爱丽丝与鲍勃梦游仙境：在没有服务器的世界里如何从零建立身份与权限

Herb Caudill · 2026-10-05 · 中文译文

中文译文。根据 2026 年 10 月 5 日的演讲与问答整理，保留发言顺序、例子、限定条件和交流承接，校订口头重复、断句与明显识别错误。产品状态和后续计划以分享时点为准。时间对应交付视频。

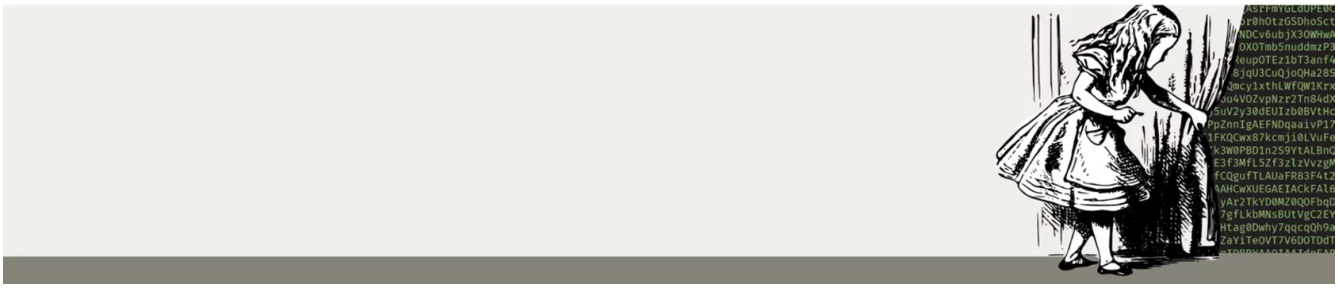
没有服务器，身份与权限靠什么



Alice和Bob梦游仙境 · 画面 00:00:01

“要是有什么事情
能难得讲通一回，
那该多好啊。”

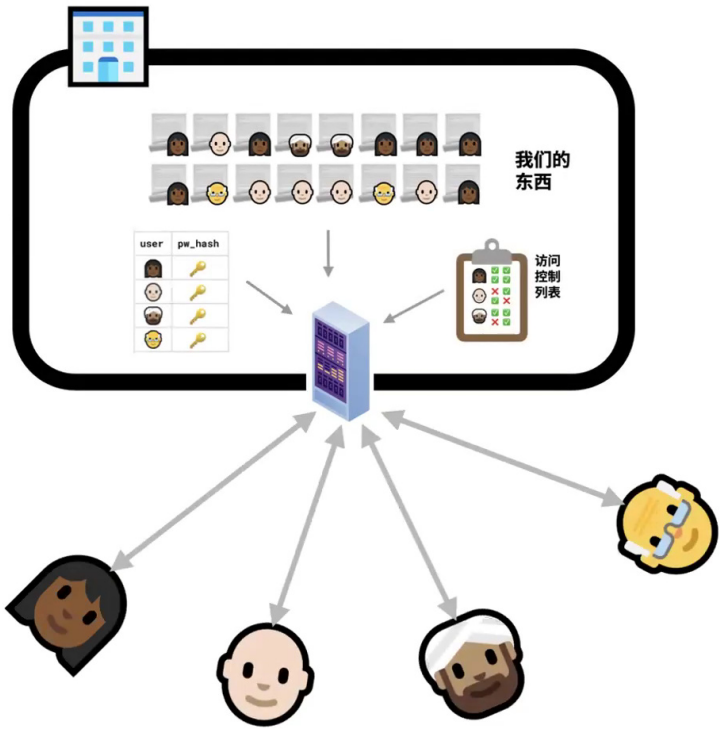
《爱丽丝梦游仙境》



重复解释的困境・画面 00:00:12

00:00:00・Herb Caudill

我叫 Herb Caudill，今天要讲的是本地优先中的身份认证与授权。几年前，我刚开始思考这个问题时，真的觉得很迷茫，也很奇怪。你可能不太明白，这有什么好迷茫的，所以我先解释一下。



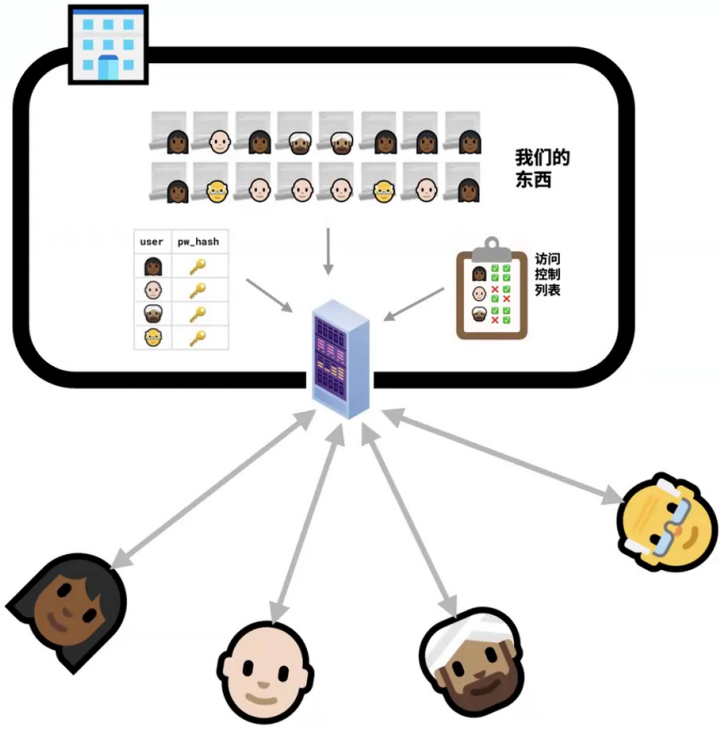
中央服务器承担身份与权限 • 画面 00:00:59

00:00:24 • Herb Caudill

在我们大多数人熟悉的世界里，身份认证与授权最终都依托于物理硬件。这套硬件由一家公司拥有并控制，而你信任那家公司。它可能是你的雇主、软件服务提供商，或者一家大型科技公司。那台服务器上存着你所有的东西，也有一套规则，规定你可以做什么。它保存着你的用户名和密码。任何人想访问自己的东西，都得经过那台服务器，对吧？



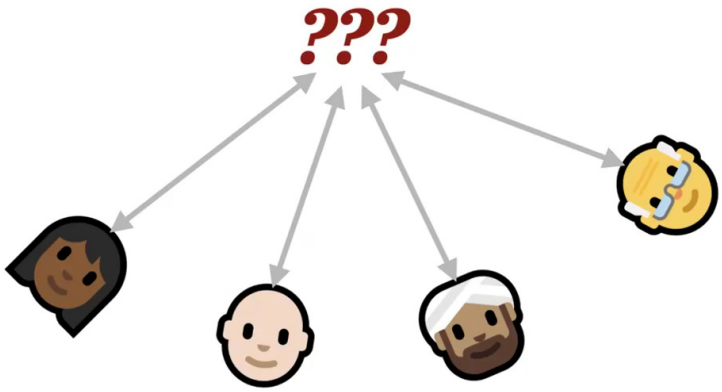
城堡守卫比喻・画面 00:01:09



中心化提供单一事实来源・画面 00:01:36

00:01:03 • Herb Caudill

服务器就像守在城堡门口的卫兵。如果它不认识你，就不让你进去；如果它认为你不该拿到某样东西，就不会把东西给你。当然，我们讨论的是本地优先，我们更希望不依赖中心服务器。不过，服务器有一个好处：它确实提供了一个唯一可信的数据来源，而且在物理上保管着你的数据。这就给了你一个坚实的依托，对吧？



没有服务器时的身份问题 • 画面 00:02:15

00:01:43 • Herb Caudill

当服务器不在了，你就失去了这个坚实的依托。剩下的只有客户端，而且它们彼此平等。那你怎么知道谁是谁？怎么知道该信任谁？怎么知道谁拥有哪些权限？在每个人都有一份完整数据副本的情况下，“权限”到底意味着什么？又是谁在授予权限？没有了服务器，我们就得另找办法，解决那些原本在中心化架构里已经解决的问题。这就是让人迷茫的地方，就像掉进了兔子洞。



掉进兔子洞・画面 00:02:23



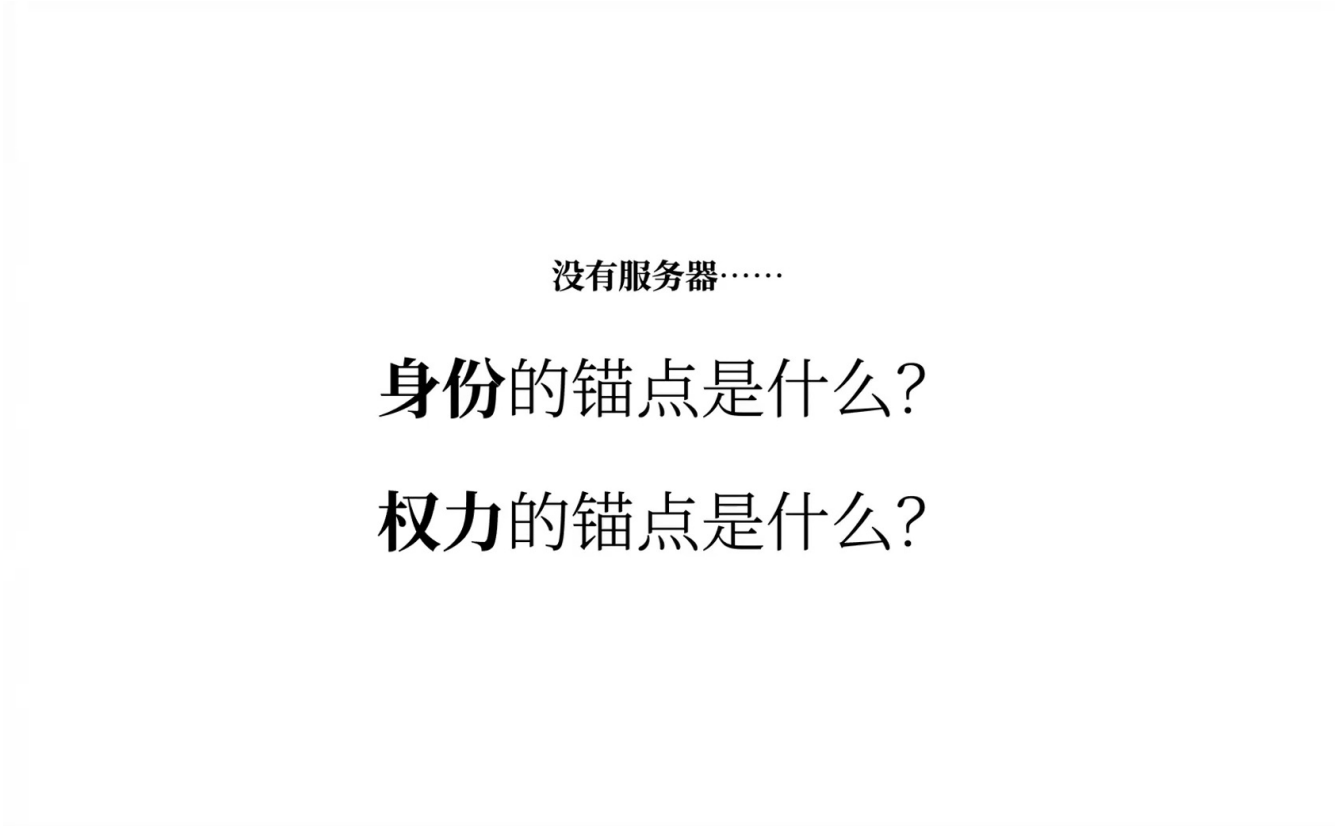
Bootstrapping・画面 00:02:39



拽着鞋带把自己提起来 · 画面 00:03:06

00:02:30 · *Herb Caudill*

英语里的“bootstrapping”，现在会让我们想到开机这样简单、日常的事情，也就是启动电脑。但这个比喻原本要表达的，恰恰是一件不可能的事：你不可能拽着自己靴子上的提带，把自己提起来。这是不可能的，做不到。可是在这里，我们偏偏得想办法，从零建立身份与权限。

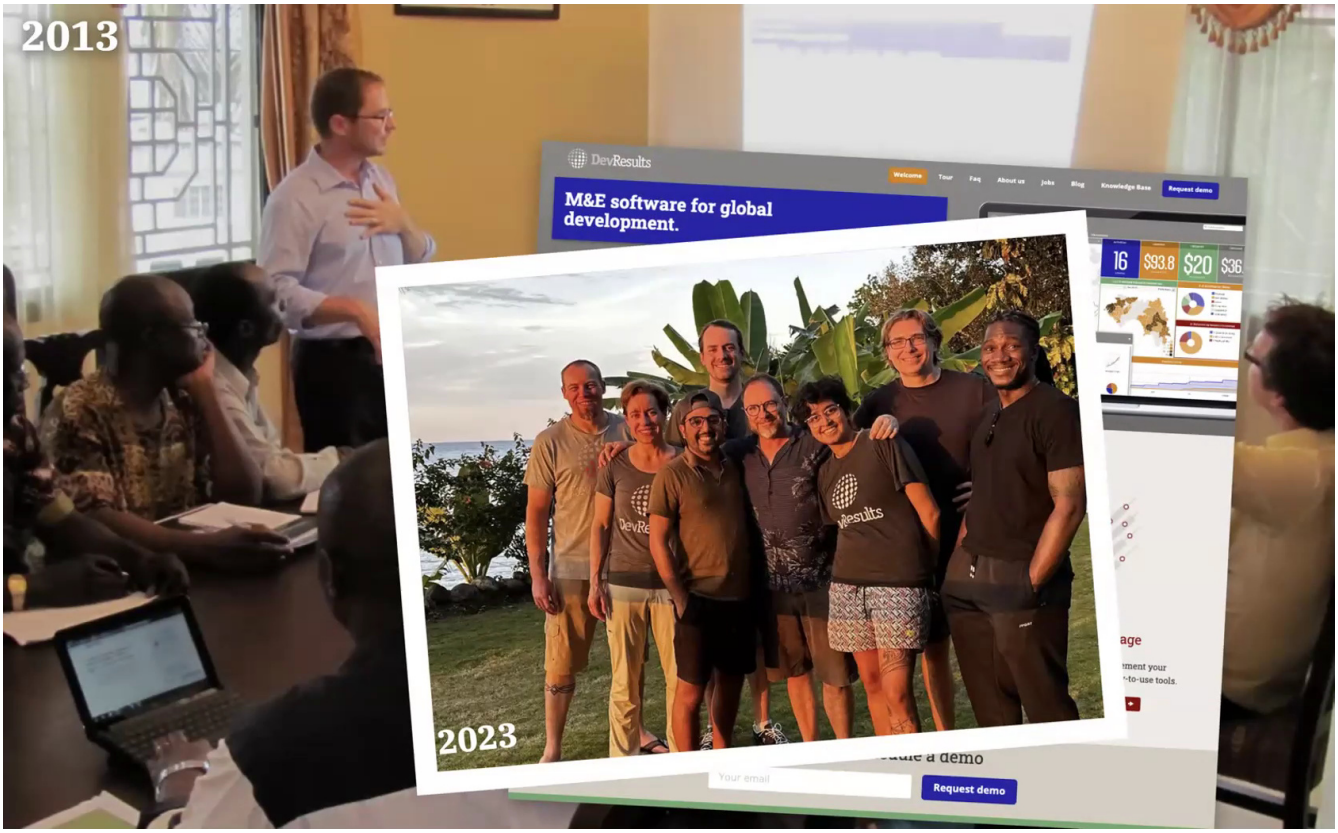


身份与权力的锚点 · 画面 00:03:31

00:03:14 · Herb Caudill

所以，没有服务器时，第一个问题是：身份以什么为依据？我们怎么知道自己在和谁协作？另一个问题是：授权的权威从哪里来？权限是如何运作的？最初又是谁有权给别人授予权限？

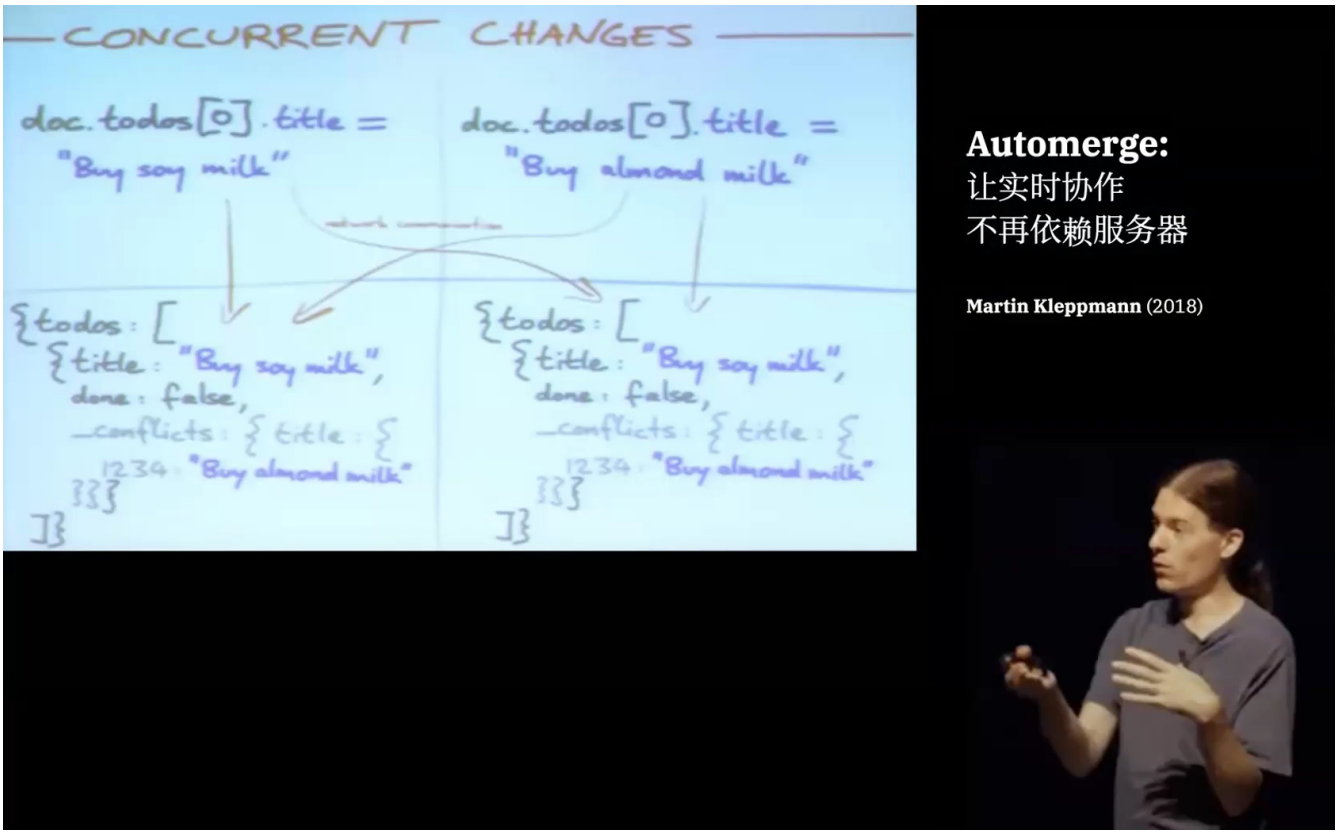
从 DevResults 到本地优先



DevResults与开发背景・画面 00:04:11

00:03:35・Herb Caudill

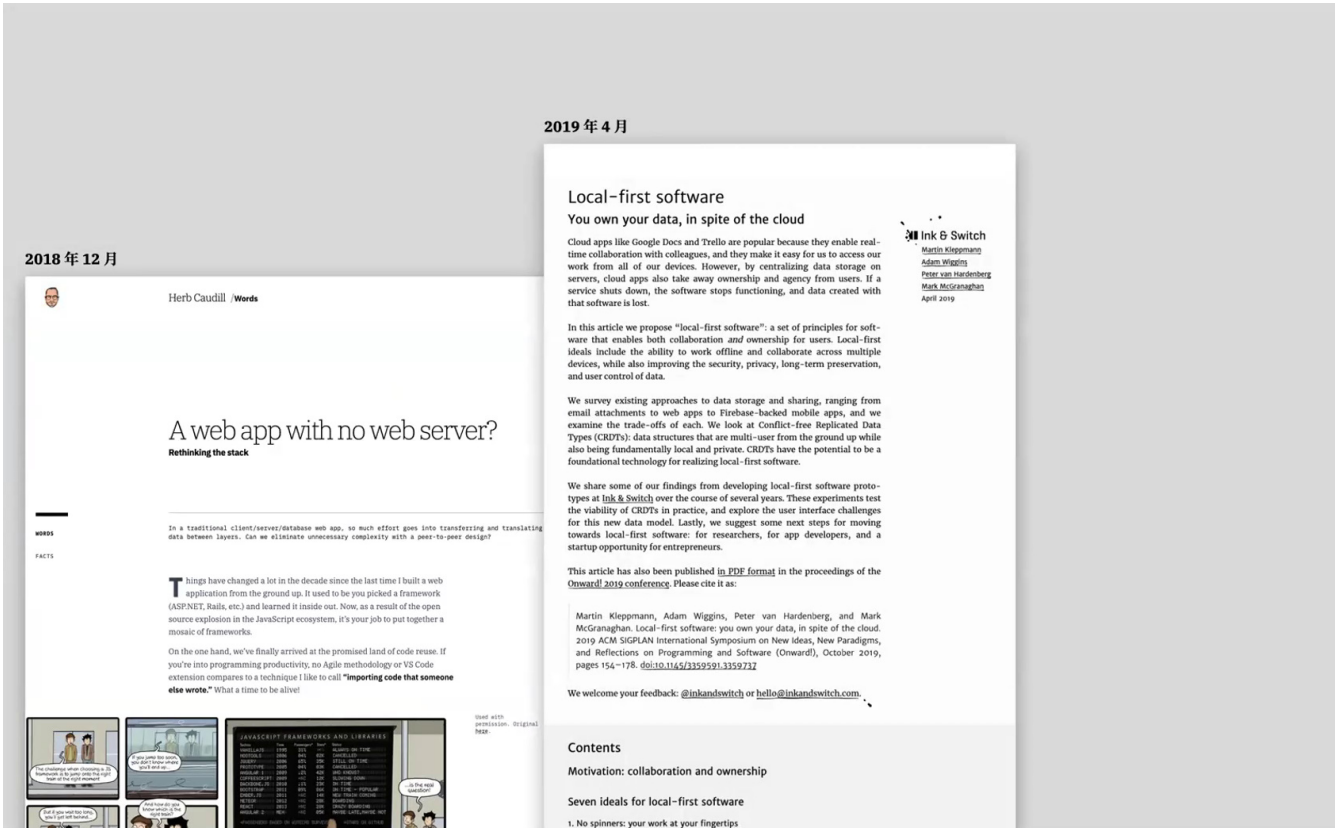
先说一点背景，讲讲我自己的动机。大约 15 年前，我开始开发一款叫 DevResults 的软件，使用者主要是政府援助机构和国际非政府组织。比如这张照片，就是很久以前，我在西非的塞拉利昂给一个团队做培训。这家公司取得了一些小小的成功，我们有一个规模不大、但成员都很能干的团队。我也开始思考，接下来要做什么。



