Alice and Bob in wonderland: Bootstrapping identity and authority in a world without servers

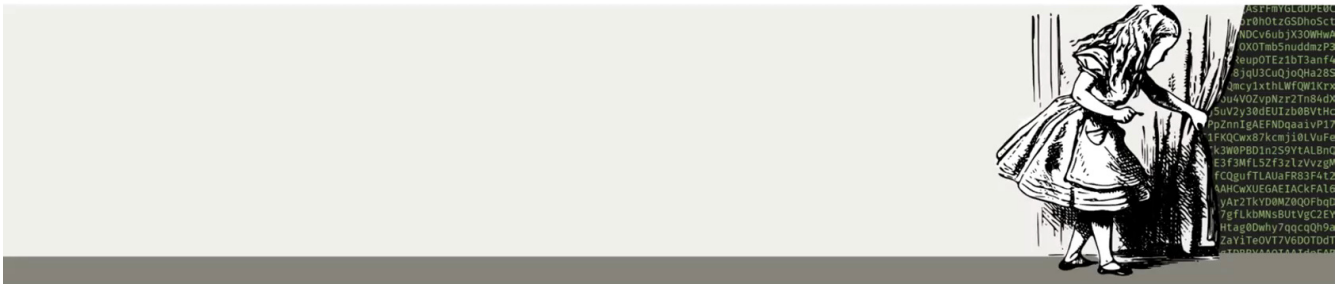
Herb Caudill · 2026-10-05 · English

English transcript (with translations of the Chinese questions). Edited transcript of the presentation and discussion on 5 October 2026. Repetitions and clear recognition errors have been corrected while preserving the order, examples, qualifications and questions. Product status and plans reflect the talk date. Timestamps refer to the edited video.

Without servers, what anchors identity and authority?

Alice 和 Bob 梦游仙境

在没有服务器的世界里，自举身份与权力



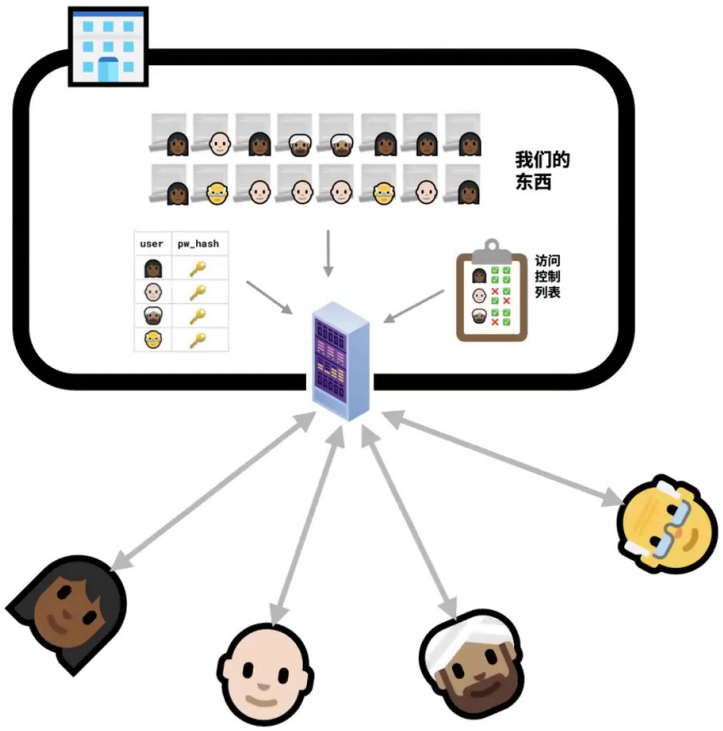
Alice and Bob in wonderland · Frame 00:00:01



Why identity feels disorienting · Frame 00:00:12

00:00:00 · Herb Caudill

My name is Herb Caudill, and I'm going to be talking about local-first authentication and authorization. When I first started thinking about this a few years ago, I really found it disorienting and weird. It might not be obvious what would be disorienting, so let me explain.



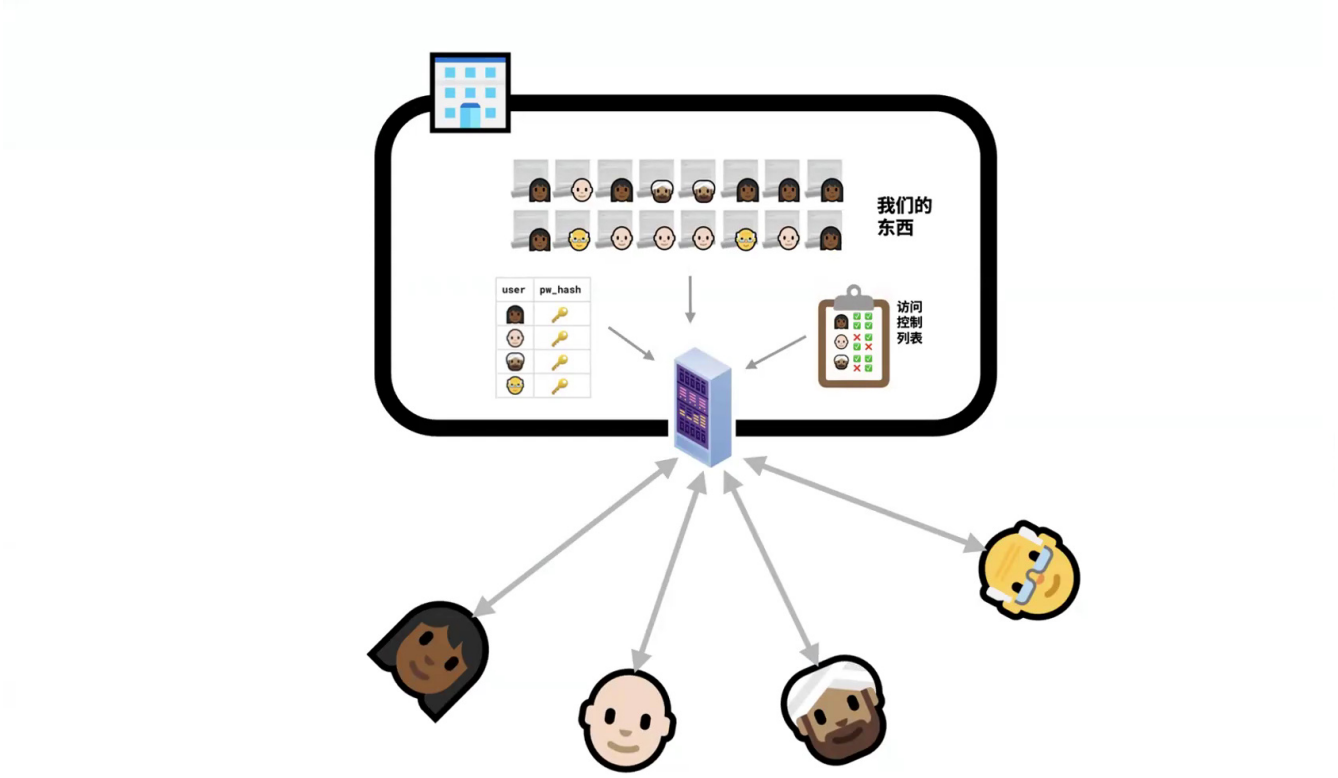
The server anchors identity and authority · Frame 00:00:59

00:00:24 · Herb Caudill

In the world that most of us are used to, authentication and authorization are ultimately anchored in physical hardware, right? That hardware is owned and controlled by a single company. You trust that company: it might be your employer, a software service provider, or a big tech company. That server has all your stuff on it, and it has rules about what you're allowed to do. It has your username and password. Anybody who wants to access their stuff has to go through that server, right?



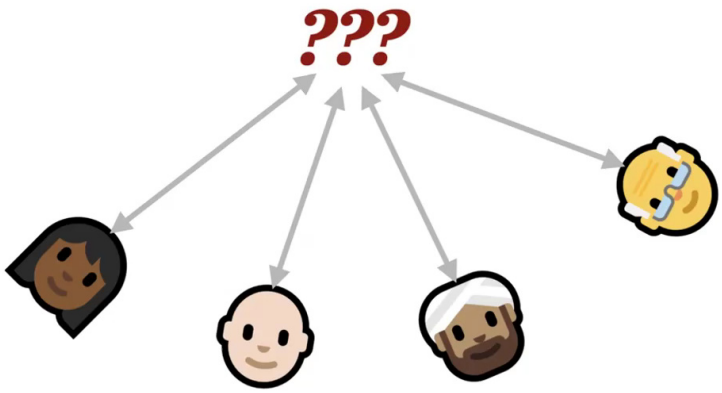
The guard at the castle gates · Frame 00:01:09



A central source of truth · Frame 00:01:36

00:01:03 · Herb Caudill

The server is like a guard at the castle gates. If it doesn't recognize you, it doesn't let you in. If it thinks you shouldn't be allowed to have something, then it just doesn't give it to you. Of course, we're talking about local-first. We would prefer not to be dependent on a central server. But the good thing about a server is that it does give you a single source of truth, and it is the physical custodian of your data. That gives you solid ground to stand on, right?



Identity without a server · Frame 00:02:15

00:01:43 · Herb Caudill

When the server goes away, you lose that solid ground. All you have are clients that relate to each other as equals. So how do you know who is who? How do you know who to trust? How do you know who has what permissions? What do permissions even mean in this context, where everybody has a copy of all the data? Who is giving permission? Without a server, we have to find new ways to solve problems that the centralized arrangement had already solved. That is what makes this disorienting, like going down the rabbit hole.



Down the rabbit hole · Frame 00:02:23



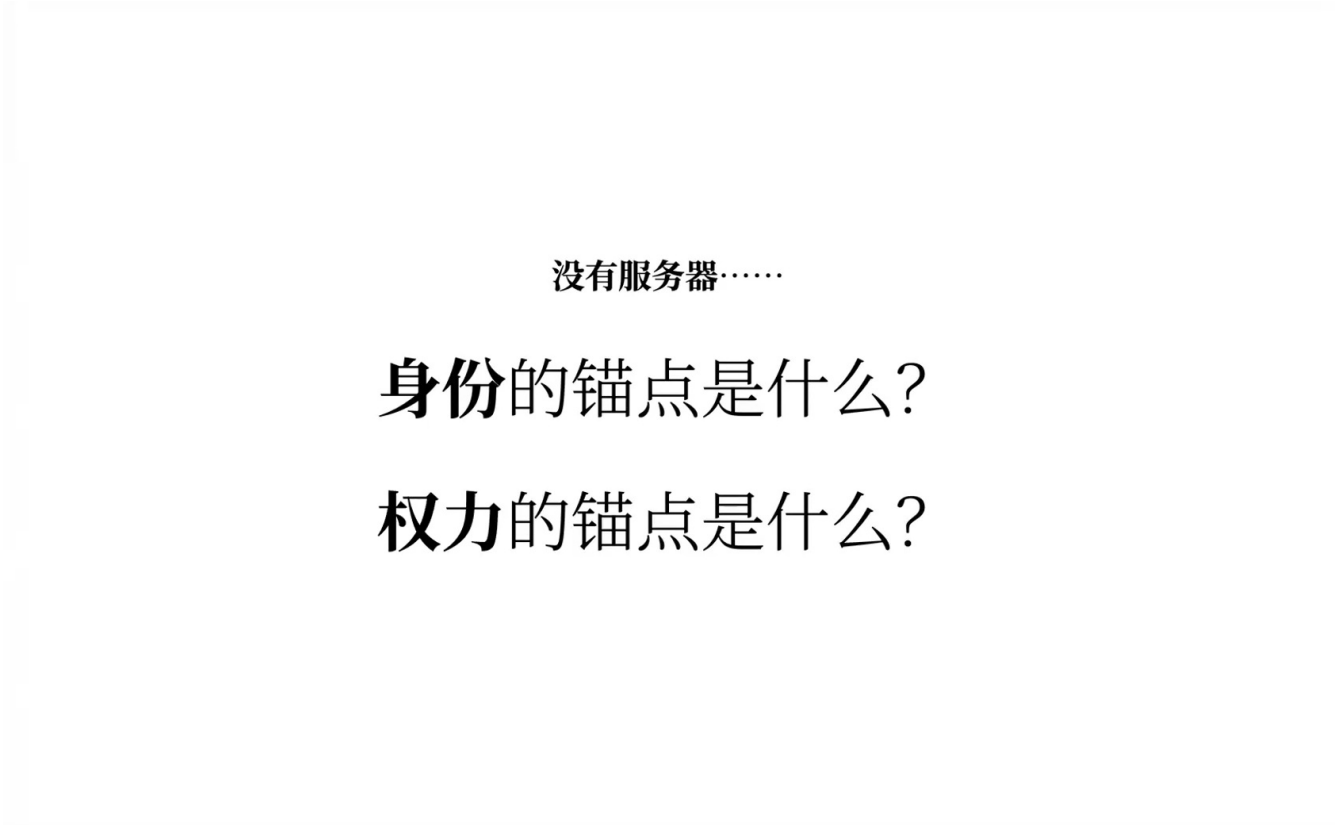
Bootstrapping · Frame 00:02:39



Lifting yourself by your bootstraps · Frame 00:03:06

00:02:30 · Herb Caudill

The English word “bootstrapping” has become a word that we associate with something as easy and routine as turning your computer on, right? Booting up your computer. But obviously, the whole point of the metaphor of bootstrapping is that it's supposed to be impossible. You can't lift yourself up by your own bootstraps. It's not possible. It can't be done. But we kind of have to figure out a way to bootstrap identity and permissions in this context.

