以发送一个恶意更新，在加密前截获密钥或内容。许多商业 E2EE 服务都是这种情况。

因此，运营上的问题不是某样东西是否加密了，而是谁控制着设备以及管理密钥的软件。在 Solo2 中，密钥仅保存在您的“保险库”（用您的密码加密的 IndexedDB）中，且软件是可验证的开源代码。

致专业读者

端到端加密是实现数字主权的一种工具。但像所有工具一样，其效力取决于握持它的手和它立足的土地。

1. 加密密钥是在哪里生成的，物理上驻留在哪里？如果运营商能够访问它们（即便是暂时的，甚至是打着恢复的幌子），那么 E2EE 就只是名义上的。
2. 是否存在对通话对象的独立验证（安全码、二维码、带外比较），以防止在建立对话期间发生中间人攻击？
3. 客户端代码是否可审计——开源、发布、可重现——还是要求必须相信供应商关于客户端实际运作的说法？
4. 服务生成并保留了哪些元数据，保存多久？即使内容是不透明的，元数据也可以重构出大部分敏感信息。

这四个问题要求的并不是高级的技术信息；它们要求的是任何诚实的运营商都能在其公开文档中回答的信息。回答的质量和准确度与回答本身一样，都能说明该产品的很多情况。

端到端加密，如果实现得当，是当代密码学奉献给日常实践的最精妙构筑之一。最初的创意——两人能在公开渠道商定一个秘密——归功于 1976 年的 Whitfield Diffie 和 Martin Hellman；半个世纪后的今天，我们仍生活在这一创意的深远影响中。然而，正如任何技术承诺一样，其价值取决于真实的履行，而非标签。正直的专业人士会问的不是“加密了吗？”，而是“谁持有密钥？”。答案的不同将导致截然不同的后果。了解这些至关重要。

来源及延伸阅读

- Diffie, W.; Hellman, M. — *New Directions in Cryptography*，IEEE Transactions on Information Theory，1976 年 11 月。公钥密码学的奠基性文章。
- Perrin, T.; Marlinspike, M. — *The Double Ratchet Algorithm*，Open Whisper Systems 的公开规范，2016 年修订版。Signal 协议及其工业衍生品的基础。
- RFC 7748 — Elliptic Curves for Security（IETF，2016 年 1 月）。现代密钥交换中使用的 X25519 和 X448 曲线的规范说明。
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications*（Wiley，2010 年）。有关密钥交换和认证加密协议的章节。
- 关于欧洲数字身份框架的第 2024/1183 号法规（欧盟）（eIDAS 2）——建立了一些框架，在这些框架中，对通话对象的独立验证获得了制度上的支持，并且名义加密与真实加密之间的区别会产生不同的法律后果。

← 上一页自毁开关 (Kill switch) 与机构俘获下一页 →商业模式作为信任的信号

最近阅读

- 分析 · 2026年5月18日 真实隐私 vs 表象隐私：你应该问自己的问题
- 分析 · 2026年5月18日 作为专业实践的自托管 (Self-hosting)
- 概念 · 2026年5月18日 24 个单词：什么是加密身份

随身携带本文，以备不时之需。

↓ Markdown ↓ 纯文本 ↓ PDF

文件将下载到您的设备中。您可以从那里将其保存、导入 Solo2 或在任何地方共享。Cuadernos 不会为您决定文件的去向。

火漆印章 · SHA-256 31e1d413ef59cfe4242b67cf4b4165d9fdbbbe7bcda096b6443cf480cb64085d

Cuadernos Lacre · Menzuri Gestión S.L. 的出版物 ·
由 R.Eugenio 撰写 · 由 Solo2 团队编辑。