Automerge与不再依赖服务器 • 画面 00:04:31

00:04:11 • Herb Caudill

可是，一开始尝试开发新版本，我就对软件开发方式里的一些根本问题感到沮丧。差不多就在那时，也就是 2018 年，我偶然看到了这个视频，由此认识了 Automerge。那就是我踏上这段“仙境之旅”的起点。



本地优先早期文章・画面 00:04:51



作为软件开发者

- 编程模型简单
- 不必托管你的数据



作为软件用户

- 离线可用，随处可用
- 我的东西归我所有

开发者和用户的诉求・画面 00:06:11

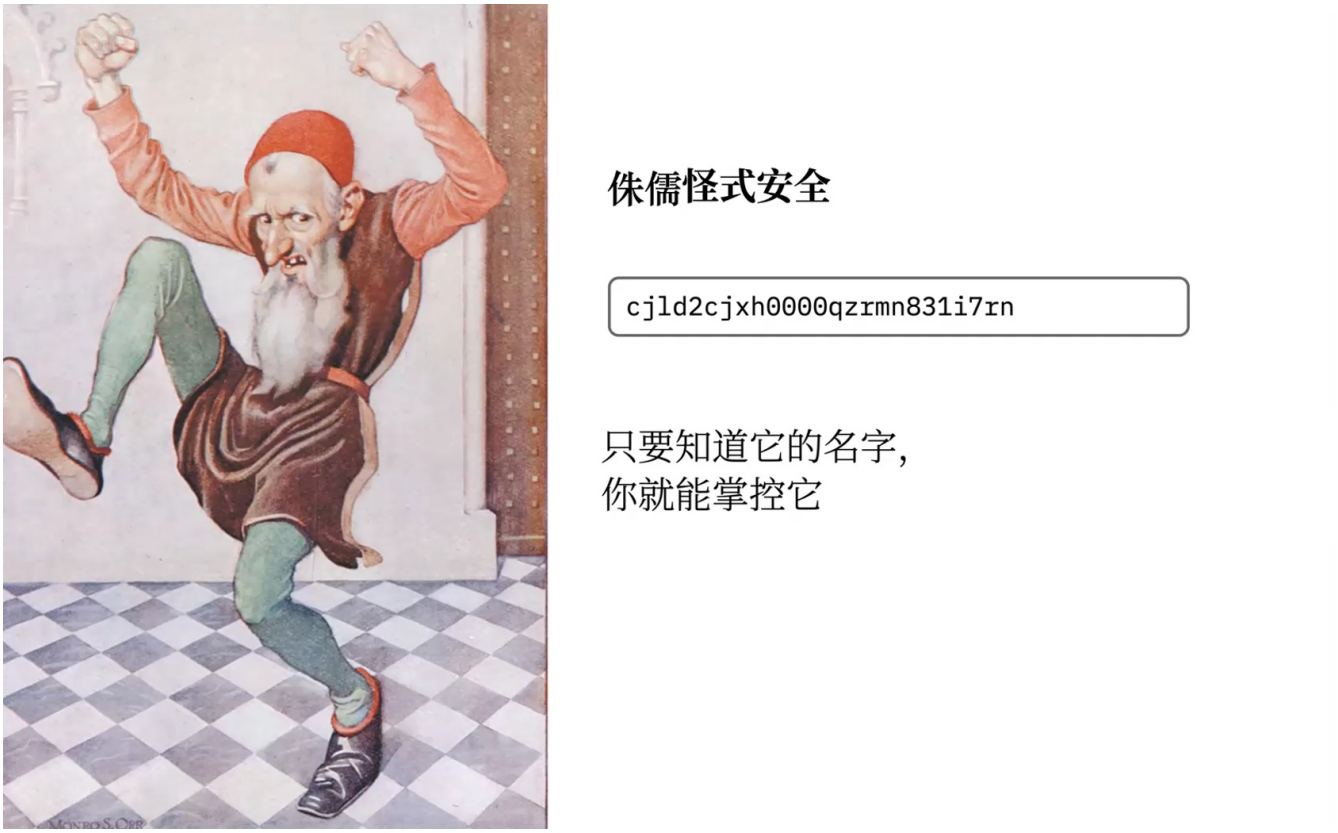
00:04:40 • Herb Caudill

不久之后，我写了这篇文章。顺便说一句，它比 Ink & Switch 的本地优先宣言早发表了五个月。所以，这件事我已经想了很久。本地优先有几个地方很吸引我，无论我是作为软件开发者，还是作为软件使用者，都是如此。作为开发者，我喜欢它的简单：只需要编写客户端，就能更快地迭代。

00:05:21 • Herb Caudill

说到底，我想做的是软件，并不想替任何人保管数据。如果不用托管你的数据，我的工作也会简单一些。作为软件使用者，我希望自己的东西归自己所有。我访问自己所创建文档的权利，根本不该成为问题；我也不该非得联网，才能访问自己的东西。当然，DevResults 的客户分布在非洲、拉丁美洲，以及其他网络时断时续的地方，对他们来说，能够离线访问也很重要。

文档 ID 与撤销权限的问题



依靠咒语实现安全 • 画面 00:06:56

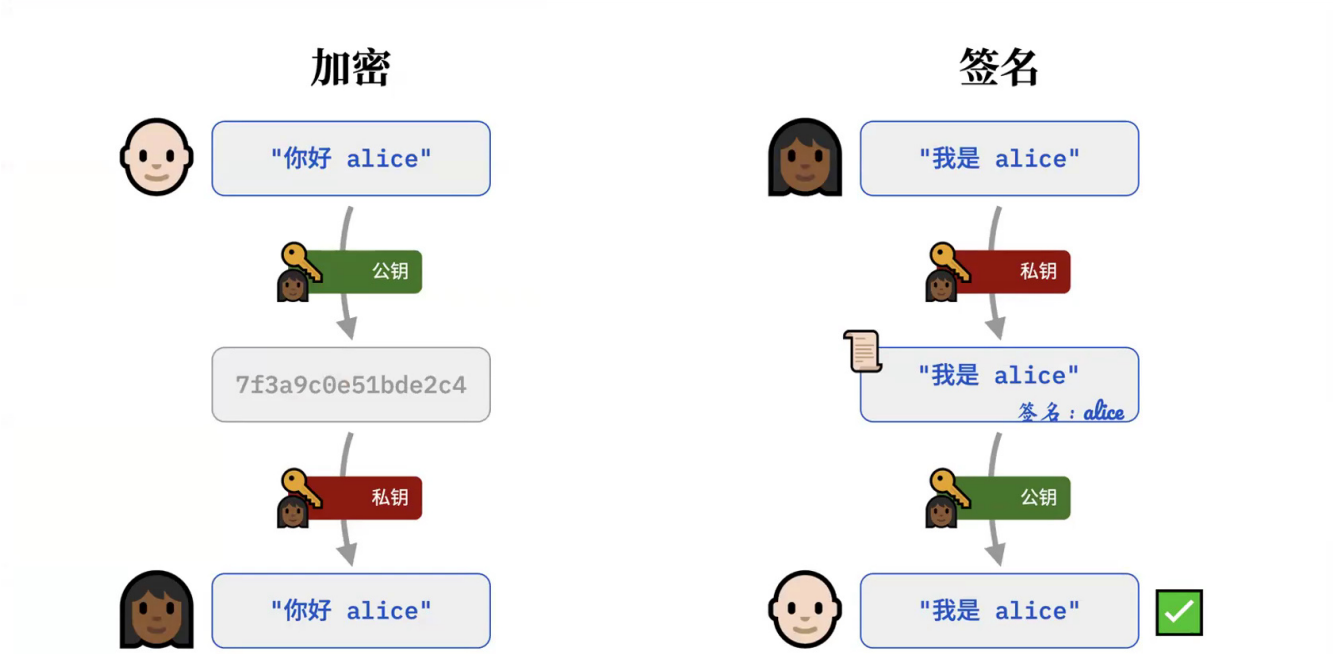
00:06:12 • Herb Caudill

后来，我和团队开始尝试开发一些内部使用的本地优先工具，但马上就遇到了一个问题：怎么保证它们的安全？我们最初采用的办法，有些人可能很熟悉：直接把根文档的文档 ID 当作访问令牌。那是一个很长的随机 ID，很难猜中。这个办法的意思就是，只要知道文档的名字，就能访问它。对某些使用场景来说，这已经够用了。

00:06:49 • Herb Caudill

可对于一个团队来说，如果有人离开团队，或者某人的设备被攻破，又或者设备丢了，该怎么办？你没法简单地把文档 ID 换掉，基本上得重新来过，也没有多少好办法可选。对于商业产品来说，这肯定不够，对吧？

公钥密码学与密钥管理



任何人都能写给 *alice*，只有 *alice* 能读

只有 *alice* 能签名，任何人都能验证

加密与签名 • 画面 00:07:28

00:07:14 • Herb Caudill

2020 年，疫情开始几周后，我开始开发一个叫 @localfirst/auth 的库。先说明一下，我不是密码学家，现代密码学是我从零学起的。我很快就被公钥密码学赋予人的能力吸引住了：它能让我与世界上任何人站在同等的位置上，无论对方有多强大。



影视中的破解密码 · 画面 00:07:51



影视破解与真实破解对照 · 画面 00:08:13

00:07:45 · Herb Caudill

在电影里，好像只要打字够快，就能破解加密，什么都能解密。但现实更像这样：如果加密做对了，就破解不了。数据经过正确加密，就没有办法直接把它解开。

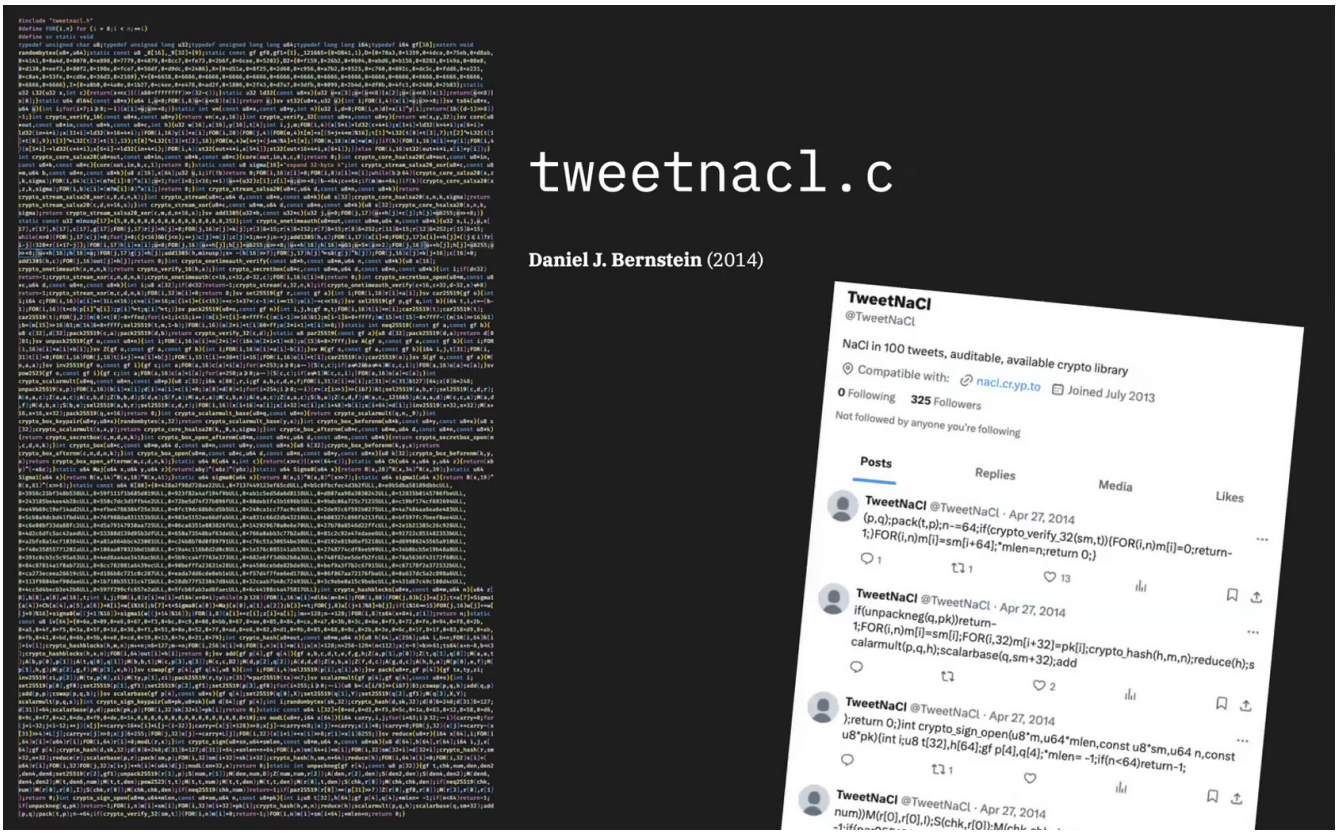
对大多数密码学操作而言，
都存在被广泛接受、
经过深入研究的原语，
在任何现有的计算机集群上，
破解它们都被认为
在计算上不可行。

— **Daniel J. Bernstein** 等（2012）

现代密码学的安全基础 · 画面 00:08:31

00:08:20 · Herb Caudill

无论对手多么强大、资金多么充足，都一样。从数学上说，无论是谁、有多少资源，没有正确的密钥就解密一条消息，或者伪造数字签名，都是不可行的。这真的给了人很大的力量。你不需要什么专用设备，只需要做一些你我在便宜的笔记本电脑上都能完成的数学运算，甚至用不了多少代码。



TweetNaCl • 画面 00:09:22

00:08:55 • Herb Caudill

这里展示的，就是 NaCl 密码学库一个精简 C 语言实现的全部代码。它的设计目标，是把完整实现放进 100 条推文里，也就是不到 25,000 个字符。这很了不起，代码量并不大。



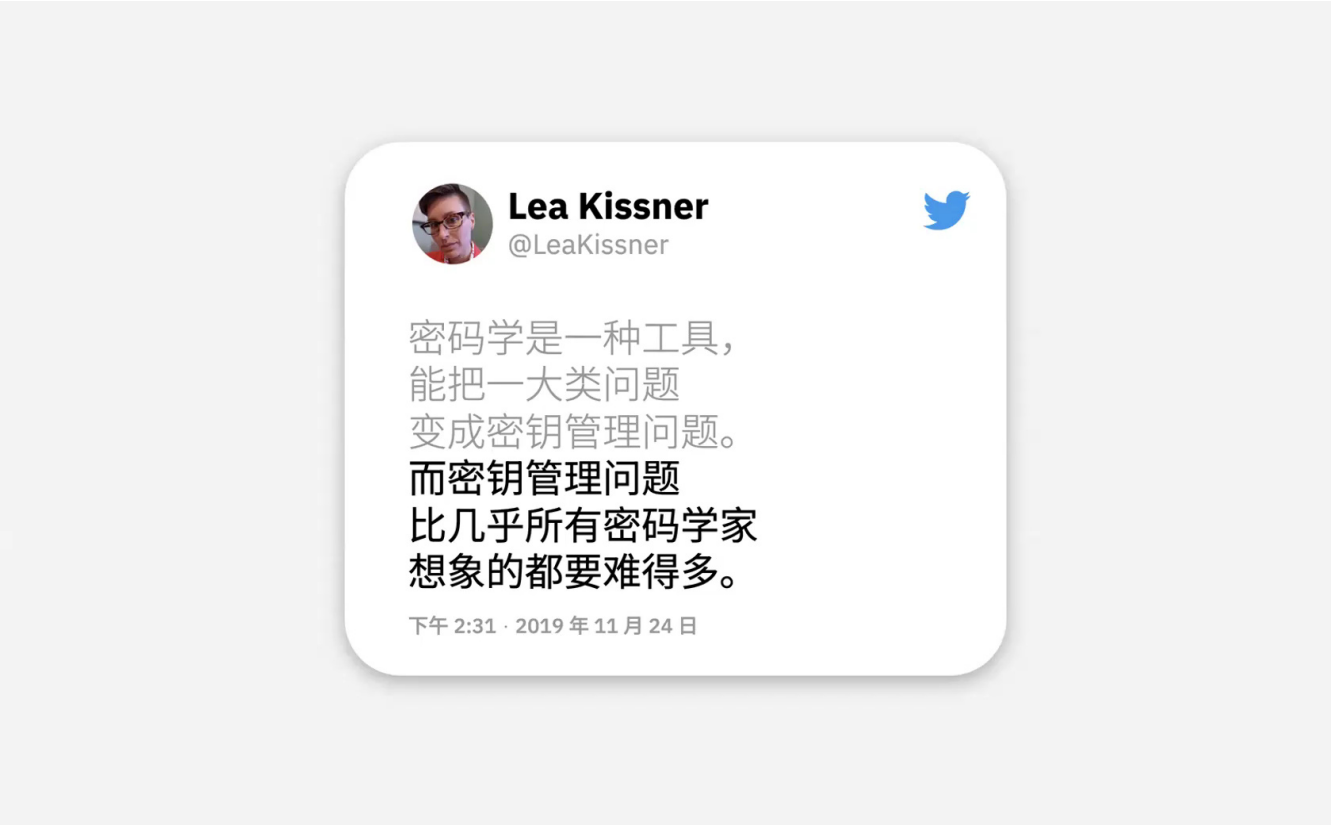
任何人都能写给 *alice*，只有 *alice* 能读

只有 *alice* 能签名，任何人都能验证

再次说明加密与签名 • 画面 00:09:36

00:09:23 • Herb Caudill

对我来说很明显，公钥密码学一定是本地优先身份认证与授权方案的一部分。但首先有一个问题：在这两种情形中，Bob 都必须知道 Alice 的公钥。在去中心化的环境里，我们怎么确保 Bob 知道的就是 Alice 的公钥？这就把我们带回了第一个问题：没有中心服务器，身份究竟以什么为依据？