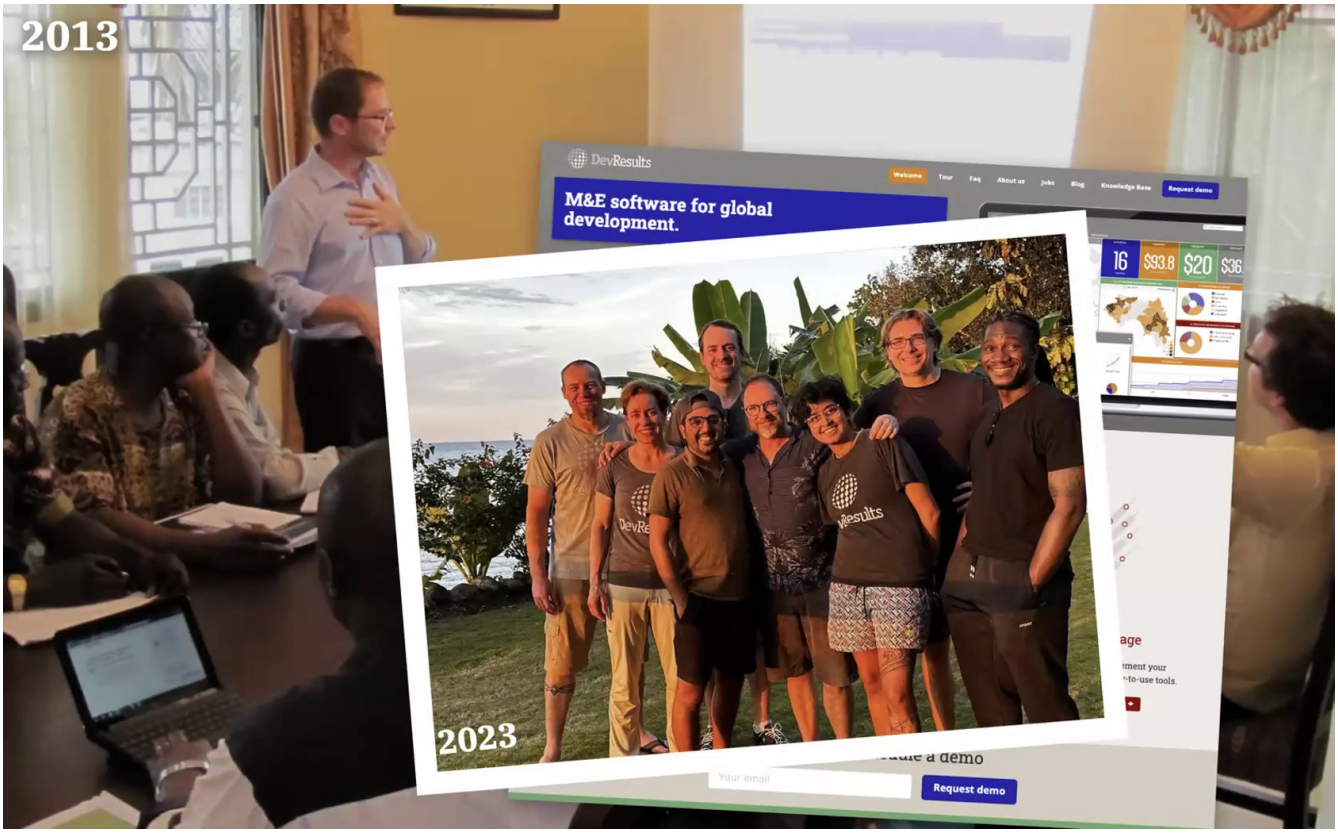


The two questions · Frame 00:03:31

00:03:14 · Herb Caudill

The first question without a server is: what anchors identity? How do we know who we're collaborating with? The other is: what anchors authority? How do permissions work? Who has the authority to give anyone permission in the first place?

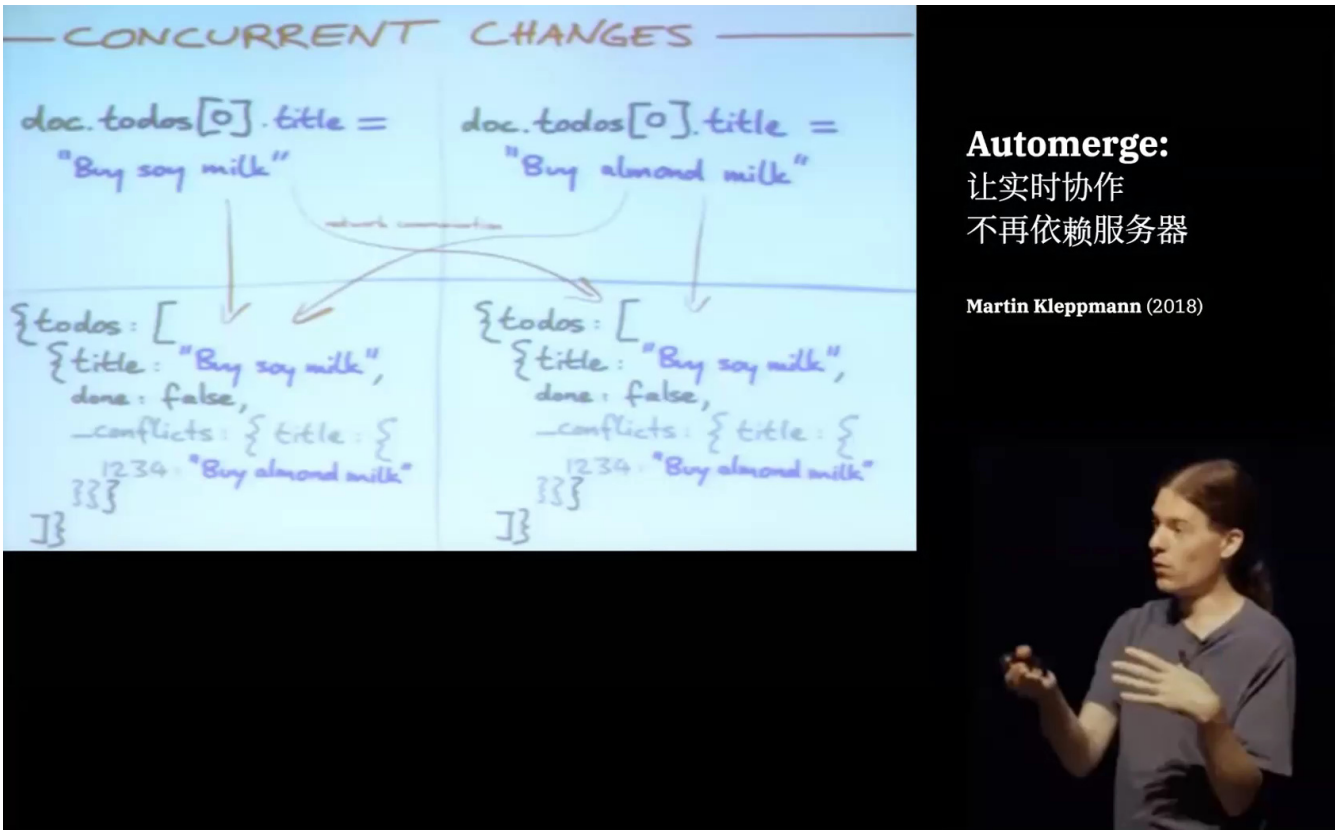
From DevResults to local-first



DevResults and the project background · Frame 00:04:11

00:03:35 · Herb Caudill

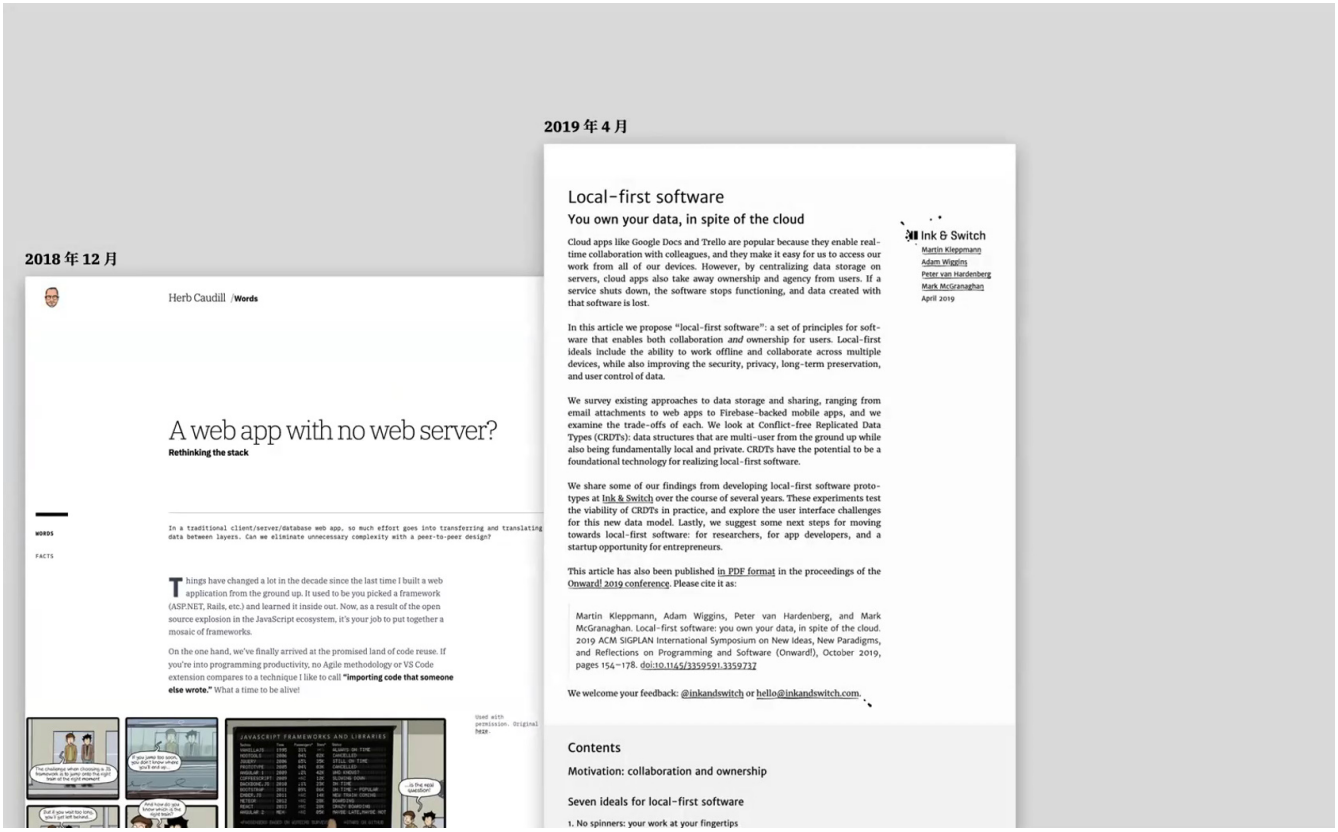
Just some background on my own motivation. About 15 years ago, I started building a software product called DevResults that's used by government aid agencies and international NGOs. This is me, for example, a long time ago, training a team in Sierra Leone in West Africa. The company has been successful in a small way. We have a small team of very competent people, and I've been starting to think about what we build next.



Automerge and independence from servers · Frame 00:04:31

00:04:11 · Herb Caudill

As soon as I started experimenting with building a new version of this, I was frustrated by some fundamental things about the way software was made. Around that time—this is 2018—I stumbled across this video, which introduced me to Automerge. That was the beginning of my journey into Wonderland.



Early local-first work · Frame 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

What developers and users need · Frame 00:06:11

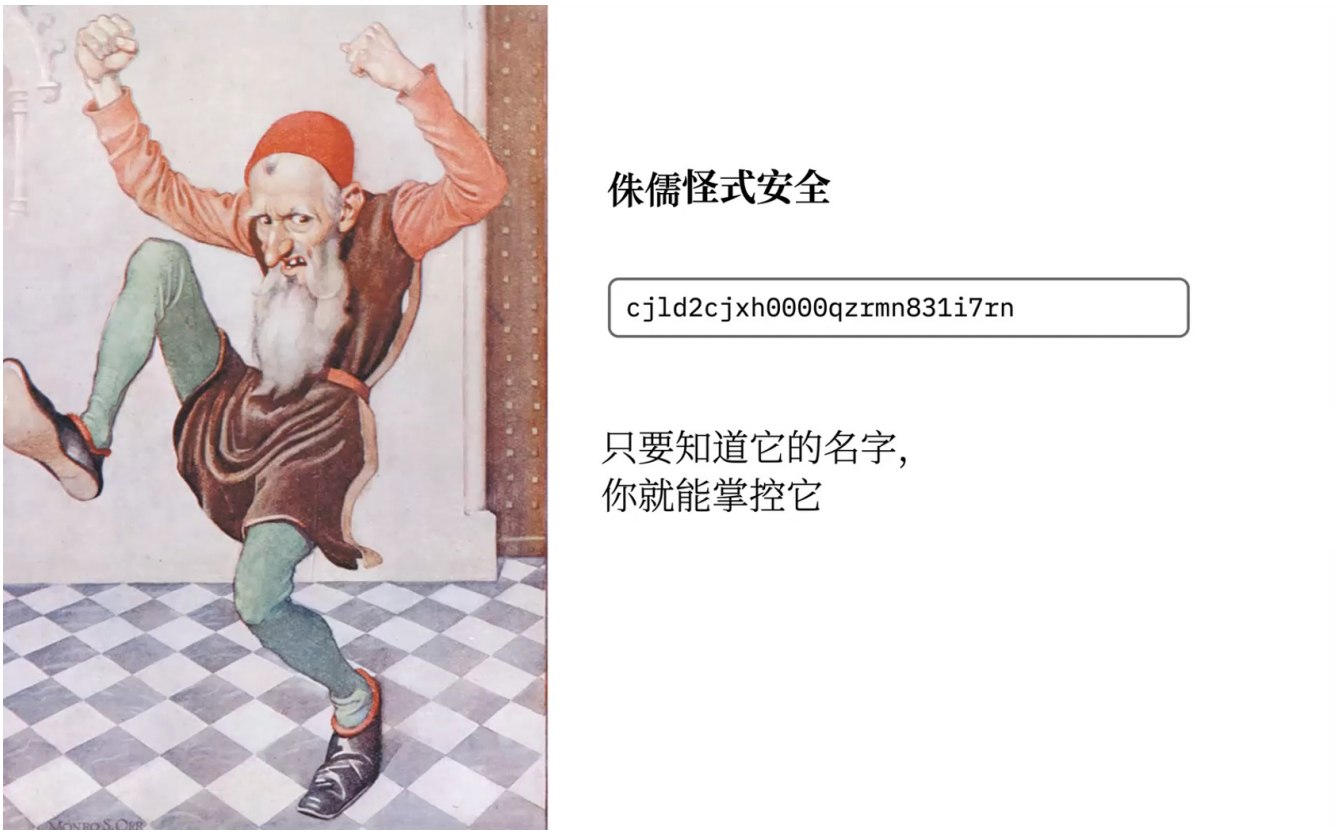
00:04:40 · Herb Caudill

Shortly after that, I wrote this post. I just want to note for the record that this came out five months before the Ink & Switch local-first manifesto. So I've been thinking about this for a long time. There are a couple of things that appeal to me about local-first. It appeals to me both as a maker of software and as someone who uses software. As a developer, I like the simplicity. All you have to write is the client, and that means that you can iterate faster.

00:05:21 · Herb Caudill

Really, what I want to do is make software. I don't really want to be the custodian of anybody's data, right? If I don't have to host your data, then that also makes things simpler for me. As someone who uses software, I want ownership of my stuff. The right to access the documents I create should never be in question. I shouldn't have to be online to access my stuff. Of course, for my customers at DevResults, who are in Africa and Latin America and other places with intermittent internet access, it's important for them to have offline access as well.

Document IDs and revocation



Security by a secret document ID · Frame 00:06:56

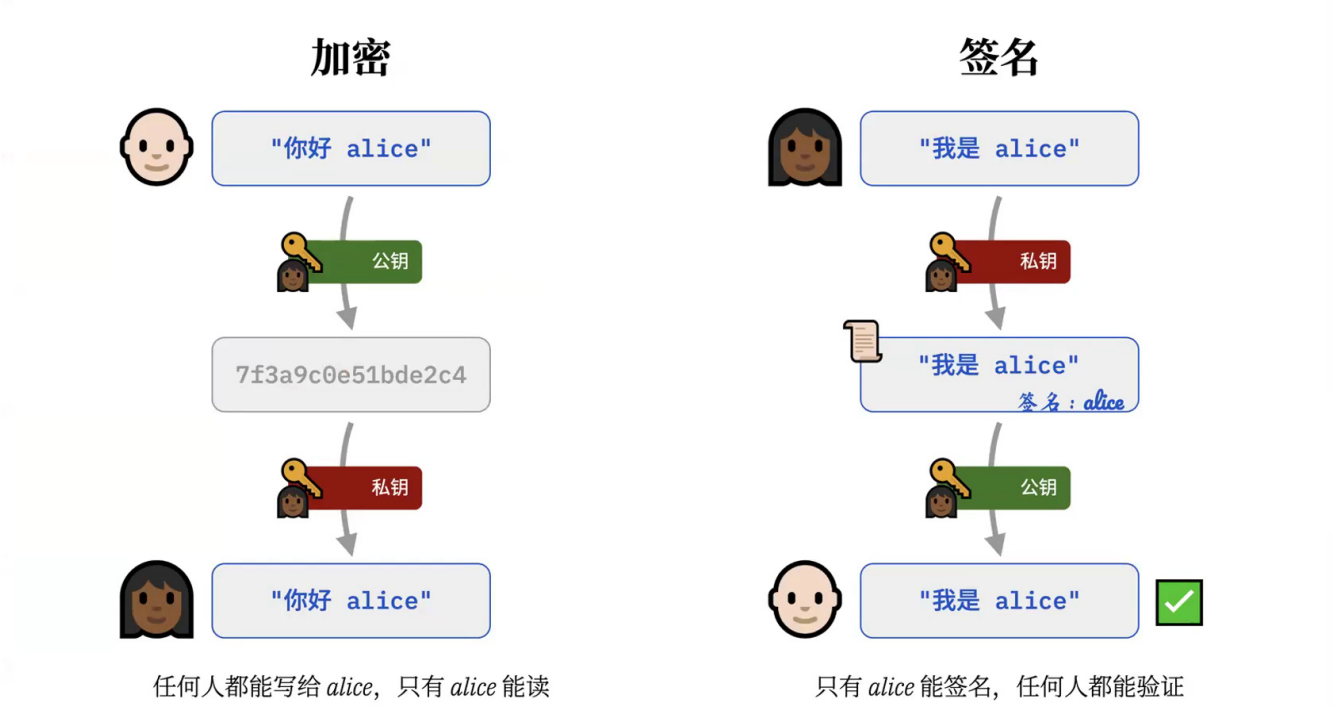
00:06:12 · Herb Caudill

With my team, we started experimenting on building local-first tools to use internally, but immediately we ran into the problem of how to secure them. The first solution that we ended up with might be familiar to some of you: we just took the document ID of the root document and used that as an access token. It's a long random ID. It's hard to guess. The idea is that just by knowing the name of the document, you have access to it, right? That's good enough for some use cases.

00:06:49 · Herb Caudill

But of course, for a team, if somebody leaves the team, or if somebody's device is compromised, or they lose their device, what do you do then? You can't just change the ID of the document. You kind of have to start over. There's not a lot of good options. That's definitely not going to be good enough for a commercial product, right?

Public-key cryptography and key management



Encryption and signatures · Frame 00:07:28

00:07:14 · Herb Caudill

In 2020, a couple of weeks into the pandemic, I started working on a library called @localfirst/auth. To be clear, I'm not a cryptographer, and I had to learn about modern cryptography from scratch. I was immediately fascinated by how empowering public-key cryptography is, in the sense that it can put me on equal footing with anybody in the world, no matter how powerful they are.



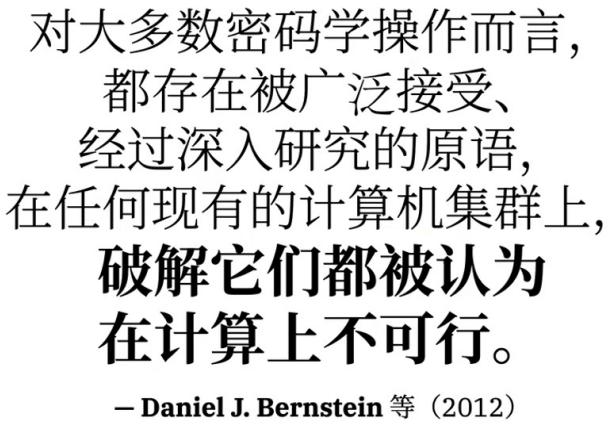
Password cracking in the movies · Frame 00:07:51



Fiction versus real cryptography · Frame 00:08:13

00:07:45 · Herb Caudill

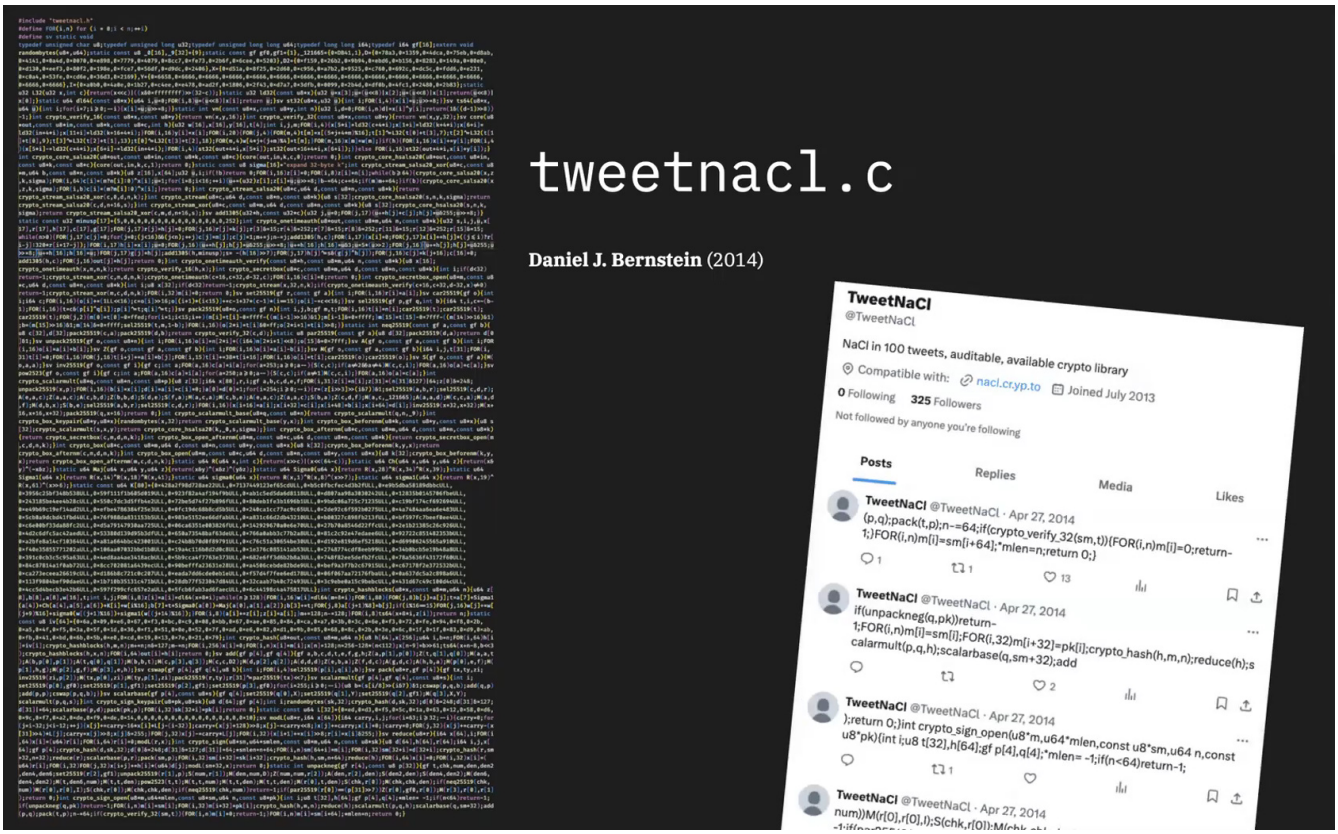
In the movies, defeating encryption is just something that you can do by typing very fast. You can decrypt anything. But the reality is more like this. If you do it right, it can't be done. There's not a way to decrypt something if it's been encrypted properly.