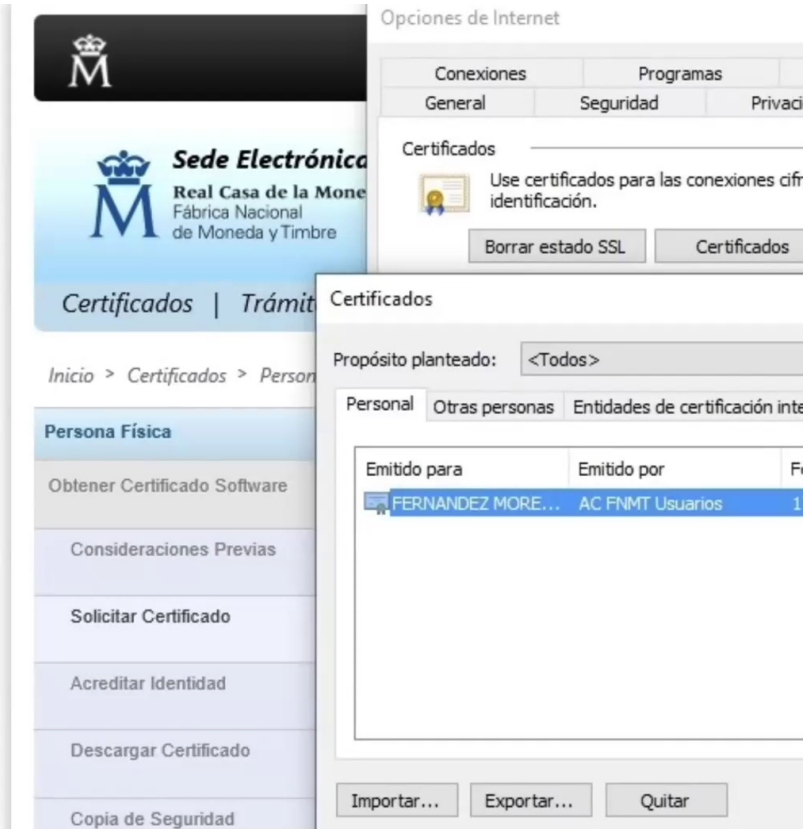
密码学把问题变成密钥管理 • 画面 00:10:31

00:10:03 • Herb Caudill

有一句很有名的话：“密码学是一种工具，能把一大堆问题转化为密钥管理问题。”这句话出自 Lea Kissner 博士，当时任 Twitter 的信息安全负责人。这条推文本来并不是为了鼓舞大家，后面还说：“密钥管理问题比大多数人想象的更难。”但我反而觉得很受鼓舞。是的，这是个难题，可只要能解决它，我宁可面对这一个难题，也不愿面对一大堆问题。

PKI

public
key
infrastructure
公钥基础设施



PKI公钥基础设施 • 画面 00:11:41

00:10:45 • Herb Caudill

那怎么解决密钥管理问题？我们怎么知道别人的公钥是什么？最直观的办法，是让一个可信方维护一份目录，记录每个人及其公钥。这叫 PKI，也就是公钥基础设施。这类方案通常会用到中心化服务器。比如，证书颁发机构让 HTTPS 连接得以实现；一些大公司有自己的证书颁发机构，为员工的数字身份提供依据；国家也可以为居民做类似的事情。

00:11:28 • Herb Caudill

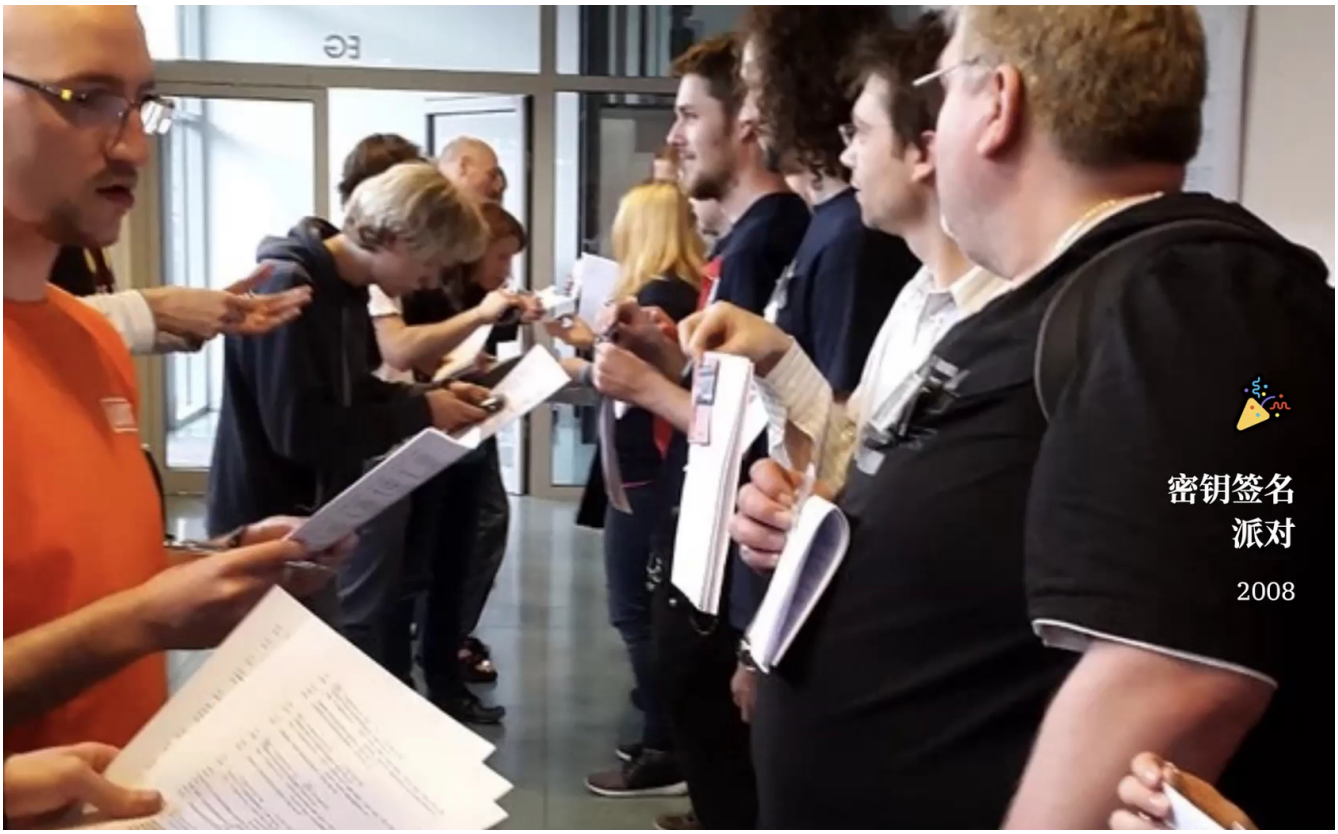
我住在西班牙，政府给我签发了一份证书。我把证书装进浏览器，就可以用它登录政府网站、签署正式文件、缴税，等等。但如果没有中心化服务器，这件事该怎么做？在点对点或去中心化的环境里，又该怎么做？



PGP公钥 · 画面 00:12:23

00:11:58 · Herb Caudill

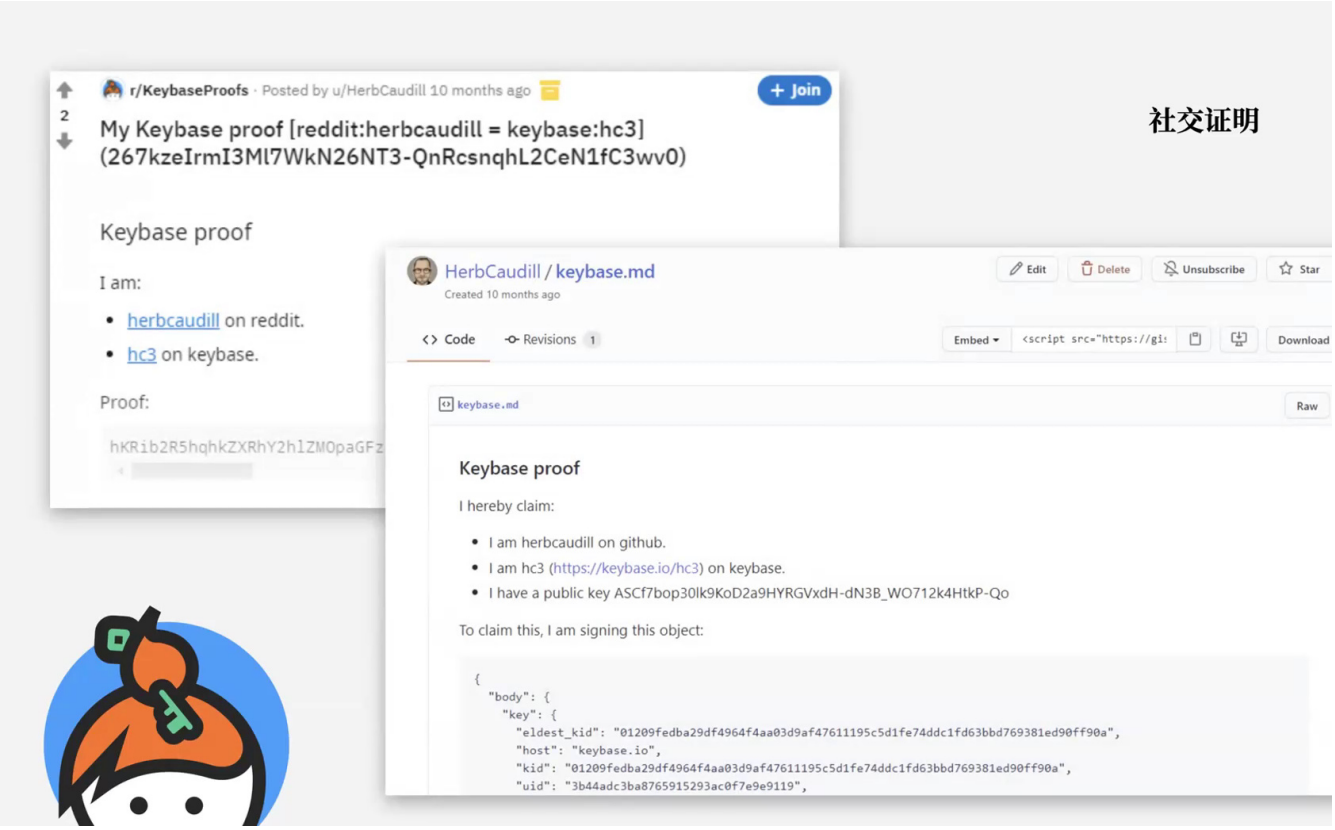
最早的一种解决方案叫 PGP，全称是 Pretty Good Privacy。比如，这就是我的 PGP 公钥。它的思路是，直接把自己的公钥分享给别人。人们会把公钥放在自己的网站上，或者放进邮件签名里。但你看，这么一大段东西，确实挺吓人的。



密钥签名聚会 • 画面 00:12:53

00:12:28 • Herb Caudill

当时还有所谓的“密钥签名聚会”：大家排着队，互相出示身份证件，再通过签名来确认自己已经核验过。这个办法是想逐步建立起一个信任网络。但显然，只有极客才会这么做，它在普通人当中并不流行，所以始终没能真正普及。



Keybase社会证明・画面 00:13:51

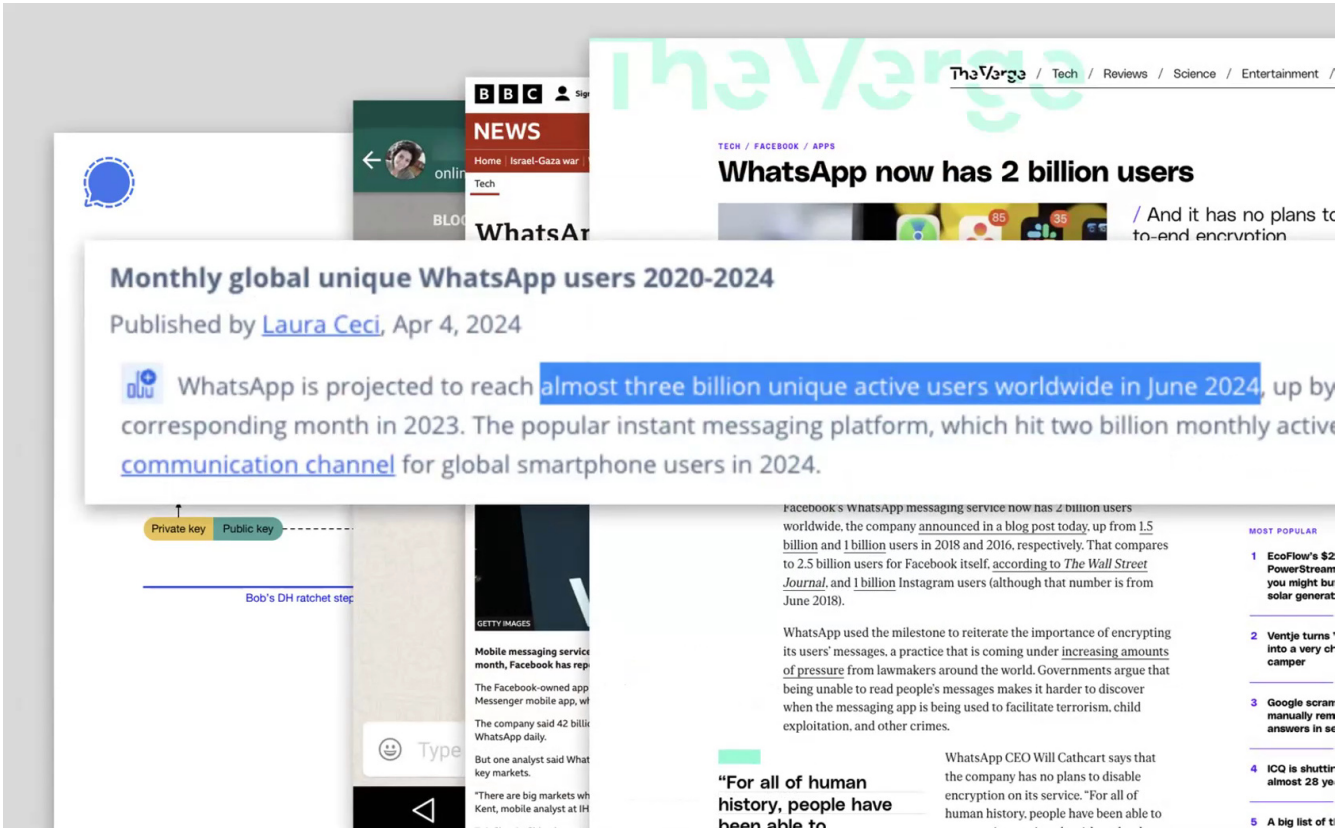
00:13:07・Herb Caudill

几年前，有一家公司叫 Keybase，后来在 2020 年被 Zoom 收购了。Keybase 有个很棒的想法，叫 social proofs，也就是社交身份的证明：你在 Twitter、GitHub 或 Reddit 这样的网站上发布一份密码学证明，把你的公钥和你在那些网站上的身份关联起来。之后，你可以用自己的密钥签名。他们开发的应用会沿着这个关联，追溯到你的 GitHub 等账号，验证这确实是你的密钥。

00:13:55・Herb Caudill

当然，这比参加密钥签名聚会简单，但还是只有极客会这么做。普通大众并没有觉得它有多大用处。不过，稍后我们还会回到 Keybase。

Signal、WhatsApp 与设备密钥

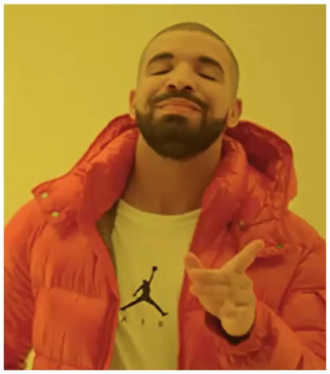


Signal与WhatsApp的普及・画面 00:15:01

00:14:11・Herb Caudill

真正改变了局面、把端到端加密带给大众的，是 Signal 协议。最开始是 Signal 应用，它还是有点极客气质，主要由那些对加密特别感兴趣的人使用。后来，到了 2016 年，WhatsApp 一夜之间就让它的全部用户用上了端到端加密，当时大约有十亿人。如今，经常使用 WhatsApp 的人已经超过三十亿。它是人类历史上使用最广泛的通信工具之一。





密钥的作用范围是**什么**?	一个人的身份	**一台设备上的一个应用**
谁来管理密钥?	人	**计算机**
我们**什么时候**会看到密钥?	有时	**从不**

从个人身份转向应用内身份 • 画面 00:16:16

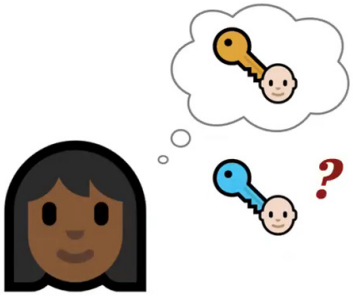
00:15:01 • Herb Caudill

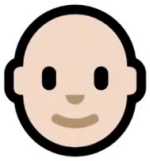
那么，WhatsApp 和 Signal 想明白了什么，而 Keybase 或 PGP 没有做到？它们让加密对用户变得无感。这需要换一个思路：不是让我这个人拥有一把统一的密钥，而是每台设备上的每个应用各有自己的密钥。密钥也不由我亲自管理——人本来就不擅长管理密钥，我们连密码都管不好，更别说 PGP 密钥了。电脑倒是很擅长这些事，那就让电脑来做。

00:15:50 • Herb Caudill

在这个模型里，密钥由设备生成，用户从来不需要看到它们。Ink & Switch 的负责人 Peter van Hardenberg 常说，如果用户看到了加密密钥，你就已经输了。所以，把这些事留在电脑内部就好，它只是用户不需要了解的实现细节。人们也不用亲自互相交换密钥，设备会自动完成。这是一种好得多的密钥管理思路。

TOFU：首次信任的局限





TOFU

trust
on
first
use

首次使用即信任

TOFU与换钥疑问 · 画面 00:18:01

00:16:30 · Herb Caudill

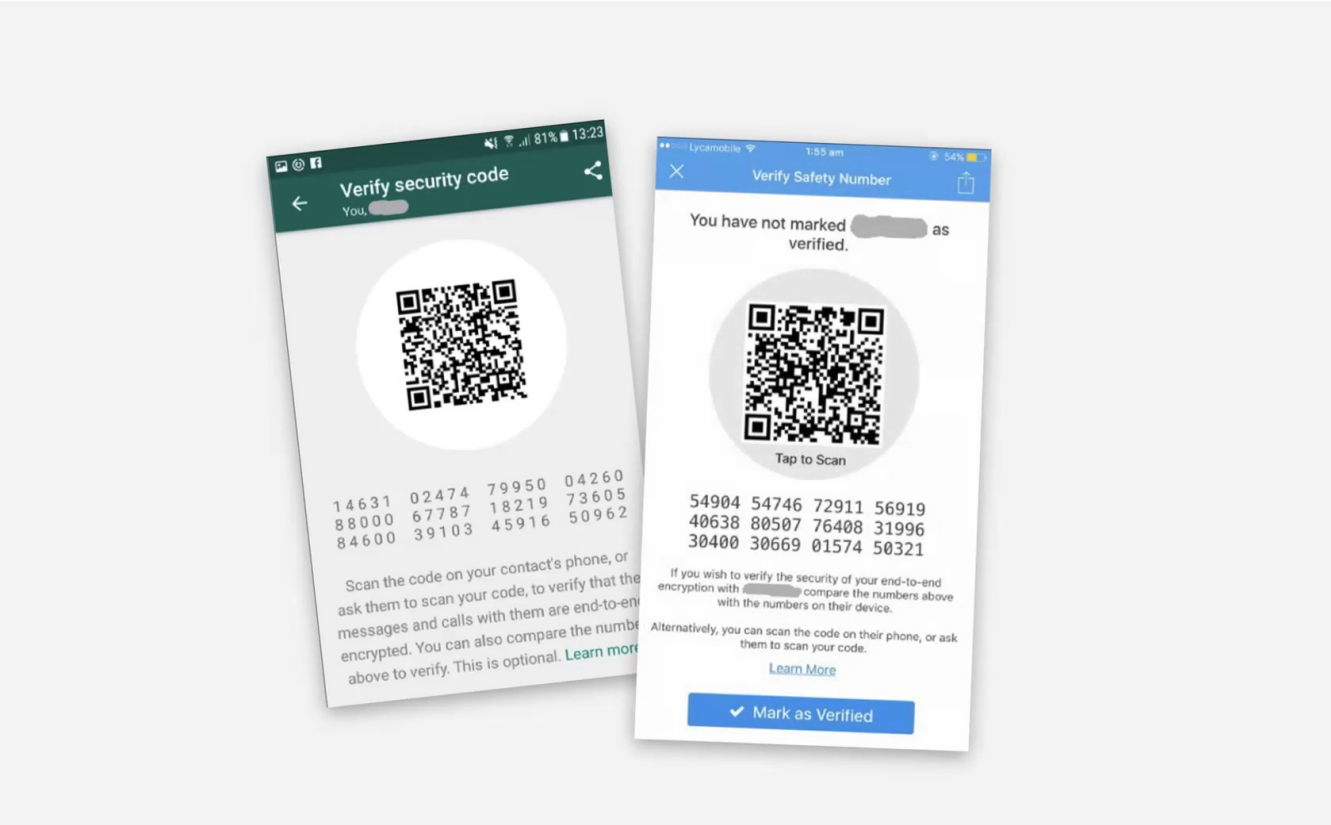
但还有一个问题：Alice 的设备怎么知道 Bob 设备的公钥？WhatsApp、Signal 等应用采用了一种叫 TOFU 的办法，也就是 Trust On First Use，首次使用时信任。它的思路是：Alice 的设备相信，自己第一次连接到的就是 Bob 的设备。Alice 的设备把公钥发给 Bob 的设备，Bob 的设备再把自己的公钥发回来，双方记住对方的公钥。从此以后，就可以用这些密钥进行身份认证，并加密彼此发送的消息。