The foundation of modern cryptography · Frame 00:08:31

00:08:20 · Herb Caudill

It doesn't matter how powerful or well-funded your adversary is. The mathematics of it just means that it's not feasible for anybody, with any amount of resources, to decrypt a message without the right key or to fake a digital signature. That's really empowering. It doesn't require any specialized equipment. It's just math that you and I can do on a cheap laptop, and it doesn't even require that much code.



TweetNaCl · Frame 00:09:22

00:08:55 · Herb Caudill

This is the entire codebase for a compact C implementation of the NaCl cryptography library that was designed to fit in 100 tweets, right? So less than 25,000 characters. That's amazing. That's not very much code.



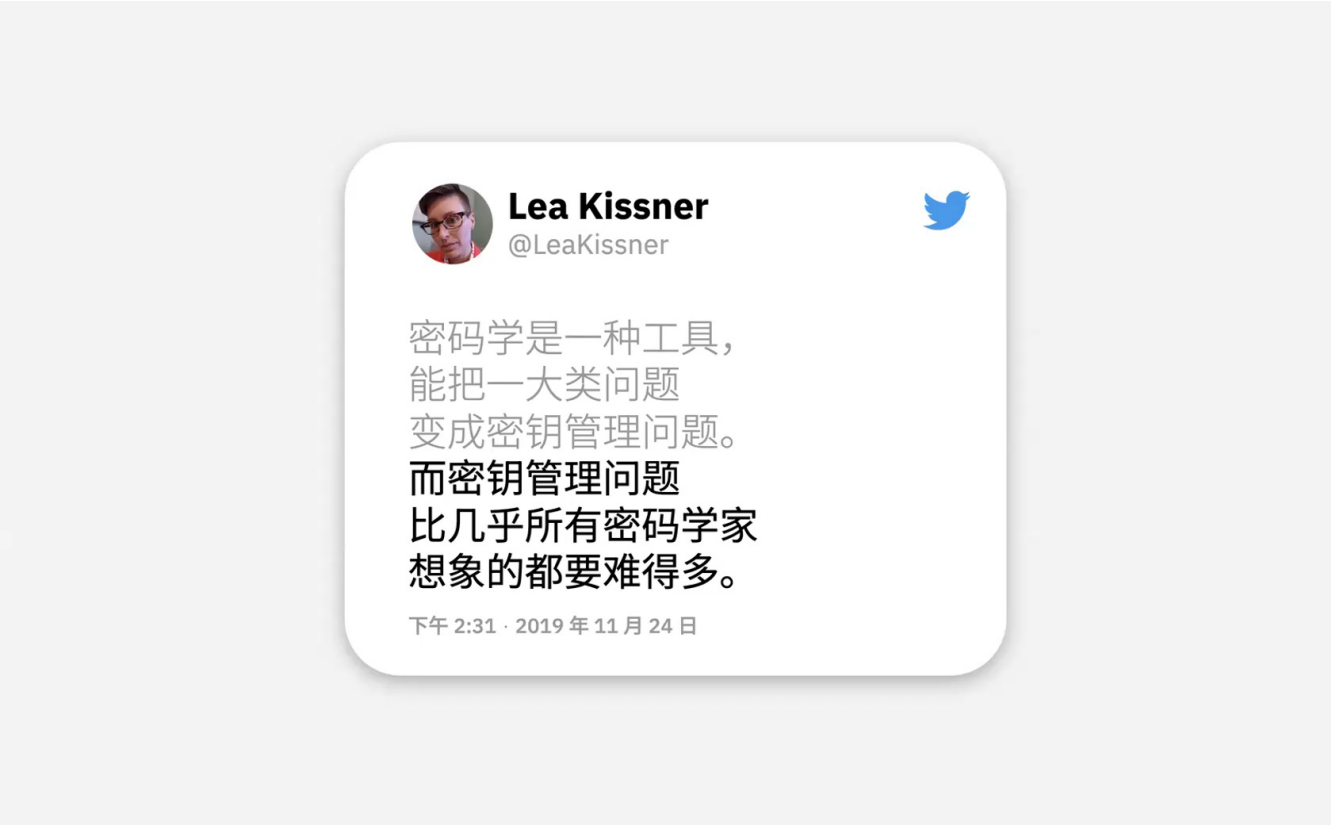
任何人都能写给 *alice*，只有 *alice* 能读

只有 *alice* 能签名，任何人都能验证

Encryption and signatures · Frame 00:09:36

00:09:23 · Herb Caudill

It was clear to me that public-key cryptography had to be part of the solution to local-first auth. But the first problem, in both of these scenarios, is that Bob has to know Alice's public key. How do we make sure that Bob knows Alice's key in a decentralized context? That brings us to this first question: what anchors identity without a central server?



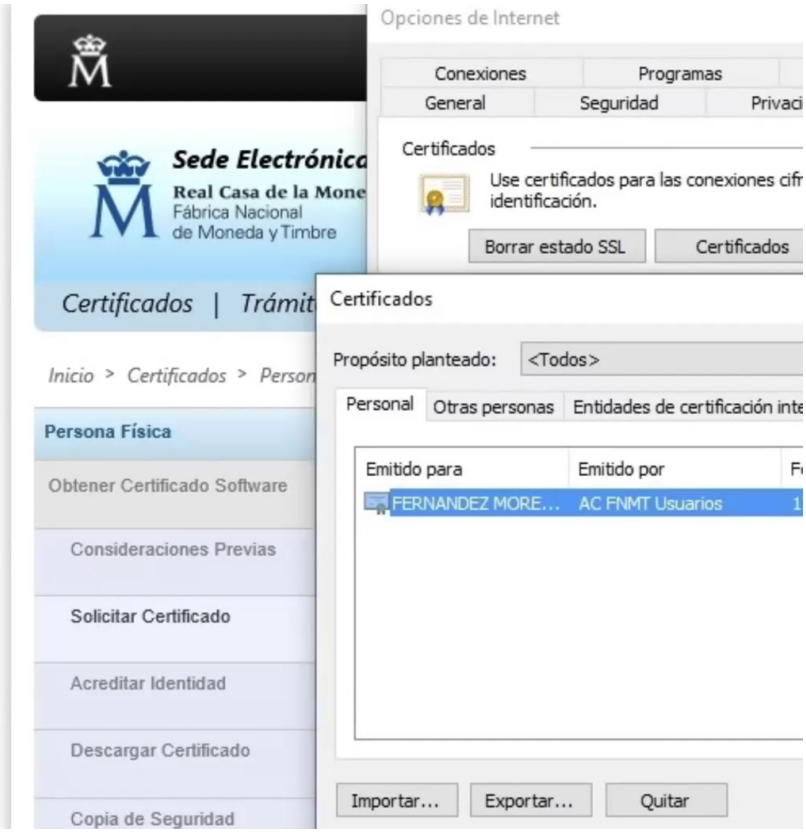
Cryptography becomes key management · Frame 00:10:31

00:10:03 · Herb Caudill

This is a famous quote: “Crypto is a tool for turning a whole swath of problems into key management problems.” This quote is from Dr. Lea Kissner, who was the head of information security at Twitter at the time. They didn't really mean this tweet to be encouraging, right? The rest of it says, “Key management problems are harder than most people think.” But I found this idea very encouraging because, yes, it's a hard problem, but I would rather have one hard problem than many, many problems, as long as I can find a solution to that one hard problem.

PKI

public
key
infrastructure
公钥基础设施



Public key infrastructure · Frame 00:11:41

00:10:45 · Herb Caudill

How do we solve the key management problem? How do we learn what people's public keys are? The most obvious solution is to have a trusted party keep a directory of people and their public keys. This is called PKI, which stands for public key infrastructure. The solutions to this generally involve centralized servers. You have the certificate authorities that make HTTPS connections possible. Some big companies have their own certificate authorities to anchor their employees' digital identities. Countries can do the same for residents.

00:11:28 · Herb Caudill

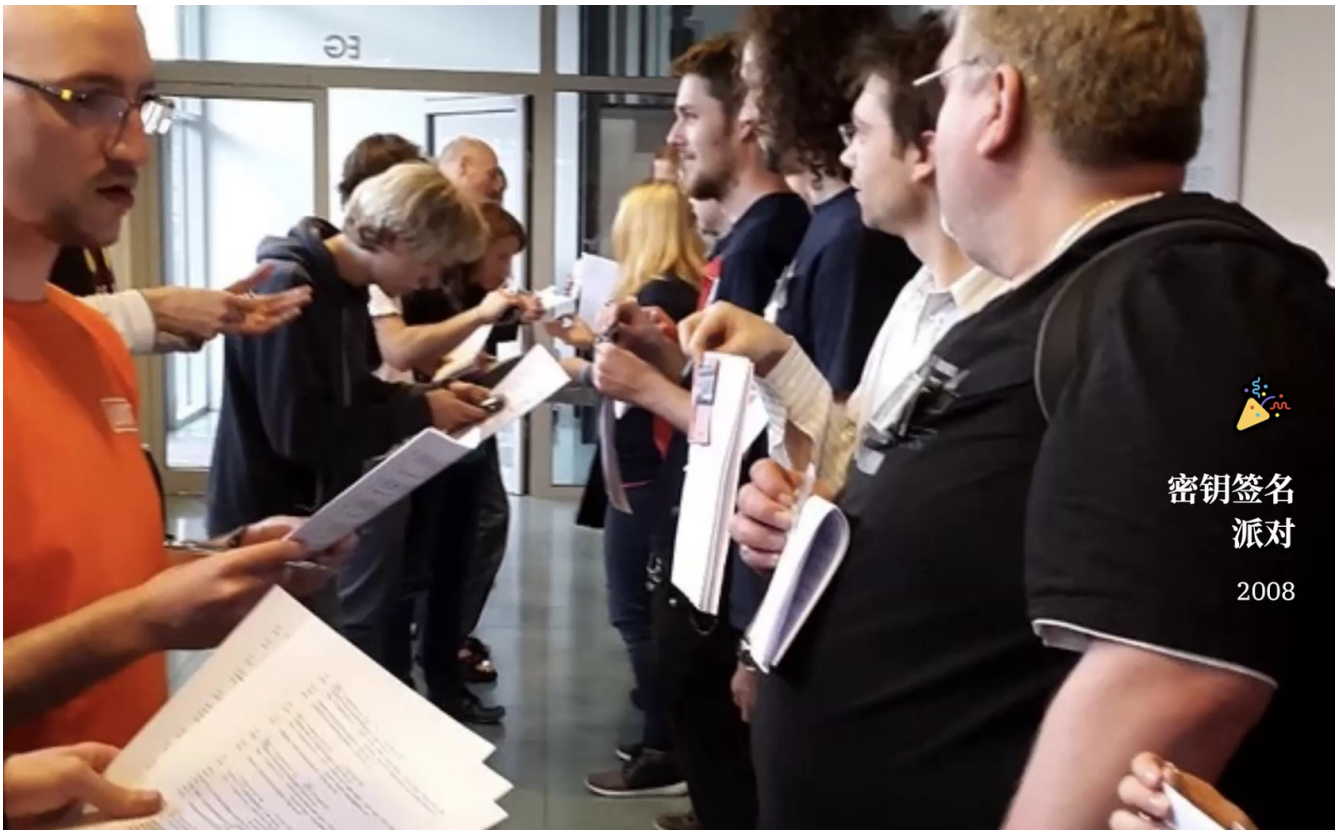
I live in Spain, and in Spain I have a certificate that's given to me by the government. I install that certificate in my browser, and then I can use that to log into government websites, sign official documents, pay my taxes, and those kinds of things. But how do you do this without a centralized server? How do you do this in a peer-to-peer context or in a decentralized context?



PGP public keys · Frame 00:12:23

00:11:58 · Herb Caudill

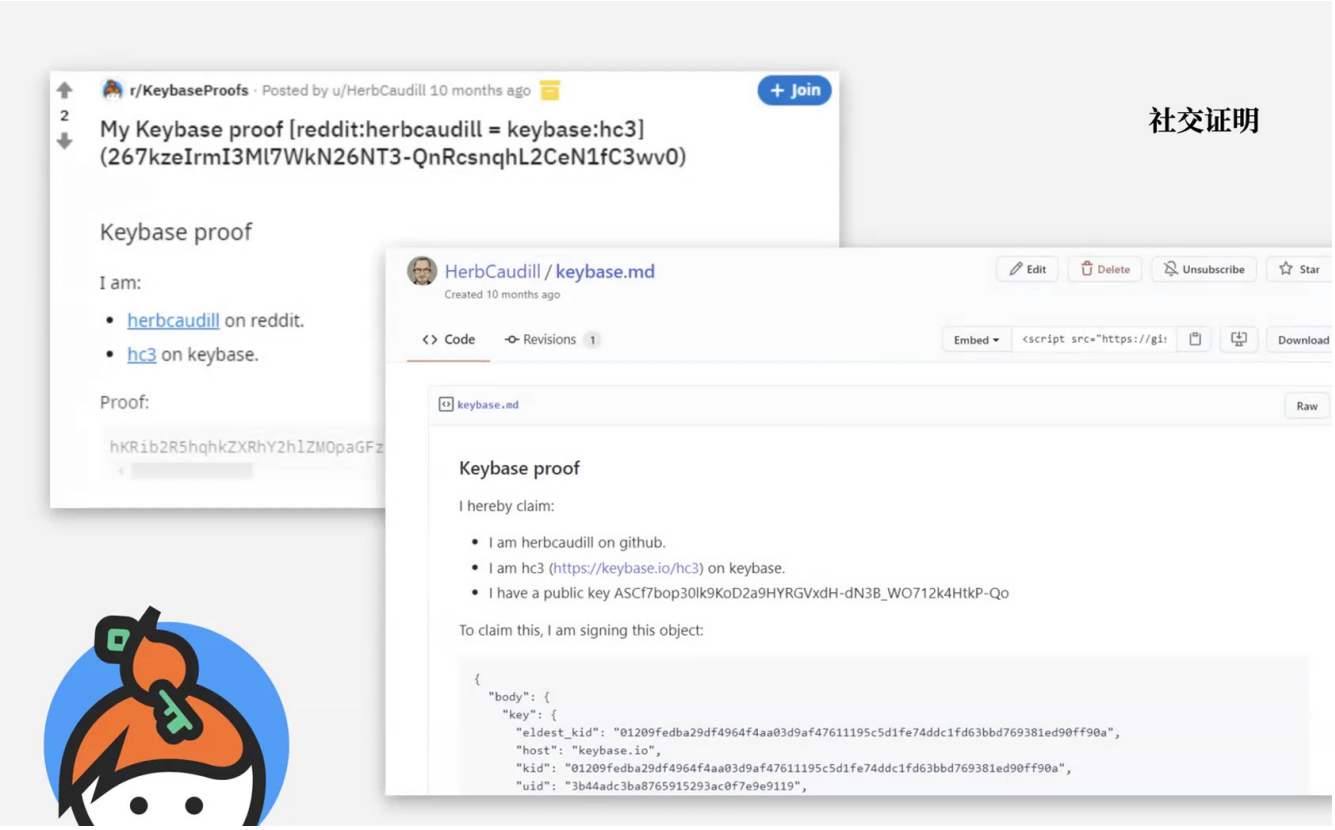
The earliest solution to this was something called PGP, which stands for Pretty Good Privacy. For example, this is my PGP public key. The idea here was that you would just share your public key with other people directly. People would put their public keys on their websites, or they would put their public key in their email signature. But of course, this is a big, scary thing.



Key signing parties · Frame 00:12:53

00:12:28 · Herb Caudill

They had what they called key-signing parties, where people would line up, show each other their ID, and sign to verify that they had seen that. The idea was that this would build up a web of trust. But obviously, only nerds did this. This is not something that was popular with normal people, and so it really never became a thing.



Keybase social proofs · Frame 00:13:51

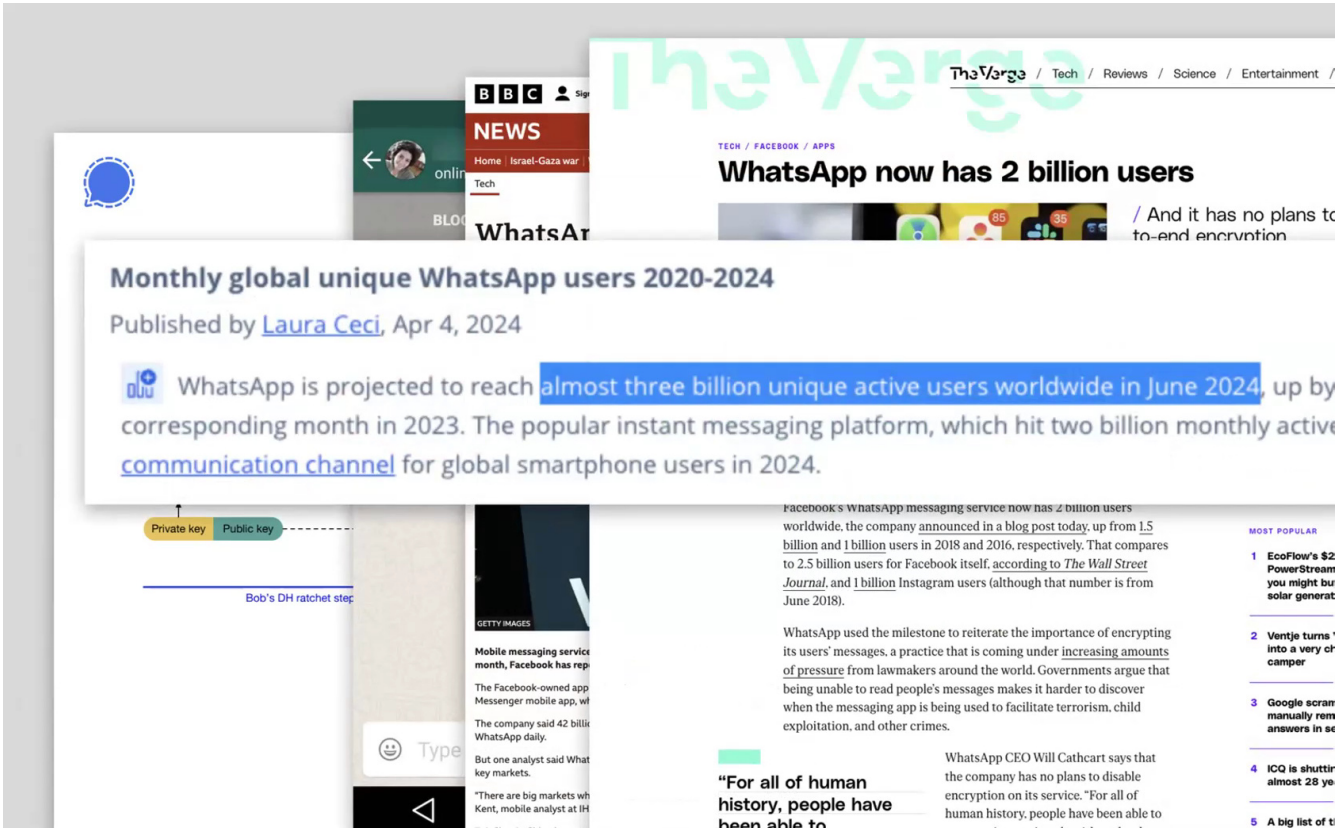
00:13:07 · Herb Caudill

There was a company called Keybase a few years back, which was actually bought by Zoom in 2020. Keybase had a very cool idea called social proofs. The idea was that you would post a cryptographic proof on a site like Twitter, GitHub, or Reddit, and that linked your public key to your identity on those sites. You could then sign something with your key. There was an application that they made that would then verify that that was your key by linking it back to your GitHub account or whatever.

00:13:55 · Herb Caudill

Of course, this is easier than a key-signing party, but it was still something that only nerds did, right? It was not something that the general population found useful. We'll come back to Keybase, though, in a minute.

Signal, WhatsApp and device keys

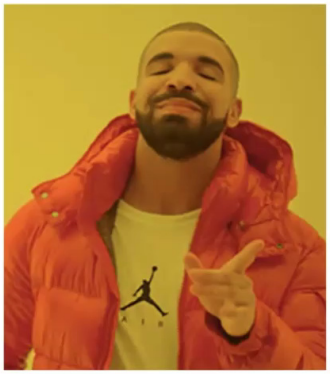


Signal and WhatsApp · Frame 00:15:01

00:14:11 · Herb Caudill

What really made the difference here, and brought end-to-end encryption to the masses, was the Signal protocol. First you had the Signal app. It was a little bit nerdy. It was still only used by people who were really interested in encryption. But then WhatsApp, from one day to the next in 2016, brought end-to-end encryption to their entire customer base, which was about a billion people at the time. Now there are over 3 billion people using WhatsApp on a regular basis. It's one of the most widely used communication tools in the history of humanity.





PGP

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**

From personal identity to application identity · Frame 00:16:16

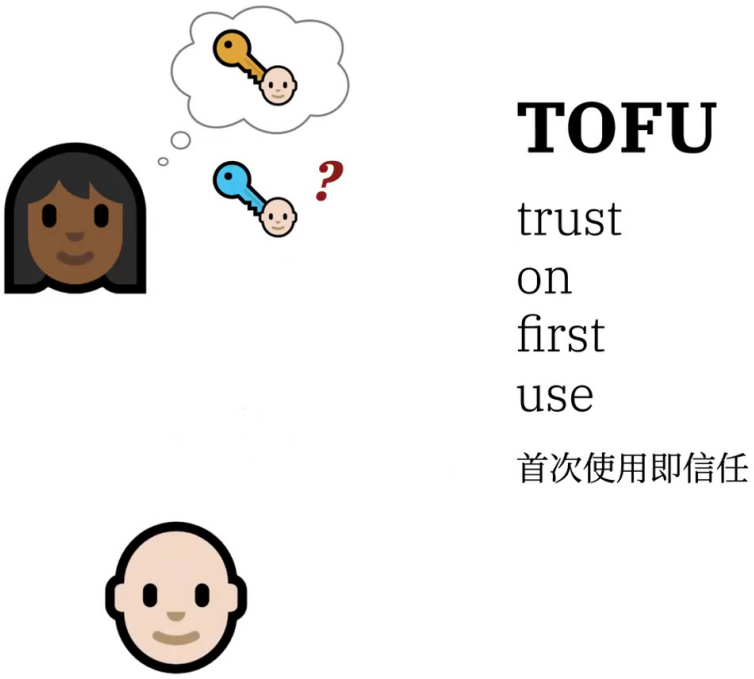
00:15:01 · Herb Caudill

What is it that WhatsApp and Signal figured out that we hadn't figured out with Keybase or with PGP? They did it by making encryption transparent to the user. This required a change in perspective. Rather than have a single key that belongs to me as a person, I have a key for every app on every device. Rather than managing the key myself—because people are just bad at managing keys; we can't even deal with passwords properly, let alone PGP keys—we let computers do it. Computers are good at that kind of thing.