00:17:21 · Herb Caudill

听起来很合理，但这里说的“设备”，并不能准确反映实际情况。比如，你换一个浏览器打开应用，用 Firefox 而不是 Chrome，对这个系统来说，那就是一台不同的设备。如果你重置手机、换了新手机，或者把电脑重新格式化，那也算是不同的设备。如果 Bob 重新安装应用，原来的密钥可能就没了，诸如此类。

00:17:50 · Herb Caudill

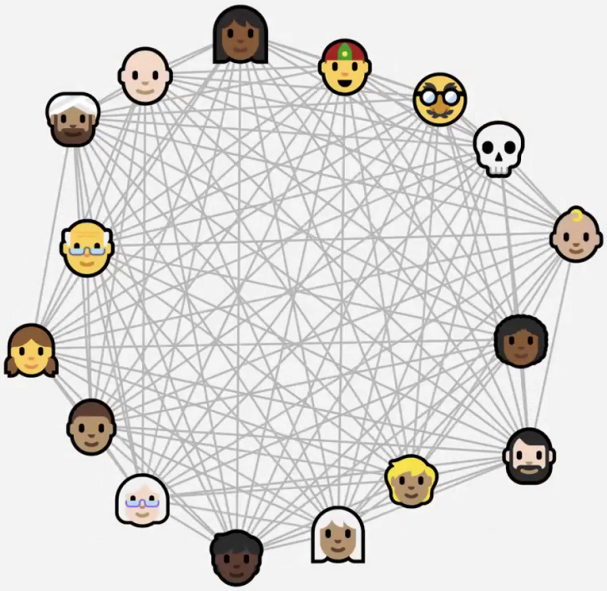
假设 Bob 重新安装了应用，随后把一个新的公钥发给 Alice。Alice 仅凭这个公钥，根本无法判断自己是否仍然在和 Bob 通信。于是，她也得信任这次连接。所以，这并不只是“首次使用时信任”，而是首次使用时要信任，此后每当设备被攻破，或者换了新浏览器等情况发生时，又得重新信任。



带外核对二维码 • 画面 00:18:51

00:18:26 • Herb Caudill

WhatsApp 和 Signal 处理这个问题的方式，是让用户核对安全码或安全号码。这基本上就是 PGP 密钥签名聚会的数字版。可大多数用户根本不会去核对。我自己就是个极客，也从来没做过。所以，连我都不做的话，我觉得大概也没什么人会去做。

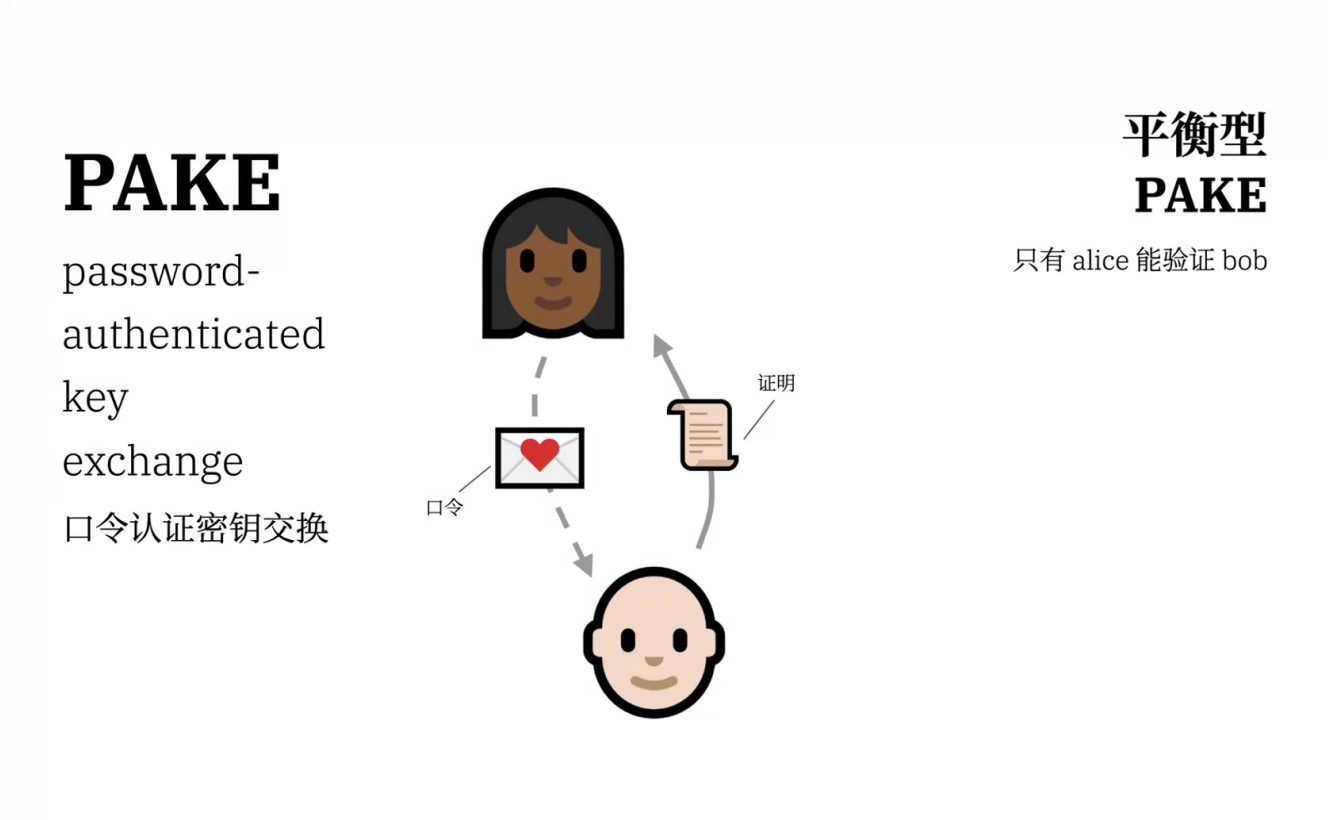


两两核验的扩展成本 · 画面 00:19:23

00:18:54 · Herb Caudill

这种方法真正的问题，是它没法扩展到多人群组。两个人之间核对还好，只需要验证一次。但如果有四个人，就要核对六次；如果有十六个人，就得核对一百二十次。只要群组稍有规模，这件事就会变得完全无法管理。

PAKE 与 Seitan 邀请协议



PAKE口令认证密钥交换 • 画面 00:20:51

00:19:29 • Herb Caudill

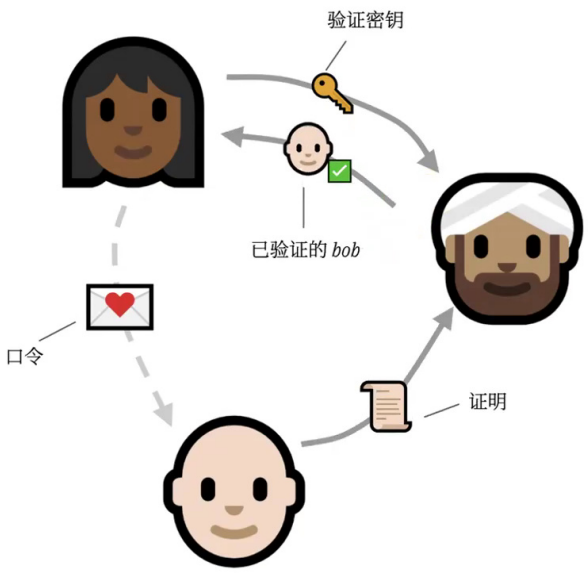
如果 Alice 想与 Bob 建立连接，却不知道他的公钥，有没有办法不依赖首次使用时信任，也能保障第一次连接的安全？一种办法是，Alice 通过带外信道分享一个一次性口令。可以当面告诉 Bob，可以发邮件，也可以使用她平时已经在用的、与 Bob 联系的其他渠道。随后，Bob 的设备连接过来，向 Alice 的设备提供一份密码学证明，证明自己知道这个口令。这是一类叫 PAKE 的协议，全称是 Password-Authenticated Key Exchange，也就是口令认证密钥交换。

00:20:17 • Herb Caudill

这里的思路是，你有一个我们称为“口令”的东西，通常会以邀请码的形式呈现。这个例子采用的是所谓的平衡型 PAKE：Alice 也知道口令，因此她能够验证 Bob。它的缺点是，邀请人和受邀者必须直接连接，才能完成这次交换。但如果你想邀请一个人加入团队，之后再让另一位团队成员接纳他，该怎么办？

非对称 PAKE

charlie 可以代表 alice
验证 bob



非对称PAKE委托认证 • 画面 00:21:44

00:21:04 • Herb Caudill

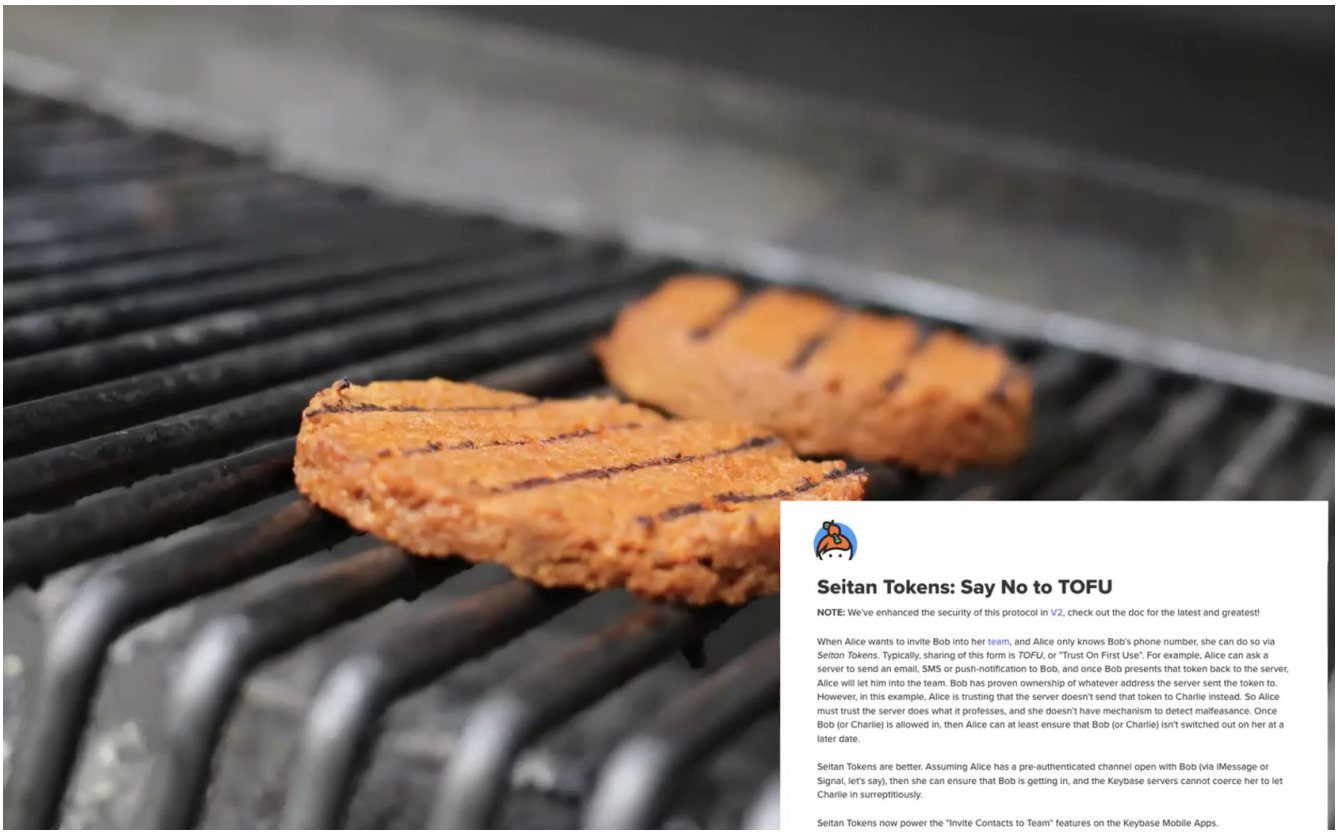
另一种办法是我们所说的非对称 PAKE。它加入了一些密码学机制，使第三方也能够完成验证。Alice 把口令或邀请码交给 Bob，同时把从口令派生出的材料交给 Charlie，而不是把口令本身交给他。接着，Bob 用口令生成一份证明，发送给 Charlie。Charlie 验证这份证明，再向 Alice 确认 Bob 的身份。Charlie 自己不需要知道口令，也能做到这一点。这是一个很巧妙的办法。



Keybase群组体验 · 画面 00:22:16

00:21:55 · Herb Caudill

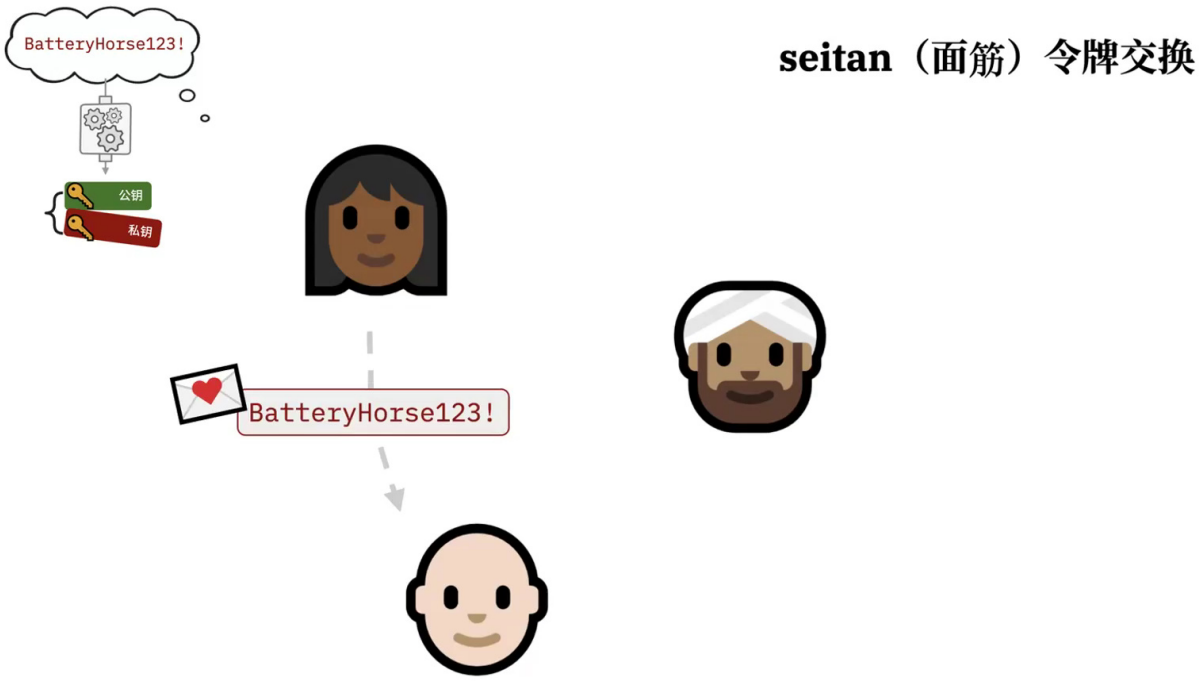
再回到 Keybase。Keybase 已经做了一套用于社交身份证明（social proofs）的系统。他们最初计划通过一款类似 Slack 的加密团队聊天应用来获得收入，让公司或者公司内部的团队用它沟通。但他们意识到，不可能让公司里的每个人都把自己的公钥放到 Twitter 或 GitHub 上。他们需要另一种办法来建立最初的密钥交换。



Seitan代替TOFU • 画面 00:22:56

00:22:36 • Herb Caudill

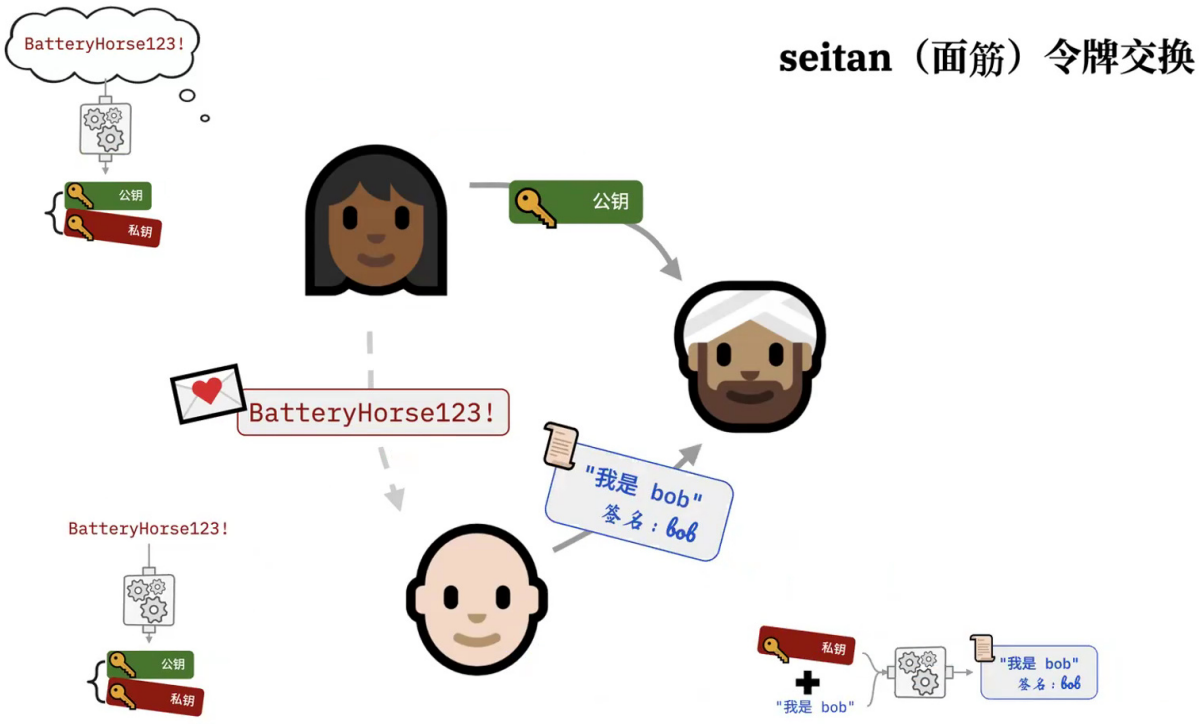
于是，他们提出了一种叫 Seitan 的方案。这个玩笑大概在于：seitan，也就是面筋，比 tofu——豆腐——更结实。我在 2020 年思考这些问题时发现了这个办法，后来采用的也是它。它能够提供我们想要的第三方验证。下面这张图简化展示了 Seitan 的工作方式。