00:15:50 · Herb Caudill

In this model, the device generates keys. The user never sees them. Peter van Hardenberg, who's the head of Ink & Switch, is fond of saying that if a user sees an encryption key, you've already lost. So you just keep that internal to the computer. It's an implementation thing that the user doesn't need to know about. People don't share keys with each other. Devices just share them with each other automatically. This is a much better perspective on how to manage keys.

The limits of trust on first use



TOFU and changed keys · Frame 00:18:01

00:16:30 · Herb Caudill

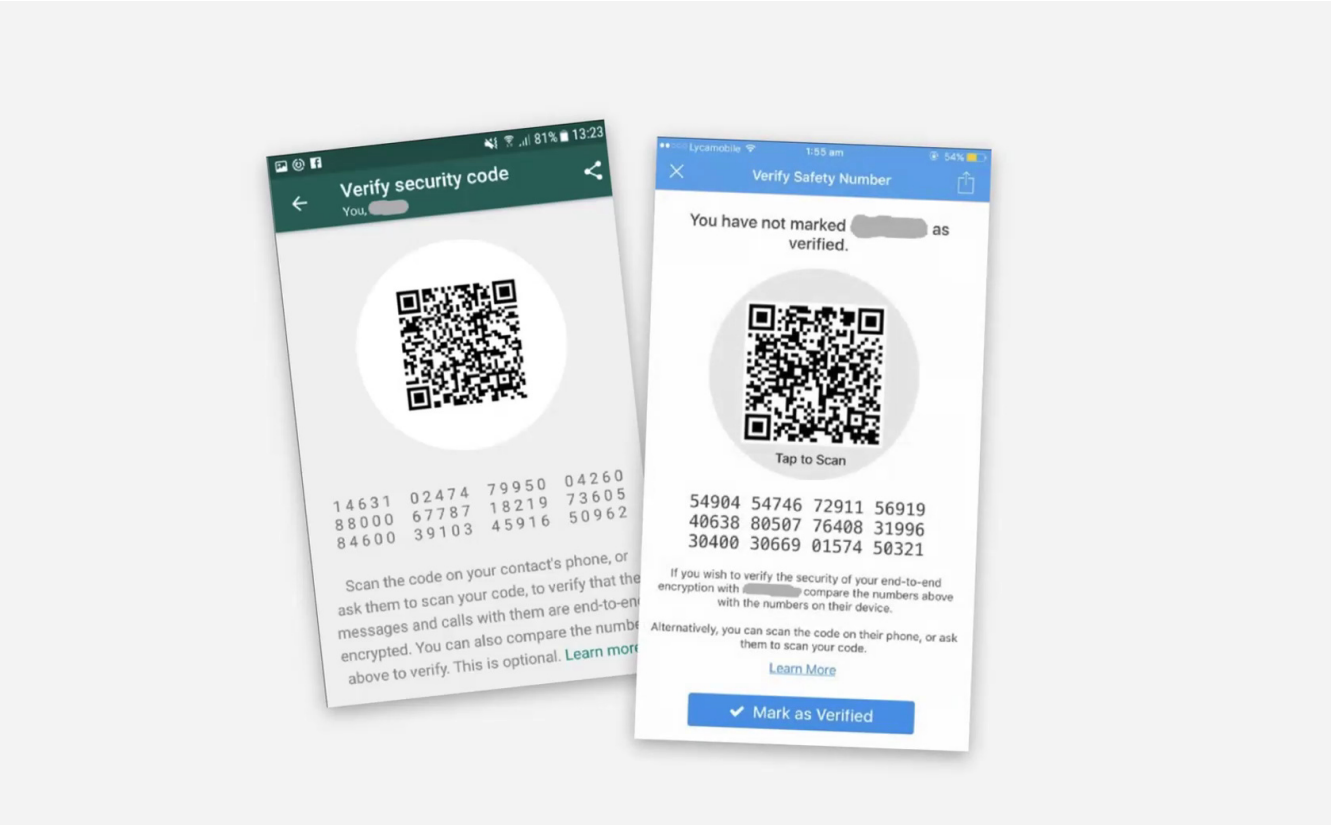
But you still have the question of how Alice's device learns the public key of Bob's device. WhatsApp and Signal and others use an approach called TOFU, which stands for Trust On First Use. The idea is that Alice's device trusts that the device it first connects with is Bob's device. Alice's device sends its public key to Bob's device. Bob's device sends its public key back, and each device just remembers the other's key. From that point on, they can use those keys to authenticate and to encrypt messages to each other.

00:17:21 · Herb Caudill

That seems reasonable, but “device” doesn't really capture the reality of what's going on here. For example, if you open the app in a different browser, like Firefox instead of Chrome, that's a different device. If you reset your phone, get a new phone, or reformat your computer, that's a different device. If Bob reinstalls the application, it can lose its keys, and so on.

00:17:50 · Herb Caudill

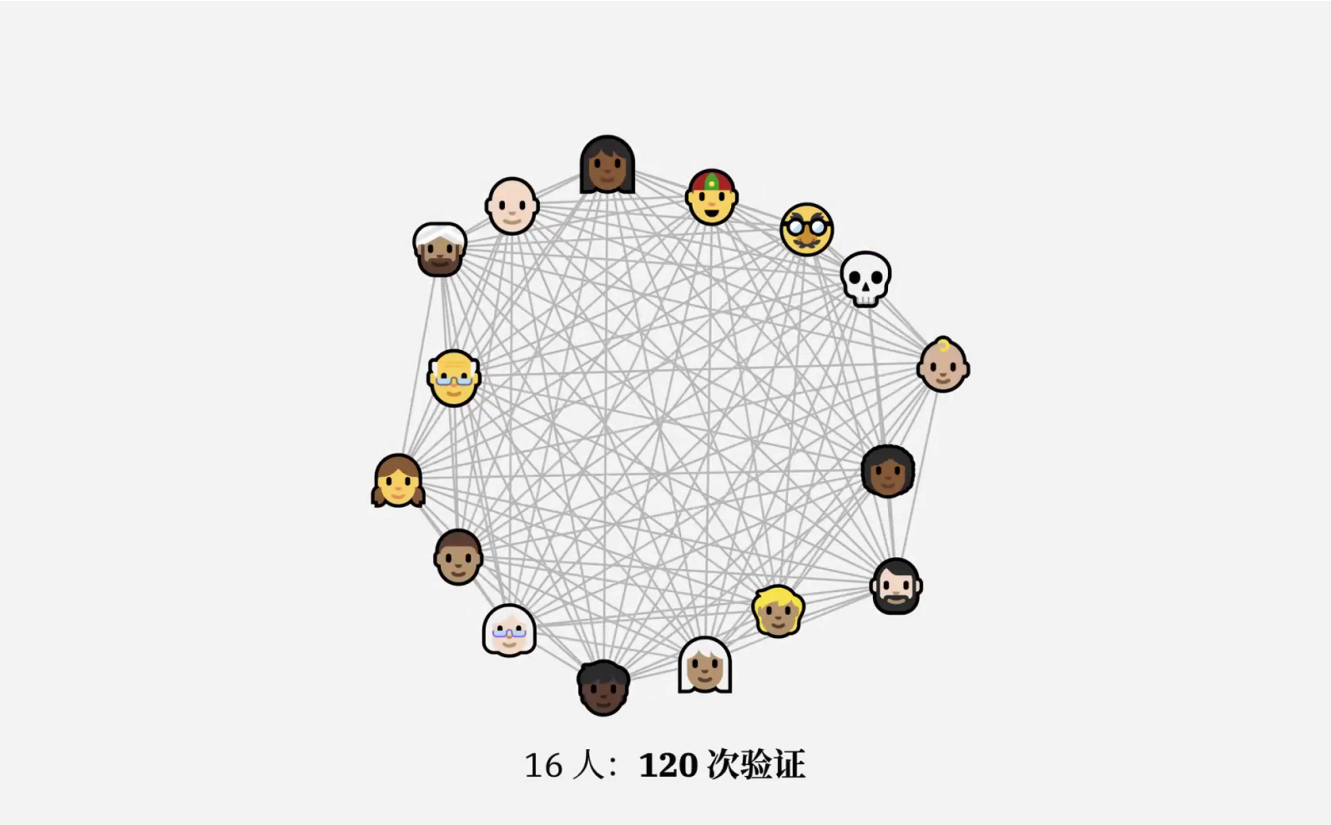
Suppose that Bob reinstalls the application. He now sends a new public key to Alice, and Alice has no real way to tell from that key alone whether she's actually still speaking to Bob. So she has to trust this connection as well. It's not really trust on first use. It's trust on first use, plus every time a device gets compromised or you use a new browser, and so on.



Comparing QR codes out of band · Frame 00:18:51

00:18:26 · Herb Caudill

The way that WhatsApp and Signal handle that is that they ask users to compare security codes or safety numbers. This is basically a digital version of the PGP key-signing party thing. Most users don't do this. I've never done it, and I'm a nerd. So if I'm not doing it, then I don't think really anybody is doing it.

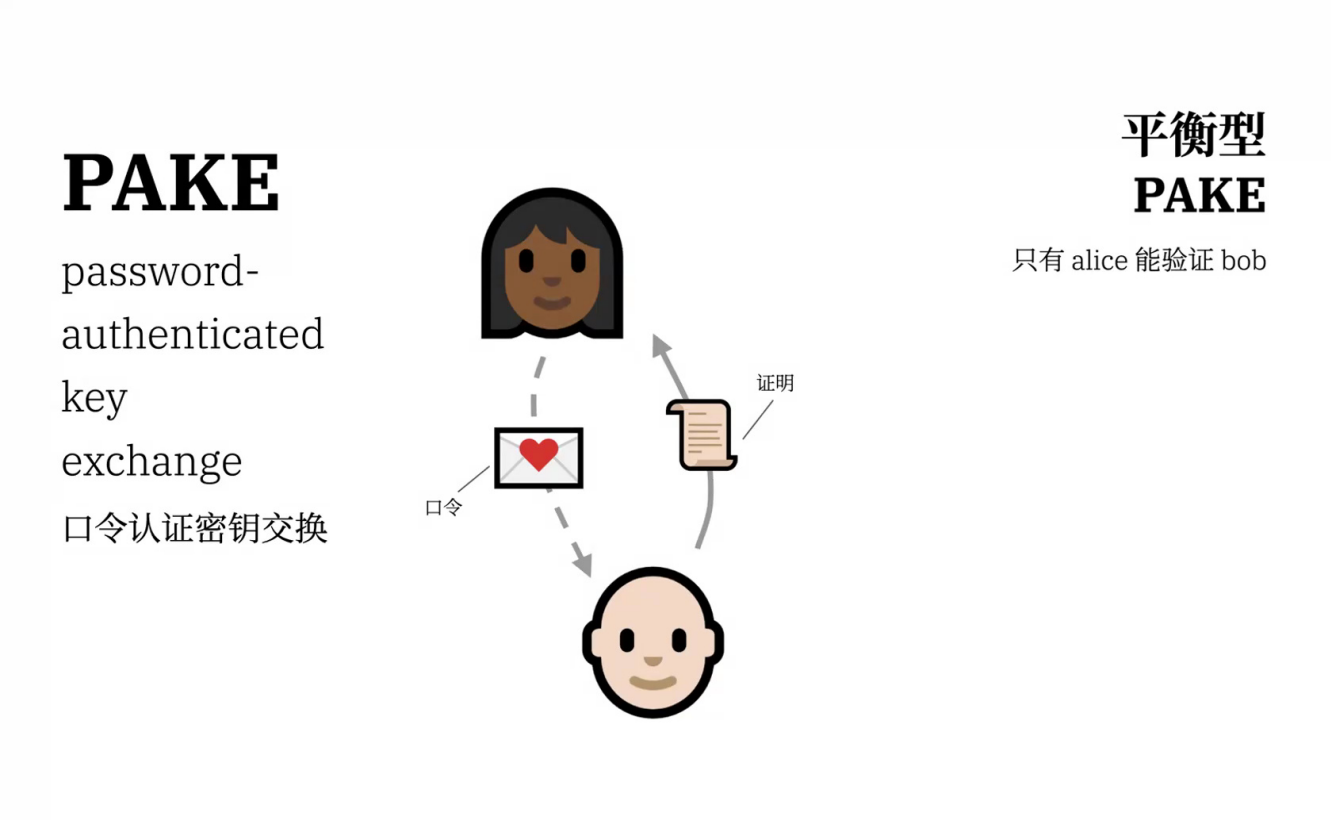


The cost of pairwise verification · Frame 00:19:23

00:18:54 · Herb Caudill

The real problem with this approach is that it doesn't scale to groups. It's one thing to do this comparison between two people, because you just have one verification to do. But if you have four people, you need six comparisons. If you have 16 people, you need 120 comparisons. It just becomes completely unmanageable once you start to deal with a group of any size.