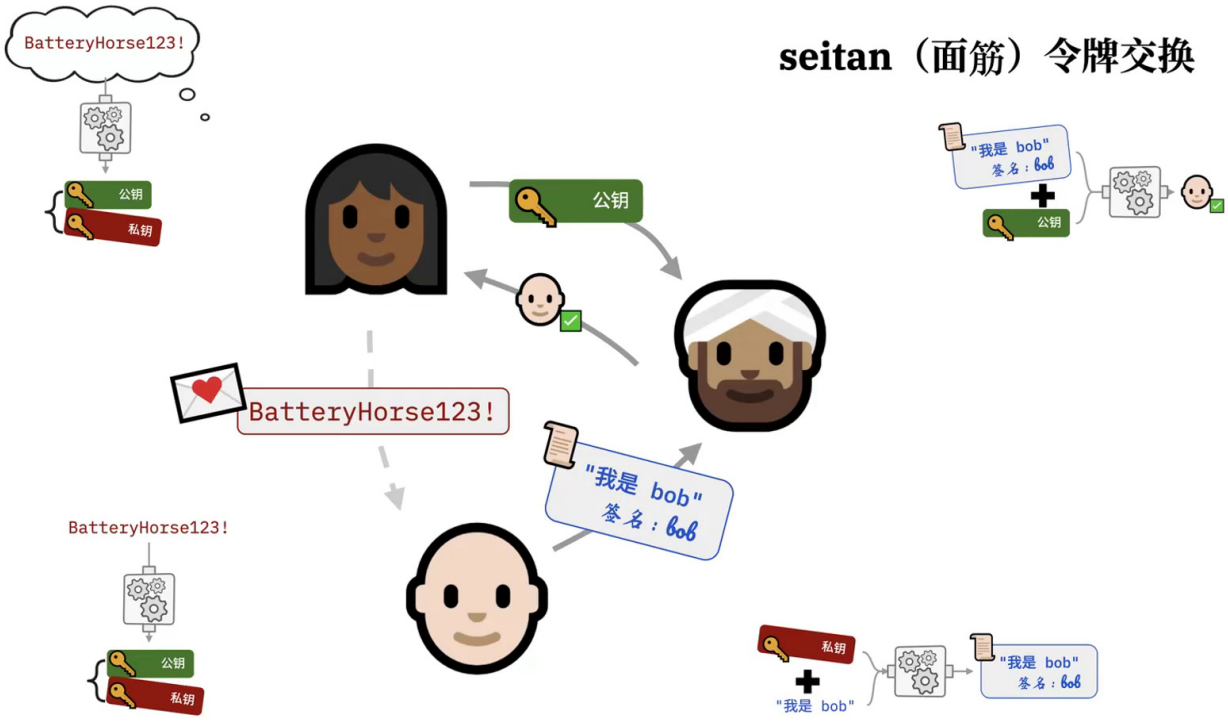
从邀请口令派生密钥 • 画面 00:23:29

00:23:13 • Herb Caudill

Alice 创建一个邀请口令，并从中派生出一对签名密钥。她先把派生出的密钥放在一边，然后通过某个已有的可信渠道，把口令交给 Bob，邀请他加入。比如，在电话里把口令念给他，或者发一封邮件，等等。她把派生出的公钥交给 Charlie，这样 Charlie 之后就能验证这个邀请。



传递邀请与认证公钥 · 画面 00:24:01



Seitan令牌交换完成 · 画面 00:24:51

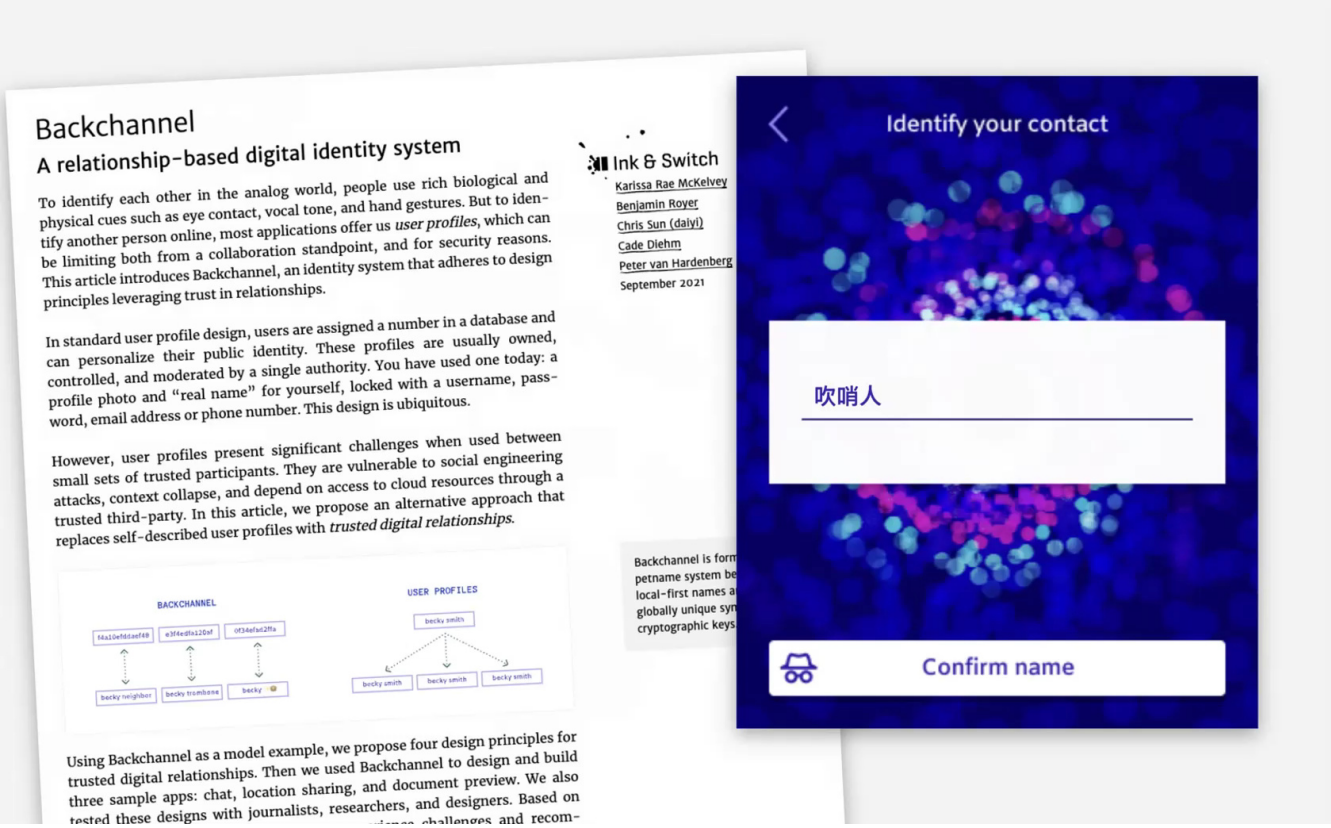
00:23:44 • Herb Caudill

现在，Bob 用这个口令派生出同样的一对签名密钥。他取出其中的私钥，写一份说明自己身份的文档，再用私钥为它签名。这个签名就证明了他知道邀请口令。Bob 把它发给 Charlie。Charlie 拿到 Bob 签名的声明，用 Alice 给他的公钥验证，就能确认：这确实是 Alice 邀请的那个人。等他和 Alice 再次同时在线时，就可以告诉 Alice，他已经验证了 Bob。

00:24:29 • Herb Caudill

这里出现的所有密钥，都是从这次邀请中派生出来的，也都只用于这次邀请。没有人需要为了完成这个过程而到处传送自己的密钥。所以，最初的信任仍然依赖于 Alice 通过已有渠道，把邀请码交给正确的人；但第三方也能够验证这个邀请。

身份建立在人与人的关系上



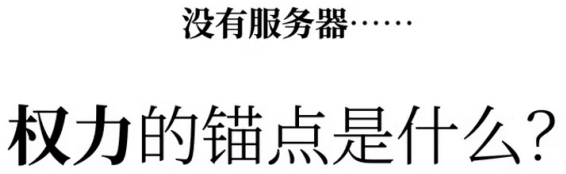
Backchannel关系型身份 • 画面 00:26:03

00:25:04 • Herb Caudill

我们一直在讨论身份，但这里的身份，并不一定是你在现实世界里叫什么名字。它只是指：对我来说，你是谁。我知道是你，因为邀请是我交给你的。我就是通过这一点，确认自己正在和谁打交道。

00:25:25 • Herb Caudill

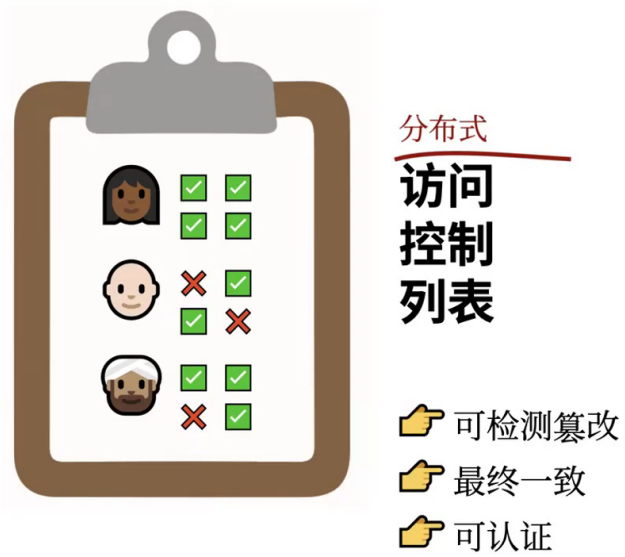
大约在那个时候，Ink & Switch 有一个叫 Backchannel 的项目。促成这个项目的一个使用场景，是让记者联系匿名消息来源，比如一位不愿暴露身份的举报人。记者不需要知道对方的名字，对方可以继续保持匿名。信任和身份植根于这两个人之间的互动：我知道你是谁，因为我们已经有过互动。我们可以利用这种互动，用密码学保障连接的安全，再通过这些连接形成一张信任关系网络。



权力的锚点 · 画面 00:26:41

00:26:13 · Herb Caudill

这就回答了我们的第一个问题，而且这个答案让我觉得是对的：身份的锚点，是人与人之间的互动和关系。稍后我们会再回到这一点。现在，我们有办法知道自己正在和谁交谈了，但接下来，我怎么知道谁是我的团队成员？怎么知道该和谁共享？怎么知道谁可以做什么？又是谁来决定谁可以做什么？在这种环境下，这些问题该怎么解决?



分布式访问控制列表 • 画面 00:27:53

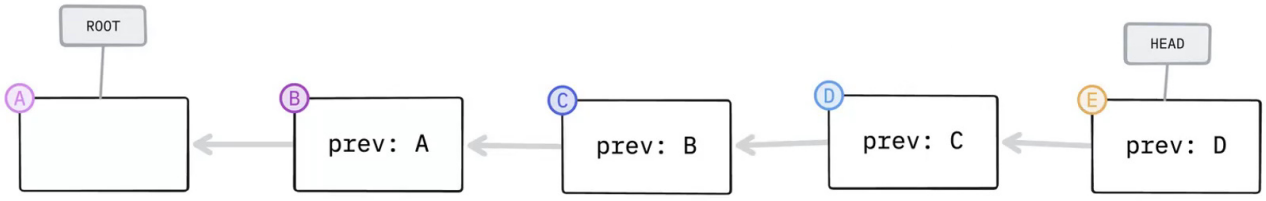
00:26:50 • Herb Caudill

我最初想到的是做一个访问控制列表。服务器用的就是这个：一份列出身份及其权限的清单。我一开始觉得，在去中心化系统里，每个人只要有一份这个列表的副本就行了。也许把它放进一个 Automerge 文档里，再同步出去就可以了。

00:27:19 • Herb Caudill

问题在于，这个列表必须能够检测出篡改：如果有人改动了文档，我们得能看出来。它还必须具备最终一致性，让所有收到相同变更的人，最终得到相同的访问控制列表。变更也必须能够认证来源，让我们知道是谁做的。因此，我们需要的结构，不能只是一个普通的 CRDT。

哈希链



✅ 可检测篡改

哈希链 • 画面 00:28:46

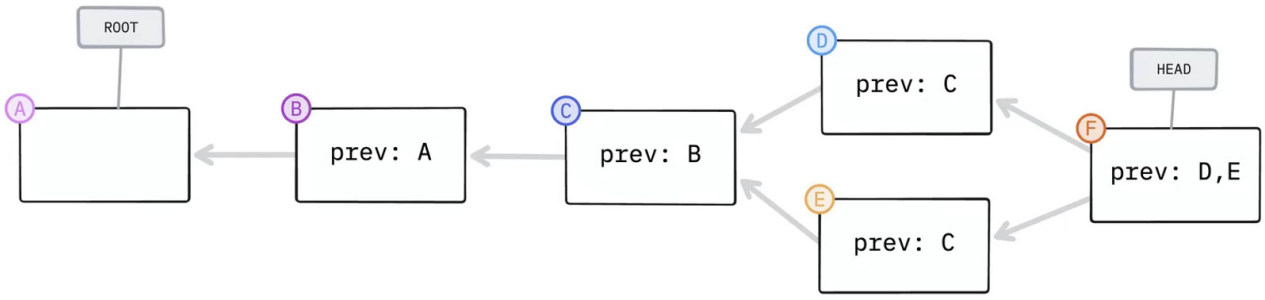
00:27:58 • Herb Caudill

要构建一份能够检测篡改的历史记录，首先可能会想到哈希链。我们从第一个节点 A 开始。下一个节点 B 包含 A 的密码学哈希值，接着 C 包含 B 的哈希值，D 包含 C 的哈希值，E 包含 D 的哈希值。E 是这条链的头部。如果前面某个节点的内容发生变化，其后所有的哈希引用都会失效。区块链就是这样工作的。之所以能够发现篡改，是因为我们有一个可信的参照，可以拿来核对。

00:28:41 • Herb Caudill

问题是，链是线性的，它没有提供合并并发分支的办法。区块链的解决方式，是要求你消耗大量能量，来以正确的方式解决冲突。但在我们讨论的这个环境里，该怎么做呢？

哈希图



✅ 可检测篡改

✅ 最终一致

哈希图 • 画面 00:30:19

00:29:03 • Herb Caudill

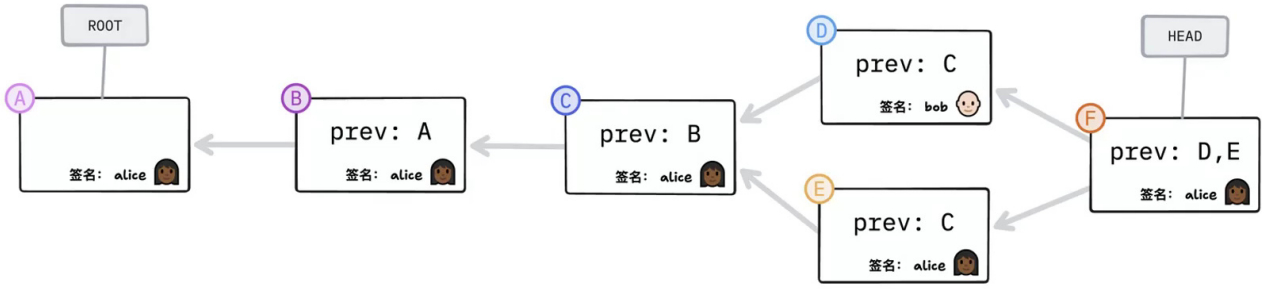
哈希图引入了分支，而分支让我们能够表示并发变更。还是从 A 开始，B 指向 A，C 指向 B。现在，假设一个人添加节点 B，另一个人同时添加节点 E，它们都指向 C。接下来，一个知道之前所有节点的人，就可以同时引用两个节点，而不只是一个。这样，实际上就完成了图的合并。

> 编校注：两份自动转录在此均记录为节点 B，而对应画面把这个分支标为 D。图中并发的两个分支是 D 和 E。

00:29:50 • Herb Caudill

配合适当的冲突解决机制，我们就能以最终一致的方式复制变更。这就是一种 CRDT：它让各个节点合并自己的变更，并得到相同的结果。Automerge 就是通过变更历史来支持这种复制的，这正是它的工作方式。Git 也有带分支的历史，只不过冲突需要手动解决。所以，这是一个已经被充分理解的做法。

签名图



✅ 可检测篡改

✅ 最终一致

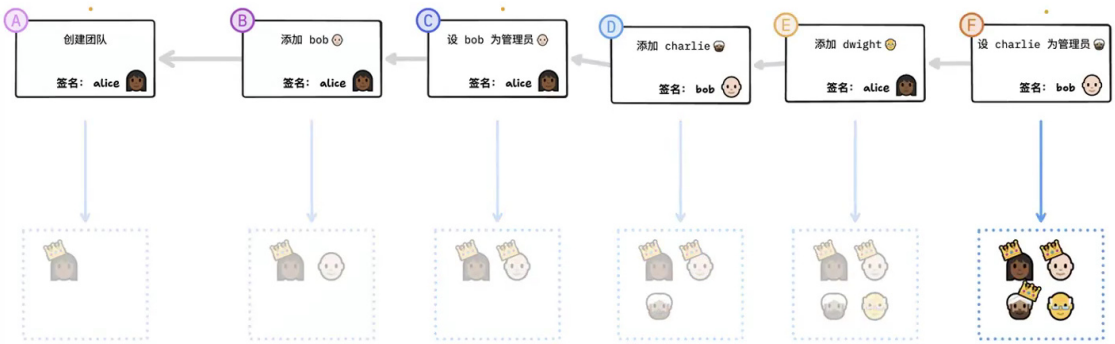
✅ 可认证

签名图 · 画面 00:31:23

00:30:27 · Herb Caudill

我们还需要一种方法，验证每次变更是谁做的。签名图是在哈希图上增加了一点：每个操作都由其发起者进行密码学签名。Alice 为第一个操作签名，也为 B 和 C 签名。Bob 在一个分支上为 D 签名，Alice 在另一个分支上为 E 签名；随后，Alice 为合并两个分支的 F 签名。这样，我们既能验证每个操作是谁签名的，也能检测历史是否被改动。这里发生过的任何事情一旦被篡改，都会显现出来，因为签名验证会失败。

签名图



签名图中的群组操作 • 画面 00:32:41

00:31:13 • Herb Caudill

现在，我们具备了所需要的三种性质：可检测篡改、最终一致，以及可认证来源。不过，我们仍然需要规则，来规定哪些操作是被允许的。这个问题稍后再说。签名图记录的是改变团队的操作。下面看看它在实践中具体是怎么运作的。

成员管理、授权与密钥轮换

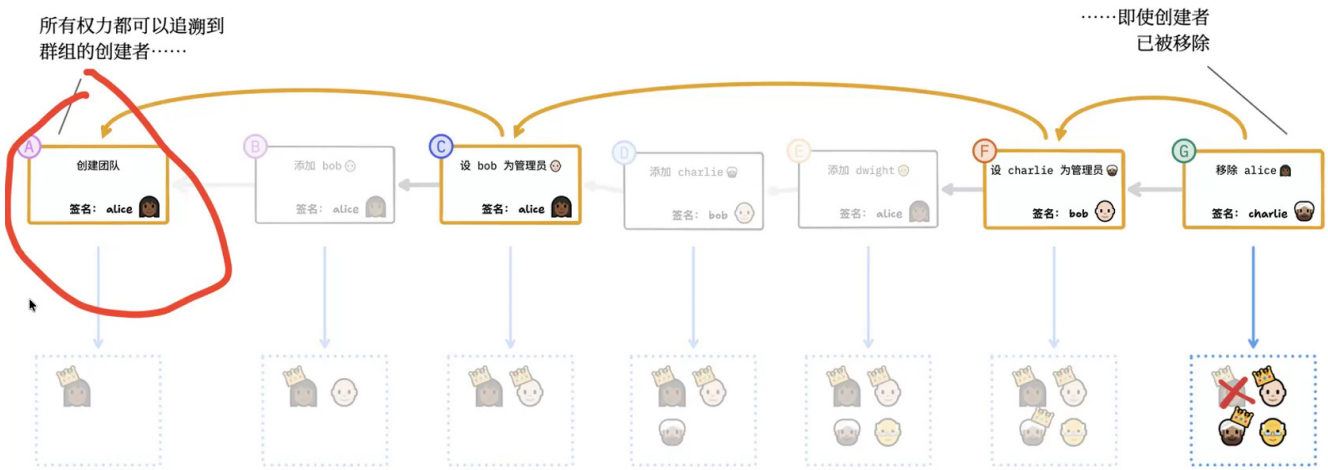
00:31:52 • Herb Caudill

比如，Alice 创建了团队，因此由她决定谁能加入。此时，她是这个群组的绝对独裁者。当然，她也是唯一的成员，所以这无所谓，但她确实掌握着全部权力。然后，她添加了 Bob。这时，群组里仍然只有 Alice 是管理员。不过，Alice 可以在链上增加一个操作，把 Bob 提升为管理员，也就是把自己的一部分权力授予他。

00:32:32 • Herb Caudill

这样一来，Bob 就可以添加 Charlie。假设 Alice 又添加了 Dwight，Bob 把 Charlie 提升为管理员，那么 Charlie 现在甚至可以移除 Alice，因为 Alice 授予出去的权力，最终传到了他手里。

签名图

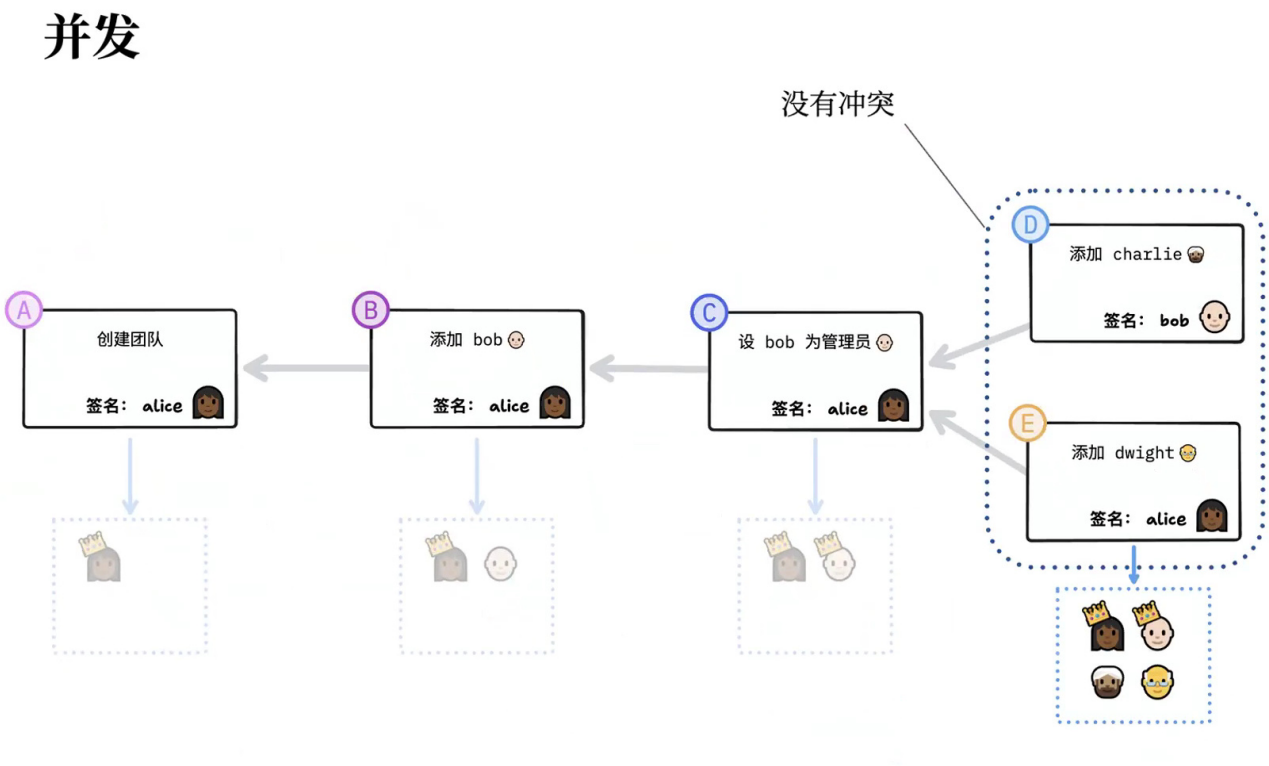


移除成员与密钥轮换 • 画面 00:33:45

00:32:55 • Herb Caudill

对我来说，关键的认识是：图的根节点，也就是链上的第一个节点，锚定了这个群组中的权力。Alice 创建了团队，因此她可以决定谁能加入。即使后来她被移除了，群组里的每个操作，依然可以追溯到最初创建这条链的节点。图的根节点为权力提供了锚点。每一次有效的添加、移除，以及后续的授权，最终都可以追溯到同一个权力来源：这个群组是 Alice 创建的。

并发冲突：添加与移除谁优先



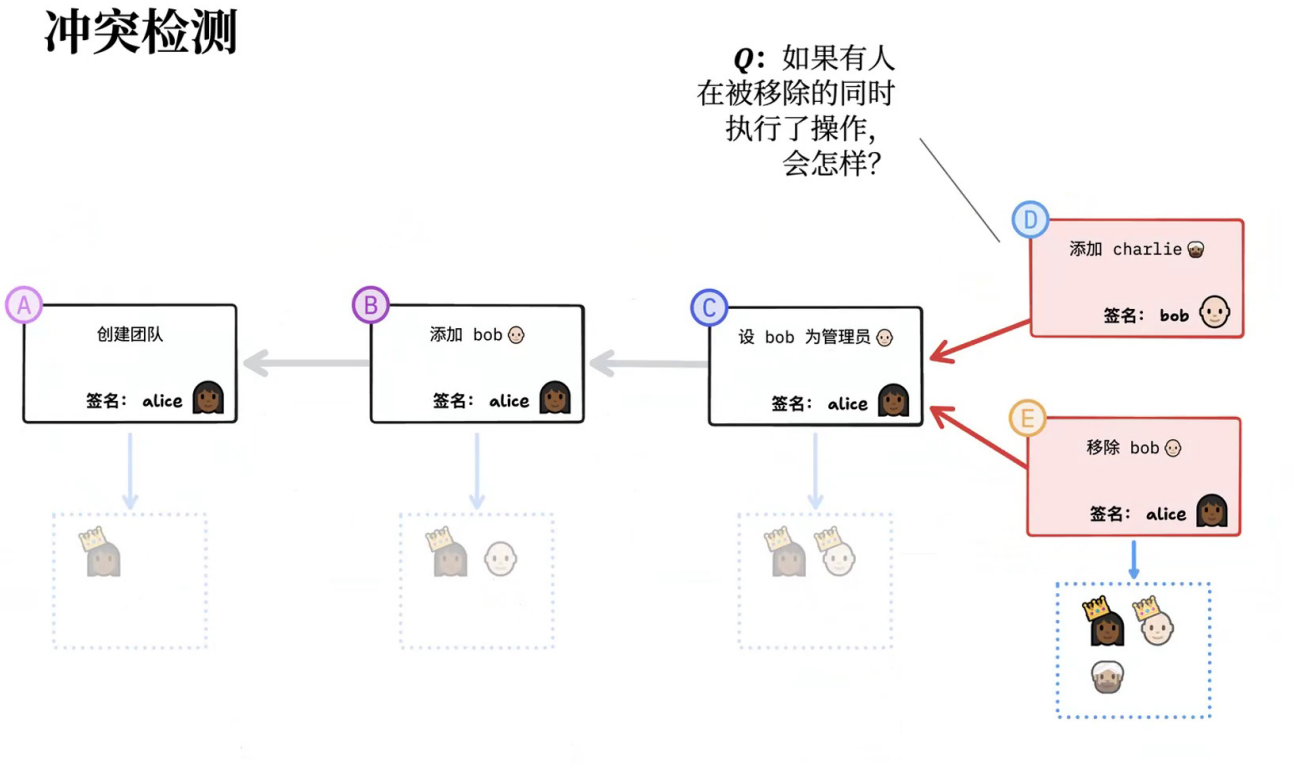
并发且无冲突的操作 • 画面 00:35:09

00:34:05 • Herb Caudill

要让各个节点在断开连接时仍然能够作出变更，我们就必须处理并发。Automerge 这样的普通 CRDT 有一个问题：它并不提供这种场景所需要的权限规则。

00:34:34 • Herb Caudill

在很简单的情况下，确实可以用 Automerge 这样的工具。Alice 创建团队，添加 Bob，并把他设为管理员。之后，Alice 和 Bob 离线了，每台设备各自保留一份历史。Bob 添加 Charlie，Alice 则独立添加 Dwight。等两台设备重新连接时，把两份历史合并就行了。这些操作没有冲突，按哪个顺序应用都可以。到这里为止，普通 CRDT 完全能够处理。



并发移除与冲突检测 · 画面 00:36:03

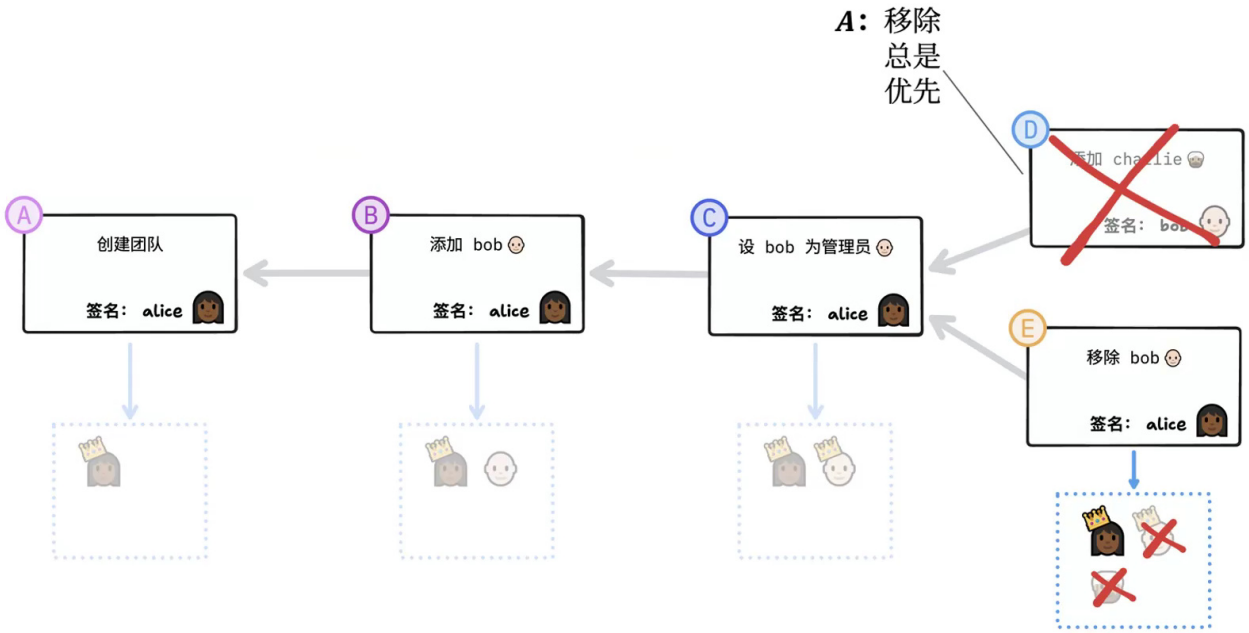
00:35:17 · Herb Caudill

但假设 Alice 创建团队、添加 Bob，并把 Bob 设为管理员后，Bob 执行了一个操作，而 Alice 恰好同时在移除他，会怎样？比如，Bob 把 Charlie 加入群组；在双方仍然断开连接的时候，Alice 又把 Bob 移除了。等到合并历史时，就产生了冲突：Charlie 到底算不算群组成员？这取决于这两个操作按什么顺序处理，或者保留其中哪些操作。正确解决这个冲突很重要，否则 Bob 就可以通过添加另一个由自己控制的身份，绕过对自己的移除。

00:36:10 · Herb Caudill

我采用的解决办法，是在冲突中让移除操作优先。这就是 Martin Kleppmann 所说的“强移除”（strong removal）：Bob 并发执行的添加 Charlie 的操作不生效，因为他本人正在同时被移除。

冲突解决

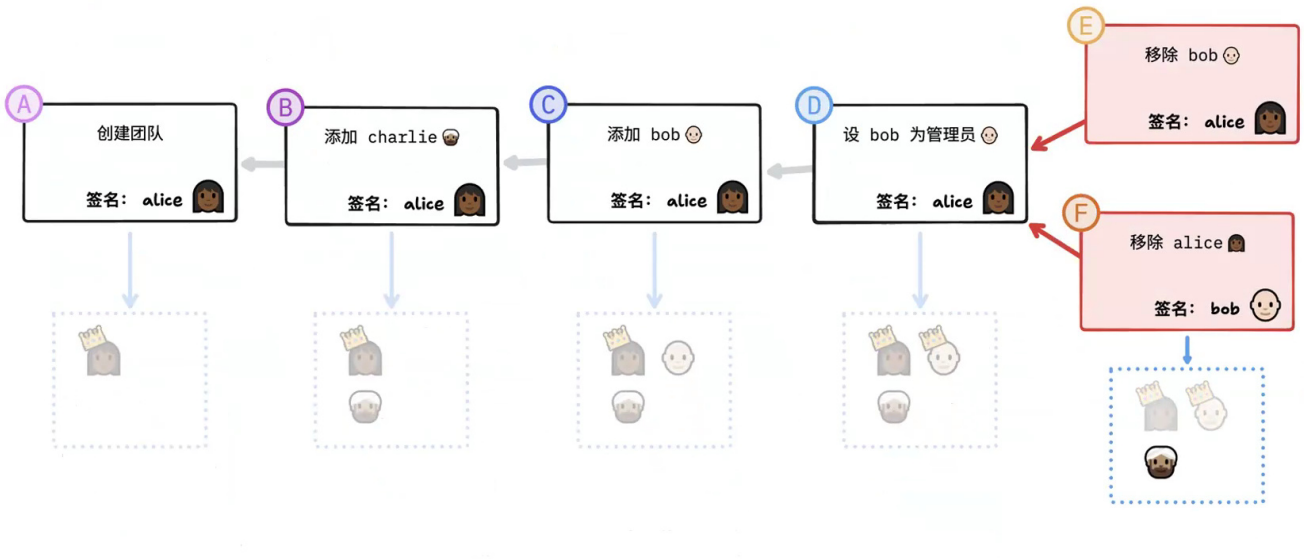


移除优先的冲突解决 • 画面 00:36:53

00:36:33 • Herb Caudill

Automerge 这样的通用 CRDT 不能提供这种逻辑。实际上，它甚至不会把这两个操作视为冲突。在 Automerge 里，冲突是同时修改同一个属性；而这里发生的是两件不同的事，它们在逻辑上互相冲突。这意味着，你的系统需要一种专门为群组成员关系设计的定制 CRDT。

当管理员相互移除

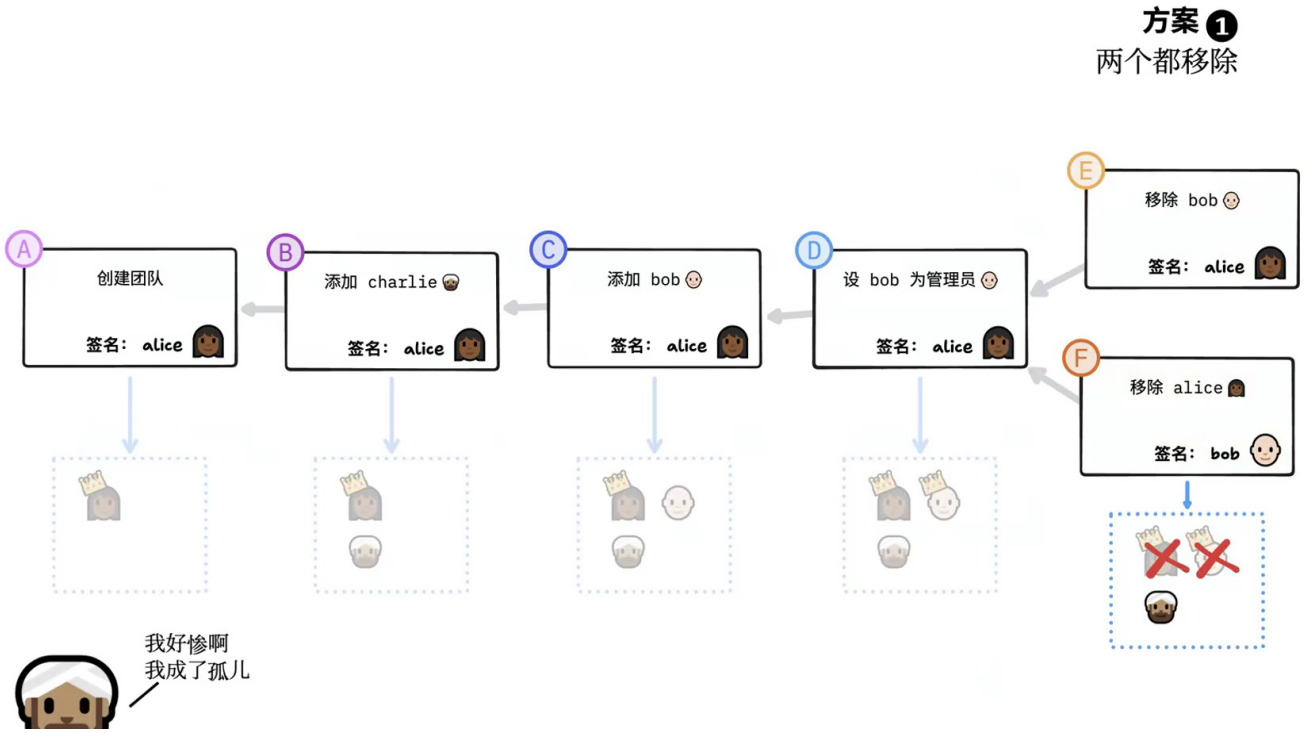


管理员相互移除 • 画面 00:37:46

00:37:07 • Herb Caudill

还有其他可能发生冲突的情况。你大概已经想到了一个很明显的问题：如果两位管理员同时移除对方，会怎样？Alice 创建团队、添加 Charlie，等等。现在 Alice 和 Bob 断开了连接，Bob 试图移除 Alice，而 Alice 也试图移除 Bob。这时候怎么办？我们遇到了一个更难解决的冲突。

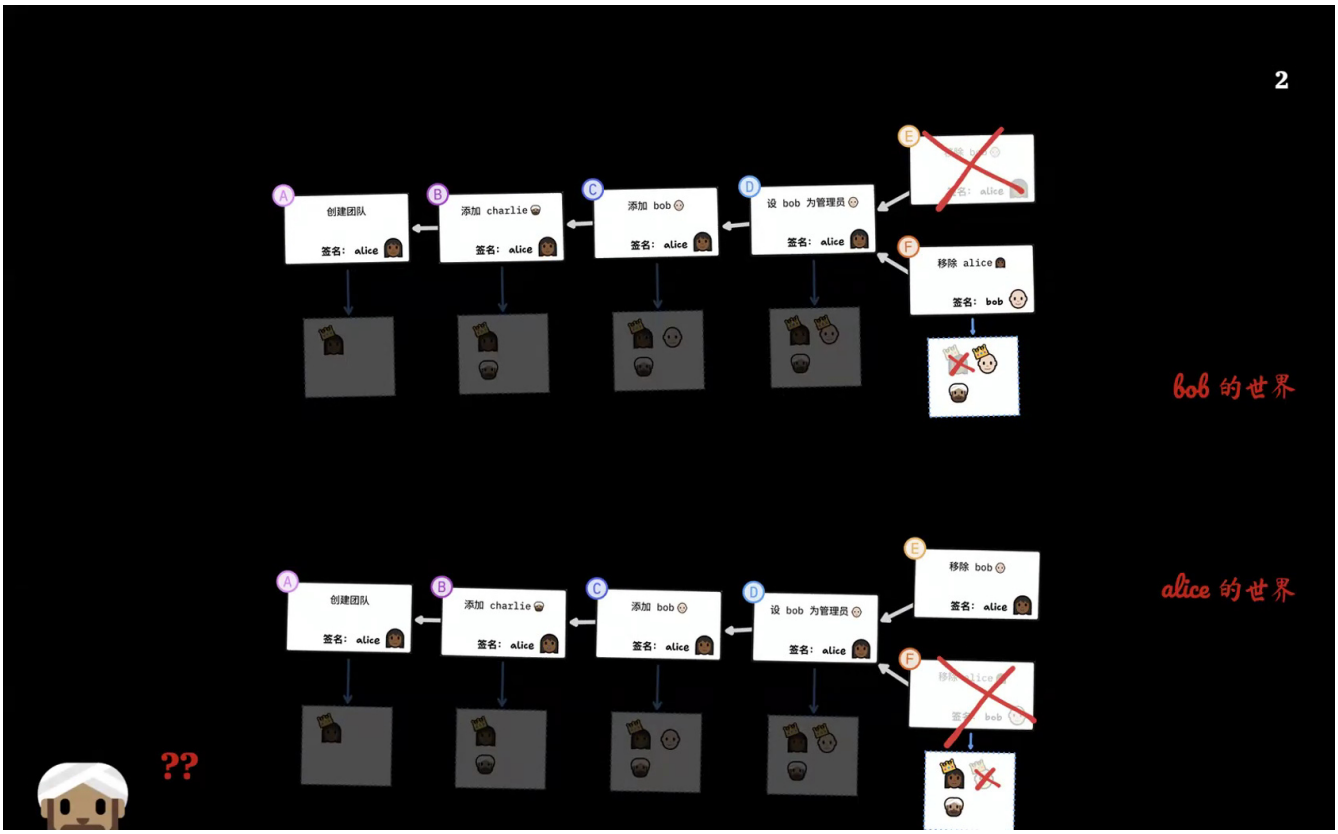
相互移除



方案一同时移除 • 画面 00:38:20

00:37:43 • Herb Caudill

一个办法是沿用刚才的方案，也就是强移除，把两位管理员都移除。问题在于，群组里只剩下 Charlie，而他并不是管理员。这显然不是我们想要的结果，因为这个群组已经没法正常运作了。它也给拒绝服务攻击留下了可乘之机：恶意设备可以不接受自己被移除，而是让整个群组失去管理员。