PAKE and the Seitan token exchange



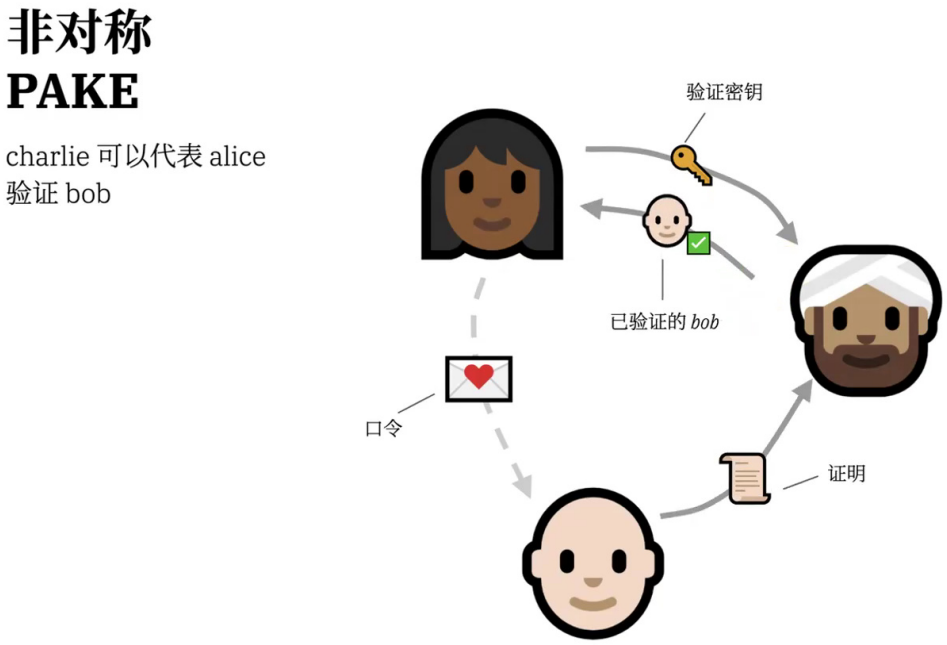
Password-authenticated key exchange · Frame 00:20:51

00:19:29 · Herb Caudill

If Alice wants to connect with Bob and doesn't know his public key, is there an alternative to trust on first use for securing that first connection? One way is for Alice to share a single-use password out of band. That can mean in person, by email, or through some other channel that she already uses to communicate with Bob. Bob's device then connects and gives Alice's device a cryptographic proof that it knows the password. This is a family of protocols called PAKE: Password-Authenticated Key Exchange.

00:20:17 · Herb Caudill

The idea here is that you have what we're calling a password. It's usually presented as an invitation code. In this example, this is what's called a balanced PAKE, in that Alice can verify Bob because she also knows the password. The disadvantage is that the inviter and the invitee need to connect directly to do this exchange. But what if you want to invite someone to a team and let another team member admit them later? How would that work?



Asymmetric PAKE and delegated verification · Frame 00:21:44

00:21:04 · Herb Caudill

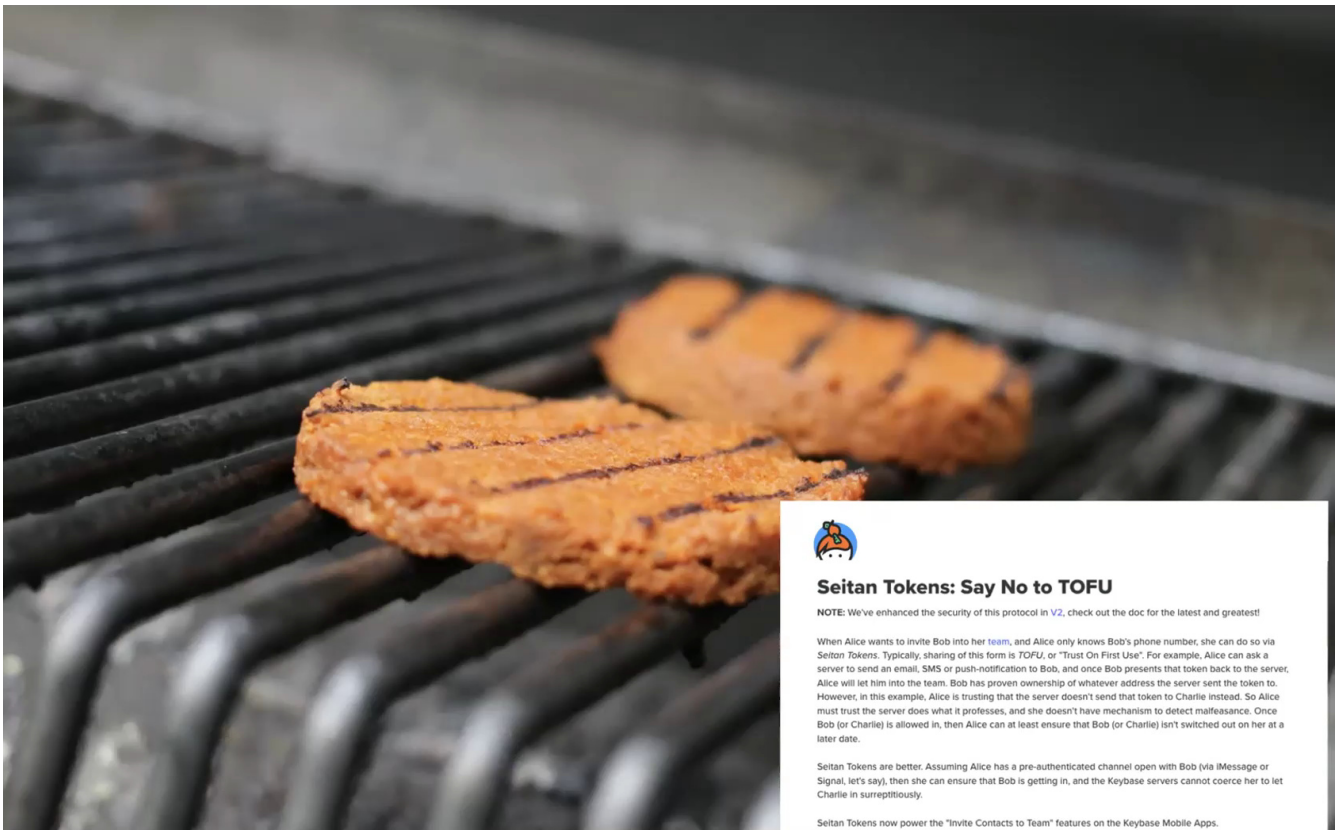
The alternative is what we call an asymmetric PAKE. It uses some additional cryptographic machinery to make it possible for a third party to do the validation. Alice gives Bob the password or the invitation code, and she gives Charlie some material that's derived from the password, not the password itself. Bob then uses the password to generate a proof and sends it to Charlie. Charlie validates the proof and confirms Bob's identity to Alice. Charlie can do this without knowing the password himself. This is a neat trick.



Keybase team messaging · Frame 00:22:16

00:21:55 · Herb Caudill

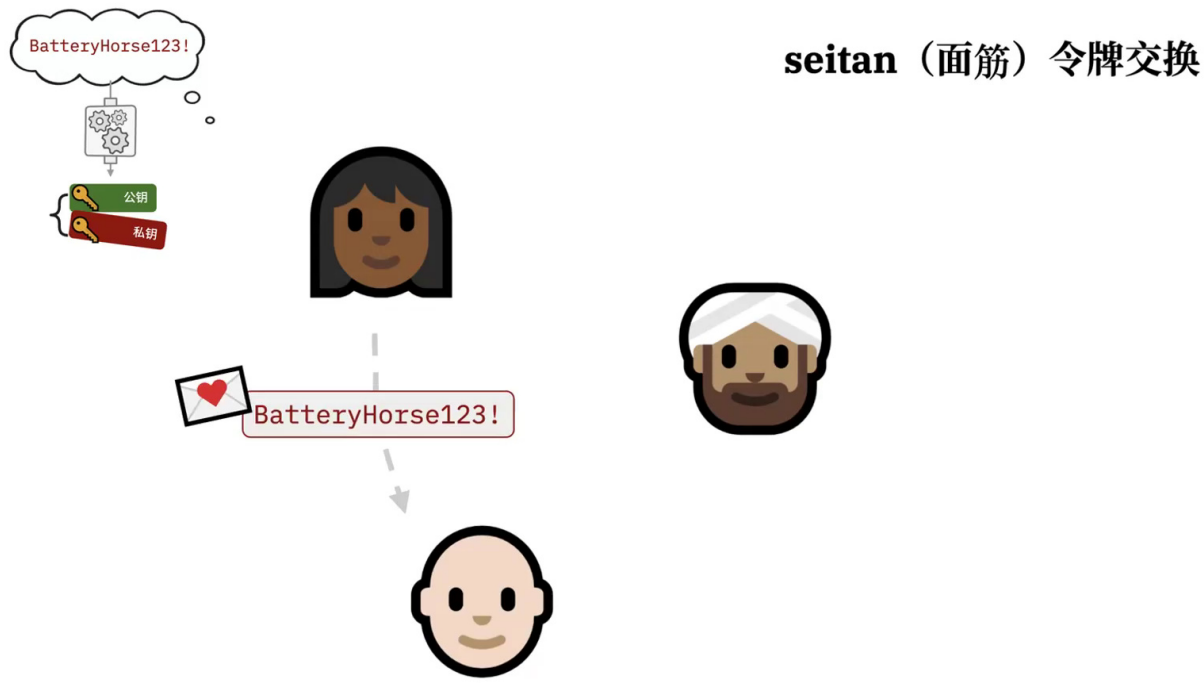
Going back to Keybase: Keybase had built this system for doing social proofs. The way they originally planned to monetize their work was to build an encrypted app for team chat, similar to Slack. It would be used by a company, or a team within a company, to communicate. They realized that they weren't going to get everybody in a company to put their public keys on Twitter or GitHub. They needed another way to bootstrap those key exchanges.