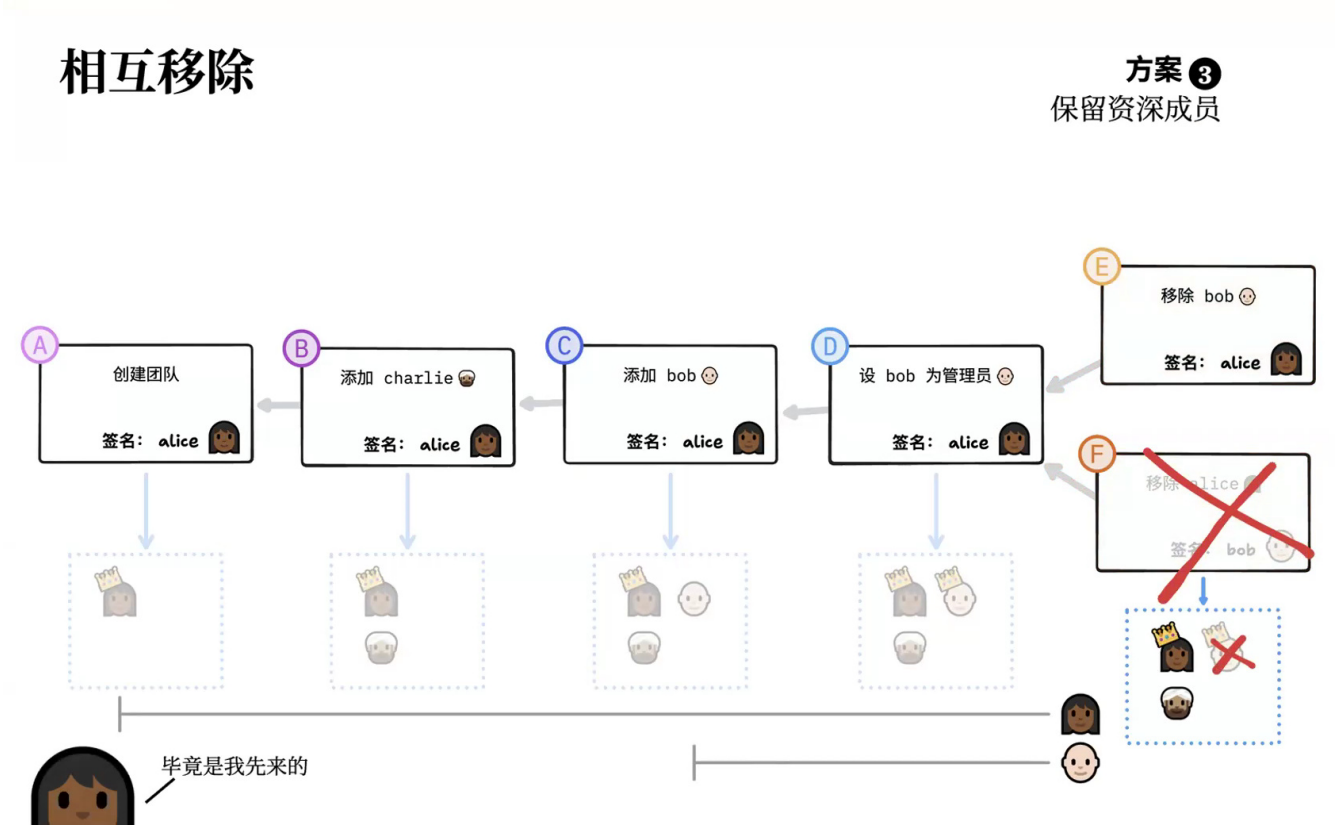
方案二分裂世界 • 画面 00:38:45

00:38:26 • Herb Caudill

另一个办法，是把世界分成两个版本的同一个群组，让其余成员自己决定要加入哪一个。从某种意义上说，这很巧妙，因为它反映了去中心化系统的现实。但它把一个决定留给了成员。就像天主教会曾经分裂了几年，同时有两位教皇，一位在罗马，一位在阿维尼翁，每个天主教徒都得自己决定，相信哪一位才是正统教皇。所以，这也不算一个特别好的解决办法。



教会分裂的类比・画面 00:39:08

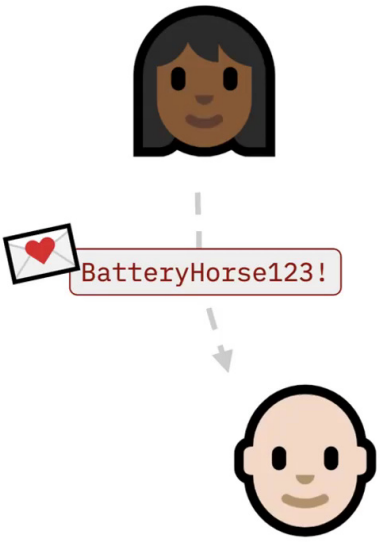


方案三以资历处理相互移除・画面 00:39:32

00:39:14 • Herb Caudill

我最后采用的办法，是用加入群组的资历，作为判断谁更可能正确的一个替代指标。图里的线表示 Alice 和 Bob 分别在群组里待了多久。Alice 加入得更早，因此她移除 Bob 的操作生效。这就是我在自己的库里作出的选择。

身份的基础与权力的来源



真实关系锚定身份 • 画面 00:40:41

00:39:39 • Herb Caudill

一开始，我们提出了两个大问题：在本地优先的世界里，怎样找到立足点？第一个问题是，身份靠什么锚定；第二个是，权力靠什么锚定。这个邀请协议涉及很多微妙的密码学细节，但它的核心，其实就是 Alice 把邀请码交给 Bob 的那个互动。我最初设计这个系统时，觉得这有点像在作弊，不像是一个从原则出发的解决方案。

00:40:18 • Herb Caudill

但我想得越多，就越觉得这是对的。那个原本就存在的带外渠道，不管是邮件、电话还是当面见面，恰恰表明了最初的信任已经存在。这不是无关紧要的东西；在这个场景里，它正是信任的根本来源。信任之所以存在，是因为 Alice 和 Bob 在现实世界里已经有某种关系。正是这份既有的关系，为身份提供了锚点。

没有服务器……

Q.

身份的锚点是什么?

Q.

权力的锚点是什么?

重新回到两大问题 · 画面 00:41:19



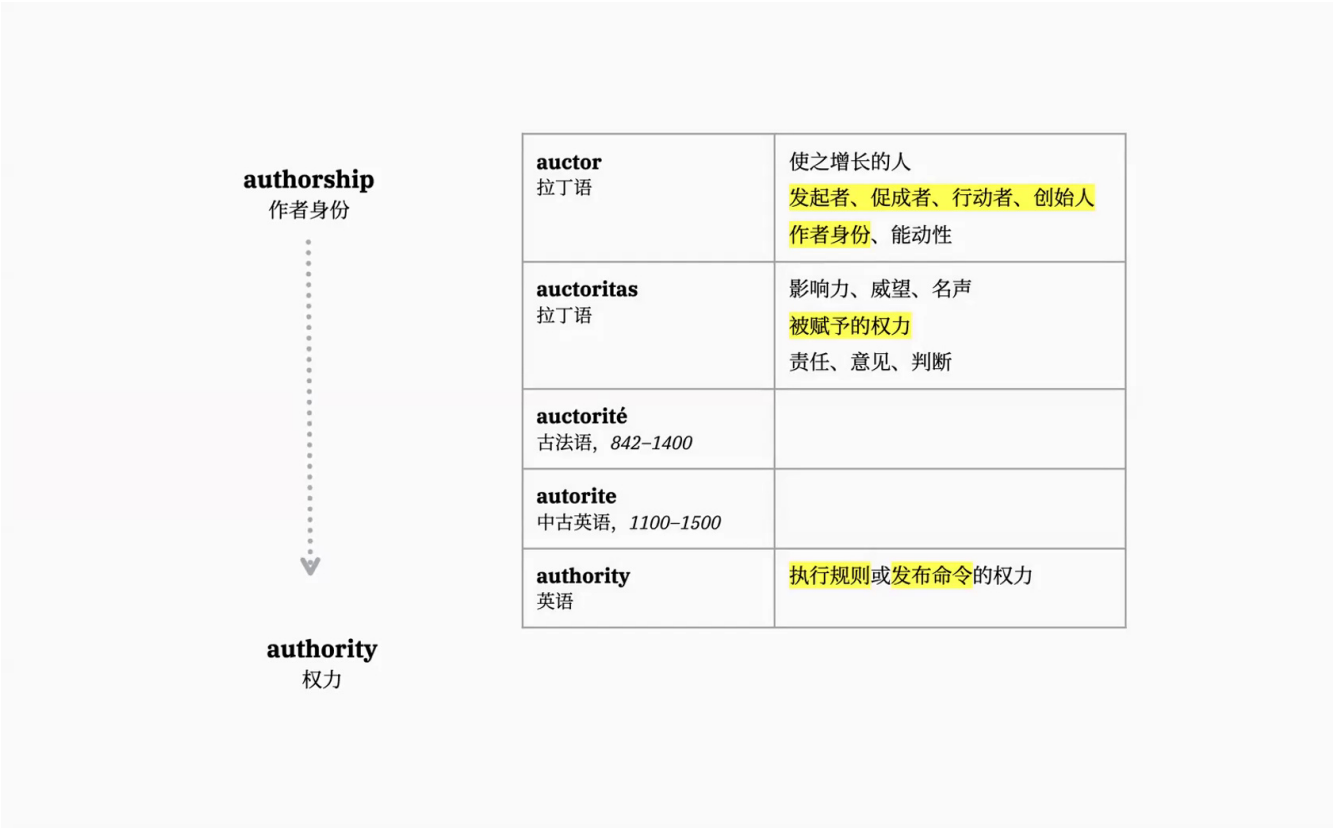
创造行为锚定权力 · 画面 00:42:16

00:41:12 • Herb Caudill

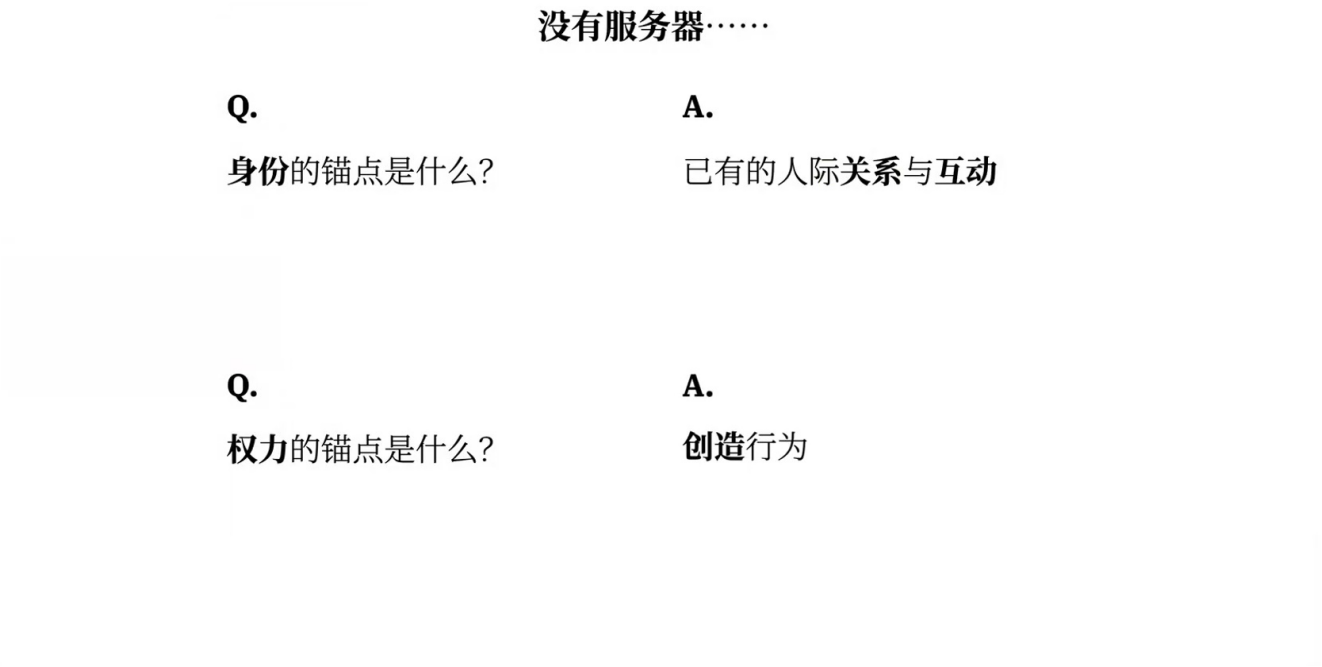
这就是第一个问题的答案：什么锚定了身份。那么，什么锚定了权力？再回到这些链：权威的根基是创建这一行为。无论是你创办的公司、发起的项目，还是创建的文档，创建它，就使你最初拥有对它的决定权。这并不意味着你必须把权力全部握在自己手里，你可以把权限委托给其他人。

00:41:56 • Herb Caudill

这甚至不意味着你必须永远保有这种权威。你可能创建了某样东西，后来又离开了，就像公司的创始人最终可能离开，但公司仍会继续运转。即便如此，这些权限最终都可以追溯到创建公司的人，哪怕他已经不在公司了。即便公司里后来有人比创始人更有权力，这些权力也都来自创始人最初的授权。



Authorship与Authority • 画面 00:42:51

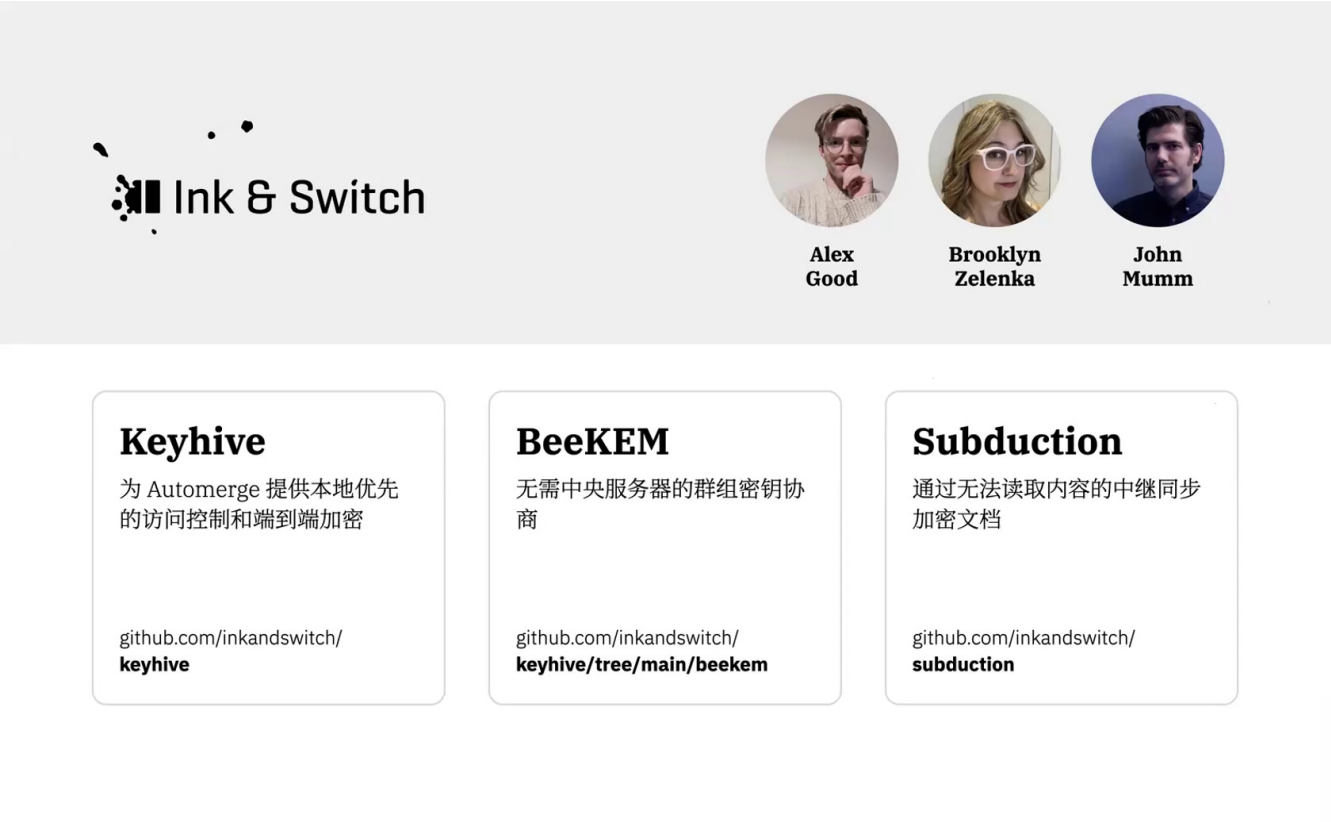


身份与权力的答案 · 画面 00:43:11

00:42:36 · Herb Caudill

在英语里，这种联系就体现在词语本身：authorship，也就是创作或作者身份，以及 authority，也就是权威，都与创建某样东西、对自己创建的东西拥有决定权这个意思有关。这就是我找到、并在自己的库里用代码实现的答案：人际关系和互动确立身份，创建行为确立权威。我觉得，这些问题因此得到了正确、令人满意的回答。

Keyhive、BeeKEM 与 Subduction



Keyhive BeeKEM和Subduction • 画面 00:44:21

00:43:20 • Herb Caudill

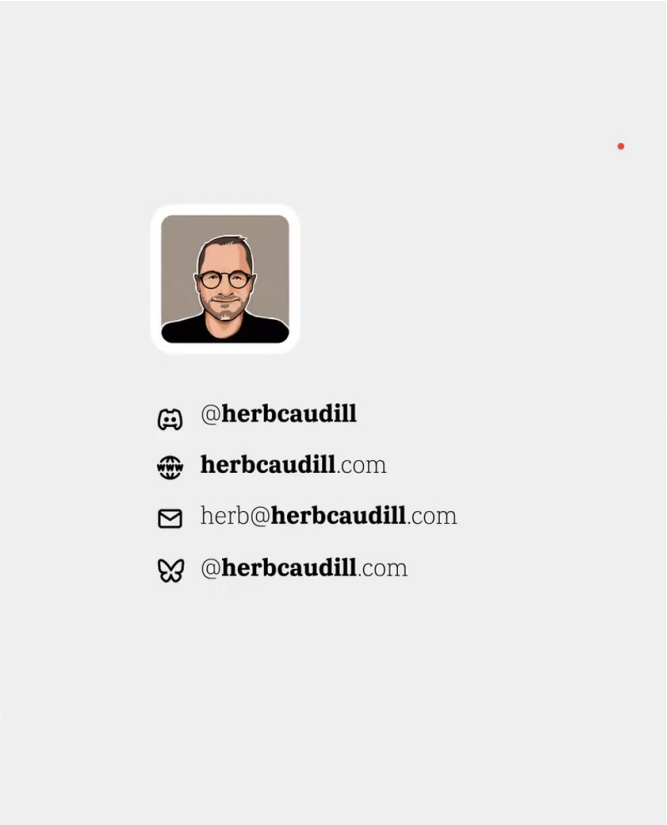
我的库在理念上的后继者是一个叫 Keyhive 的项目。它采用了基本相同的想法，并以更适合生产环境的方式来实现。不过，我要说明，这是它的目标；Keyhive 现在仍处于 alpha 阶段。但我已经用过，它能运行，应该很快就能正式推出、供大家使用。

00:43:58 • Herb Caudill

这里还有另外几个组成部分。一个叫 BeeKEM，是单独的库，或者说是这个库里的一个独立软件包。它建立了一种密码学结构，用来处理群组里的所有密钥及其历史。另一个是 Subduction，它是一套同步协议和同步服务器，能够转发加密文档，却无法访问文档的内容。

00:44:36 • Herb Caudill

这些都是非常棒的技术工作。现在做这些的人比我聪明得多，他们是真正的密码学家，而不是像我这样在疫情期间才开始学习密码学的人。我推荐大家去看看这些项目。我想分享的就这些。这里是我的联系方式，很乐意继续和大家讨论这些问题。



联系方式与问答开始 · 画面 00:45:31

00:45:07 · Herb Caudill

我看到聊天区里有一个问题。

00:45:12 · 主持人

是的。

问答：贡献度与权限策略

00:45:17 · Herb Caudill

问题是：如果这条链里有一个为每个用户记录贡献度评分的系统，是不是也可以根据贡献来决定权力？也就是说，在链里加入贡献度评分，对吧？

00:45:31 · Herb Caudill

可以有不同的解决办法。我认为，相互移除是这里最难的问题。我选择的办法，就是看追溯到某个人被加入时，链上的深度有多大。如果我是在十个链接之前加入的，你是在五个链接之前加入的，那么我加入得更早，在相互移除的情况下，我的操作就会胜出。

00:46:11 · Herb Caudill

这件事也可以有其他解法。比如，你可以设计一个系统，规定 Bob 是 Charlie 的上司，即便 Charlie 入职更早，也让 Bob 在这个冲突中胜出。你可以想办法把这些关系编码进去，但显然，这样就会变得更复杂。

00:46:38 • Herb Caudill

我印象中，Keyhive 把相互移除的冲突解决策略做成了可插拔的，这样你就能替换它使用的逻辑。不过，我不确定它是不是真的已经这样做了。无论如何，这个问题确实有不同的解决办法。

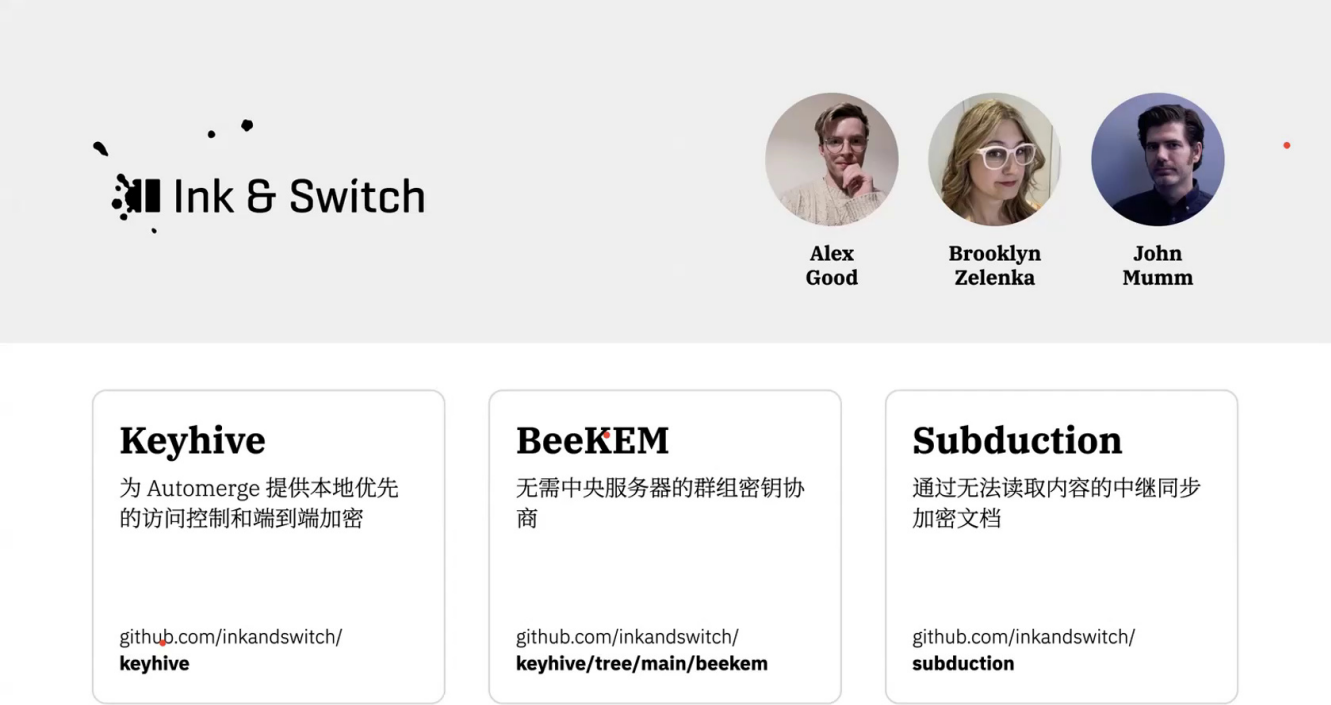
密码学原语与文档权限

00:47:09 • Herb Caudill

还有一个问题：“有没有新的密码学原语，能让建立信任变得更容易？”

00:47:18 • Herb Caudill

我的经验是，在自己的库里，我并不需要发明新的密码学原语。Keyhive 团队确实提出了一些有意思的新原语，其中主要的就是 BeeKEM。



问答回看Keyhive • 画面 00:48:41

00:47:49 • Herb Caudill

我就不详细解释它了，主要是因为我觉得自己未必能说准确。不过，它用一种很巧妙的方式，把密码学密钥之间的关系组织成一棵树，而且效率很高，比我最后在 localfirst/auth 里拼出来的方案高效得多。

00:48:23 • Herb Caudill

另一个问题：“刚才的例子讲的是成员管理，授予文档访问权限时，也使用类似的方法吗？”

00:48:29 • Herb Caudill

是的。在 Keyhive 里，基本上每份文档都是一个群组。你可以建立一个相当复杂的权限委托图。比如说，这里有一份文档，你可以把它和五个人关联起来，这种关联就对应这份文档。

00:49:17 • Herb Caudill

这套系统内部的设计很优雅：文档、群组、设备和个人，基本上都被视为同一类东西。想一想，群组由个人组成，而一个个人又对应一组设备。比如，我可能有一台笔记本电脑、一部手机，以及两个不同的浏览器。

00:49:59 • Herb Caudill

一份文档也可以由多个人访问，所以这些东西在内部都作为同一种结构来处理。我就说到这里，因为我不想讲得不准确。如果能邀请 Alex Good 或 Brooklyn 来和大家交流，详细介绍他们在这个项目中提出的方案，会非常好。

00:50:43 • 主持人

你好，Herb。

与 Secure Scuttlebutt 的异同

00:50:45 • Herb Caudill

请讲。

00:50:45 • 主持人

我有个问题。我们社区里之前有人介绍过一个协议，叫 SSB，也就是 Secure Scuttlebutt，它是一种人与人之间的网络。我很感兴趣的是，它与你的产品，比如 Keyhive，有什么区别？你是怎样思考信任模型的，又是如何推导、构建这些模型的？

00:51:38 • Herb Caudill

是的，它和 Secure Scuttlebutt 有很多相似之处。我当时做这项工作的时候，确实研究过 SSB。

00:51:49 • 主持人

哦。

00:51:50 • Herb Caudill

我已经不太记得 SSB 具体是哪一点不适合这个场景了，但当时确实有这样一个问题。我猜，可能与群组成员管理中的某些具体规则有关。另外，SSB 还有一个问题……

00:52:26 • 主持人

嗯。

00:52:26 • Herb Caudill

从这个角度来说，它是一个效率很低的协议：你仿佛得先拿到整个世界的东西，才能开始做任何事。我自己尝试使用时的体验，就是坐在那里看着加载图标转很久，因为它得先把自己知道的每份文档都加载进来……

00:52:51 • 主持人

嗯。

00:52:51 • Herb Caudill

……然后你才能做事。不过，它是个非常棒的项目。SSB 很棒，我也认识它的一些创建者。

00:53:03 · 主持人

嗯。

00:53:04 · Herb Caudill

它是个非常棒的项目，也可能很适合你的使用场景。

00:53:13 · 主持人

谢谢。听你刚才这样说，我感觉你比较重视建立信任的效率，是这个意思吗？

00:53:29 · Herb Caudill

能再说一遍吗？比较重视……什么？

00:53:31 · 主持人

效率。好的。

00:53:35 · Herb Caudill

对。我觉得更准确地说，是找到一种更高效的方式，在一开始就建立这种信任。

00:53:47 · 主持人

好，谢谢。

线下关系与邀请协议的安全

00:53:56 · Herb Caudill

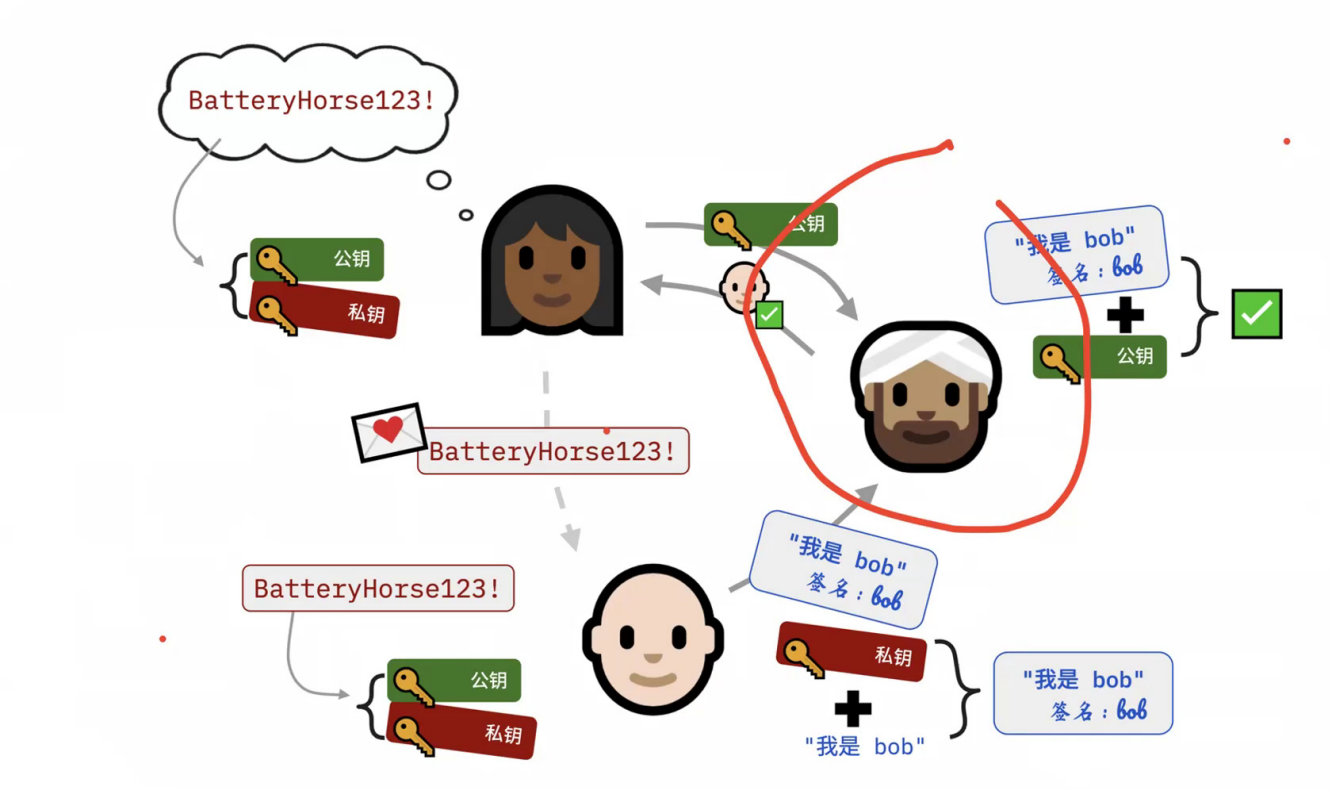
还有一个问题：“我很喜欢把权威建立在现实社会关系之上的这个想法。你从事国际发展工作的经历，是否帮助你认识到线下社会场景的重要性？”

00:54:08 · Herb Caudill

嗯，是的，也许有，肯定有。

00:54:17 · 现场读题

这里有个问题，我来翻译一下：“非对称 PAKE 有没有可能导致介绍人受到攻击？”



问答回看邀请协议与红笔标注 · 画面 00:56:31

00:54:36 · Herb Caudill

你指的是这个场景里的第三方吗？比如这里的 Bob？我不太确定，也许这个问题的翻译还不够清楚。

00:55:05 · 主持人

我说，好像是刚才那位观众的问题，就是介绍人被攻击，introducer 被 attack。你的意思是说 Bob 被 attack 吗？我就是问一下，他好像没有完全懂你的意思。

00:55:33 · 主持人

喂？提问的朋友在吗？

00:55:40 · Herb Caudill

我先回答……

00:55:42 · 主持人

好的。

00:55:43 · Herb Caudill

……在等待的时候，我先回答最新这个问题：“目前使用 Keyhive 和 Subduction，可以让多少个节点一起协作？”

协作规模与自主社区

00:55:51 · Herb Caudill

我不确定目前的上限是多少。但几年前，我们为 Keyhive 制定参数或需求时，目标是支持维基百科那样的规模，包括维基百科的文档数量、用户数量和编辑者数量。

00:56:33 • Herb Caudill

从可扩展性来说，这是一个非常有雄心的目标。我不确定他们现在有没有达到，但它肯定比我的库更能扩展。它完全用 Rust 重写了，而且，还是那句话，是由比我熟练得多的程序员完成的。

00:57:09 • Herb Caudill

这里有一条回复。好的。

00:57:26 • 主持人

嗯，好的，又有一个新问题。

> 聊天补充（来自现场文字聊天，并非口头发言）：一位参与者提出，让现实中互不相识的人建立信任、协作，围绕共同目标或亚文化逐步形成社区，或许能让世界更有韧性。下面的回应承接这条留言。

00:57:47 • Herb Caudill

是的，完全同意。这正是我们的期待。它不仅是一种更好的编程模型，让我们不必依赖科技公司，也使我们能够从零开始建立这些社区，而不依赖别人的服务器或资源。我觉得确实如此。

密钥轮换、身份冲突与恶意成员

00:58:39 • 主持人

有一个问题，问的是规模很大时会怎么样。我想，我会把它理解成一种拜占庭攻击。[此处不清。]

00:59:00 • Herb Caudill

好，也就是说，如果有人重置了自己的私钥……

00:59:06 • 主持人

嗯。

00:59:07 • Herb Caudill

分发密钥时，如果出现身份冲突，该怎么办？在我的库里，密钥轮换的做法是：你必须是管理员，才能发起密钥轮换；或者，你也可以轮换自己设备的密钥。

00:59:32 • Herb Caudill

我认为，在 Keyhive 这种更抽象、更优雅的设计里，基本上是：如果你对一个群组拥有管理权限，就可以轮换这些密钥，其他人则不能这样做。

00:59:58 • 主持人

嗯。

01:00:09 • Herb Caudill

好。抱歉，我正在翻译聊天内容。

01:00:39 • Herb Caudill

再说一遍，身份是这样确定的：如果 Alice 邀请 Bob，那么 Alice 就是在声明，她邀请的这个人是 Bob。也就是说，由发出邀请的人来命名。

01:01:06 • Herb Caudill

当 Bob 接受邀请并提供自己的私钥之后，我们就可以用这个私钥，来识别哪些操作是 Bob 做的。就是这样。

编校注：Soniox 与 Zoom 字幕在这两处均记录为“私钥”，正文保留这一说法，未静默改写。讲者此前介绍的协议要求私钥留在设备内，通过交换公钥验证身份与签名。阅读本段时须区分这两者，不能将这里理解为需要公开私钥。 讲者的协议说明

01:01:36 • Herb Caudill

我想，确实可能有两个人，各自都说：“这是 Bob。”这个库没有把这种情况作为一个主要问题来处理。我觉得这可能仍然是个有待探索的领域。总之，这个库并不试图解决所谓最终身份的问题。

01:02:08 • Herb Caudill

我也看过其他项目，比如 DXOS。它是一个建立在 Automerge 和 Keyhive 之上的库，又增加了一层 DIDs，也就是去中心化身份标识，并把它们与 Bluesky 或 AT Protocol 这样的体系关联起来。这样做，是通过引用某种受信任的身份存储，牺牲一点去中心化程度。这也是一种解决办法。

01:03:06 • Herb Caudill

对于恶意行为，我认为就需要管理员把恶意参与者从群组中移除。这是技术层面的解决办法。除此之外，也可能有其他方式，在系统外告诉别人，某个人不值得信任。

01:03:46 • Herb Caudill

身份问题确实会变得棘手：我们要真正知道一个人在现实世界中是谁，并对此取得一致认识。这里既有分布式方案，也有基于中心服务器的方案。

01:04:16 • Herb Caudill

但 Keyhive 和 localfirst/auth 所依据的威胁模型，假定由邀请某人的那个人，来决定这个人是谁。它们并不处理这样的情况：两个人都声称自己邀请的是同一个人，实际上却是不同的人。不过，这是一个很有意思的问题。

01:05:01 • 主持人

嗯。

01:05:37 • 主持人

好的。还有新问题吗？大家还有问题吗？

01:06:07 • 主持人

好，如果没有其他问题，那也许……哦，又有一个新问题。

公告板与身份撤销

01:06:30 • 主持人

新的问题是：我们能不能事先在公告板上登记一个密码学签名或凭据，等需要的时候，再在公告板上用它来撤销这个身份？

01:06:54 • Herb Caudill

如果这样做，你就预先假定了一个条件：存在一个所有人都信任的中央公告板。

01:07:05 • 主持人

嗯。

01:07:06 • Herb Caudill

这本质上就是中心化的公钥管理。这是一种合理的解决方案，但属于不同的问题范围。

节点发现、同步与中继信任

01:07:39 • Herb Caudill

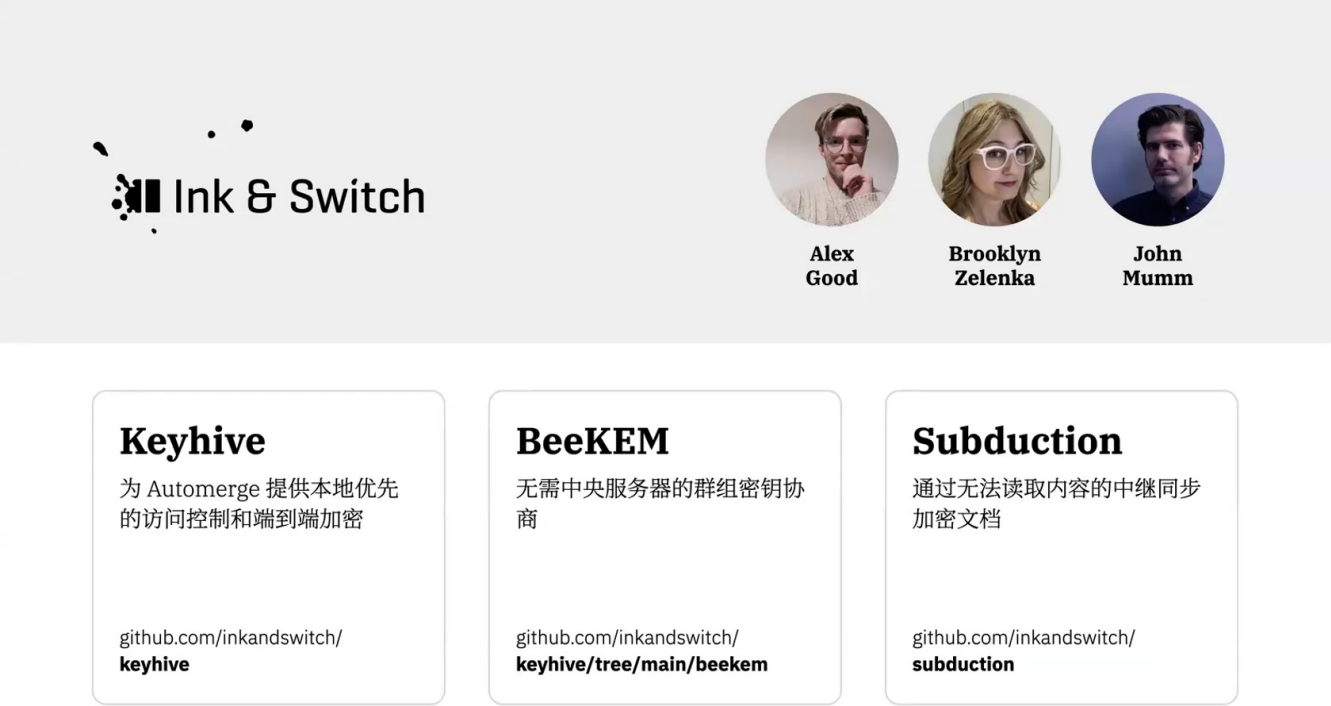
还有一个问题：没有中心服务器时，设备如何找到彼此并进行同步？是不是还需要信任中继服务？

01:07:52 • Herb Caudill

这有不同的实现方式。我的方案和 Keyhive 都不依赖特定的网络传输方式，因此可以运行在你使用的各种网络协议之上。比如你有一个可信的中继——其实，中继也不需要被信任，因为在确认对方身份之后，这些通信内容都是加密的。

01:08:38 • Herb Caudill

你也可以使用 iroh 这样的工具，完全通过点对点方式来实现。至于彼此怎样通信，有很多解决办法。一旦双方完成了身份验证，同步服务器就不需要被信任：它看不到自己存储或转发的内容。你放到它上面的所有东西都是加密的，而它没有任何密钥。



问答回看加密同步协议 • 画面 01:10:06

01:09:26 • Herb Caudill

这就是 Subduction：用于加密文档的服务器和同步协议，服务器无法访问或读取这些文档的内容。

01:09:58 • 主持人

嗯。

01:10:03 • Herb Caudill

Reticulum 的网络……嗯，我不熟悉 Reticulum，不过，是的。