Seitan instead of TOFU · Frame 00:22:56

00:22:36 · Herb Caudill

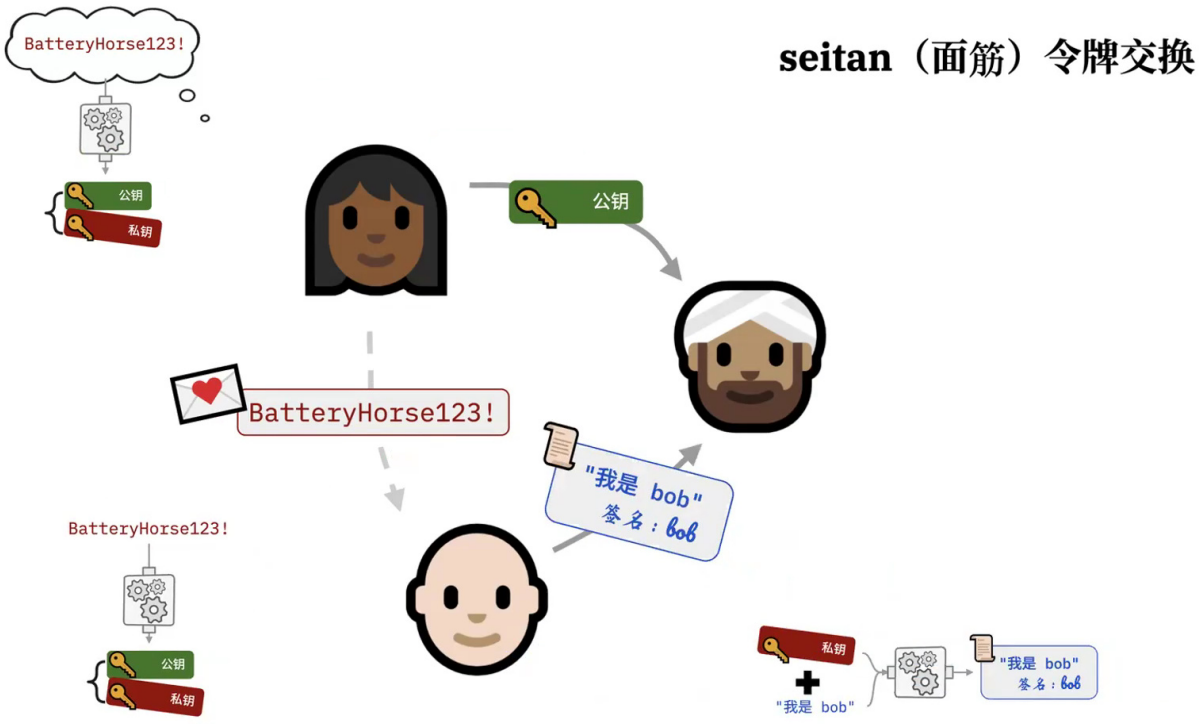
They came up with something called Seitan, presumably the joke being that seitan is firmer than TOFU. I found this approach when I was thinking about this in 2020, and that was the approach I took. It gives us the kind of third-party verification that we were looking for. Here's a simplified picture of how Seitan works.



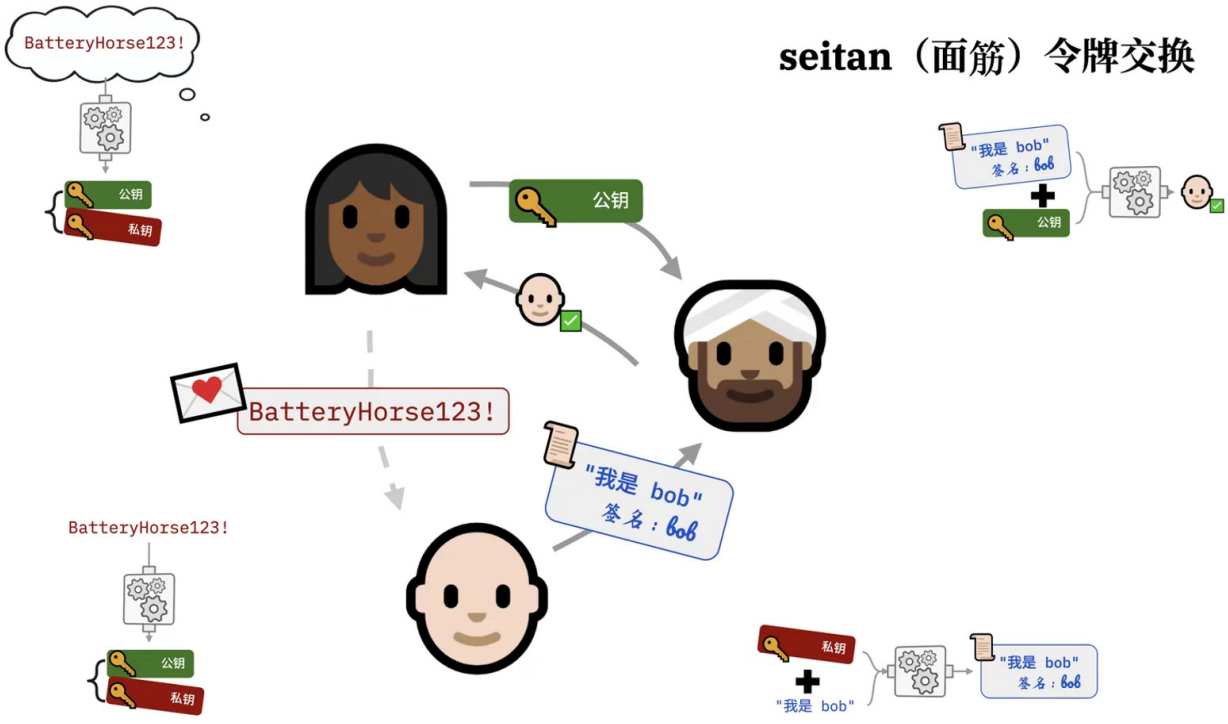
Deriving a key pair from an invitation · Frame 00:23:29

00:23:13 · Herb Caudill

Alice creates an invitation password. She derives a signing key pair from that password. She puts the derived keys aside and invites Bob by giving him the password through some existing trusted channel: she reads it to him on the phone, or sends him an email, or whatever. She gives Charlie the derived public key so that he can verify the invitation later.



Sending the invitation and verification key · Frame 00:24:01



Completing the Seitan token exchange · Frame 00:24:51

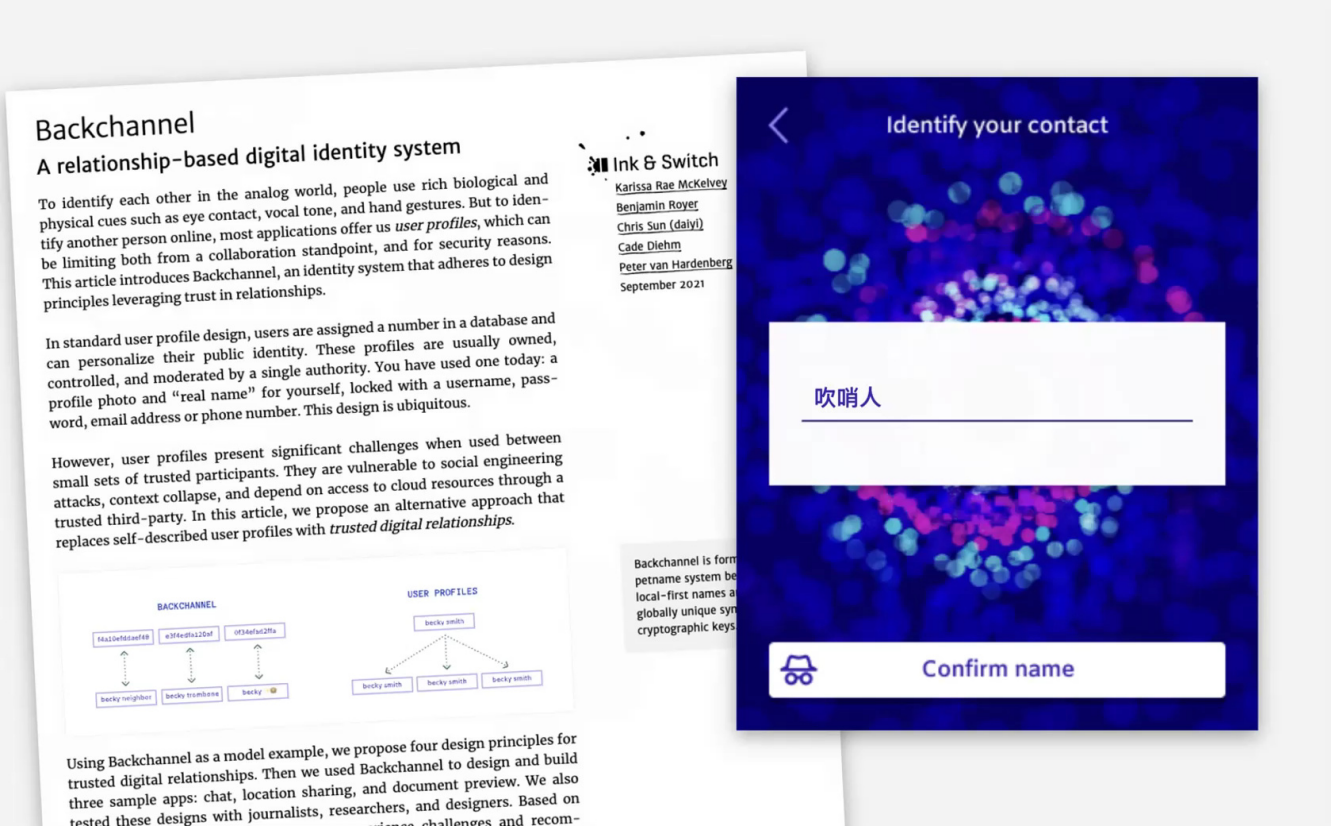
00:23:44 · Herb Caudill

Now Bob derives the same signing key pair from the password. He takes the private key from that pair, writes a document identifying himself, and signs it with the private key. His signature is now the proof that he knows the invitation password. He sends that to Charlie. Charlie takes Bob's signed statement, checks it using the public key Alice gave him, and concludes that this is indeed the person Alice invited. When he and Alice are back online together, he can communicate back to her that he was able to verify Bob.

00:24:29 · Herb Caudill

All of the keys we're seeing here are derived from the invitation. They're all specific to this invitation. Nobody is sending their own keys around to do this. So this initial trust still depends on Alice giving the invitation code to the right person through an existing channel, but it can be verified by a third party.

Identity is rooted in relationships



Relational identity and Backchannel · Frame 00:26:03

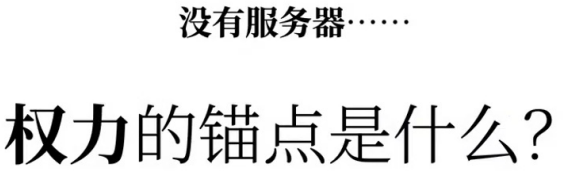
00:25:04 · Herb Caudill

We've been talking about identity, but that doesn't necessarily mean who you are in the world, in the sense of your name. It just means who you are to me. I know it's you because I gave you the invitation. That's how I know who I'm dealing with.

00:25:25 · Herb Caudill

Around this time, Ink & Switch had a project called Backchannel. One of the motivating use cases was connecting to an anonymous source, such as a whistleblower: having a journalist talk to somebody who didn't want to reveal their identity. The journalist doesn't need to know the source's name; the source can stay anonymous. Trust and identity are rooted in the interactions between these two human beings. I know you because we already interact. We can use that interaction to cryptographically secure connections, and use those connections to form a network of trust relationships.

Distributed access control and signature graphs



What anchors authority? · Frame 00:26:41

00:26:13 · Herb Caudill

That gives us an answer to our first question, and it's one that feels right to me: identity is anchored in human interactions and human relationships. We'll come back to that in a minute. We have a way of knowing who we're talking to, but now, how do I know who's on my team? How do I know who I should share with? How do I know who's allowed to do what? Who decides who's allowed to do what? How do you resolve that in this setting?



A distributed access control list · Frame 00:27:53

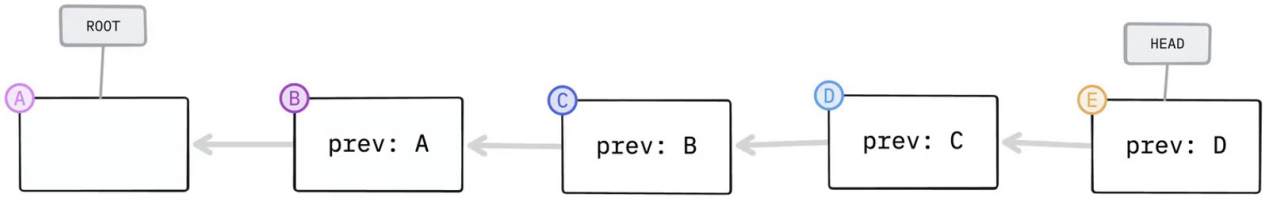
00:26:50 · Herb Caudill

I started by thinking about how to make an access control list. This is what a server uses: a list of identities and their permissions. My first thought was that, in a decentralized system, everybody just needs a copy of that list. Maybe we could just put it in an Automerge document and sync it.

00:27:19 · Herb Caudill

The problem is that the list needs to be tamper-evident. If somebody changes this document, it needs to be visible that the document was changed. It needs to be eventually consistent, so everybody who receives the same changes arrives at the same access control list. The changes also need to be authenticated, so we know who made them. We need a different structure from just a plain CRDT for this.

哈希链



✅ 可检测篡改

Hash chains · Frame 00:28:46

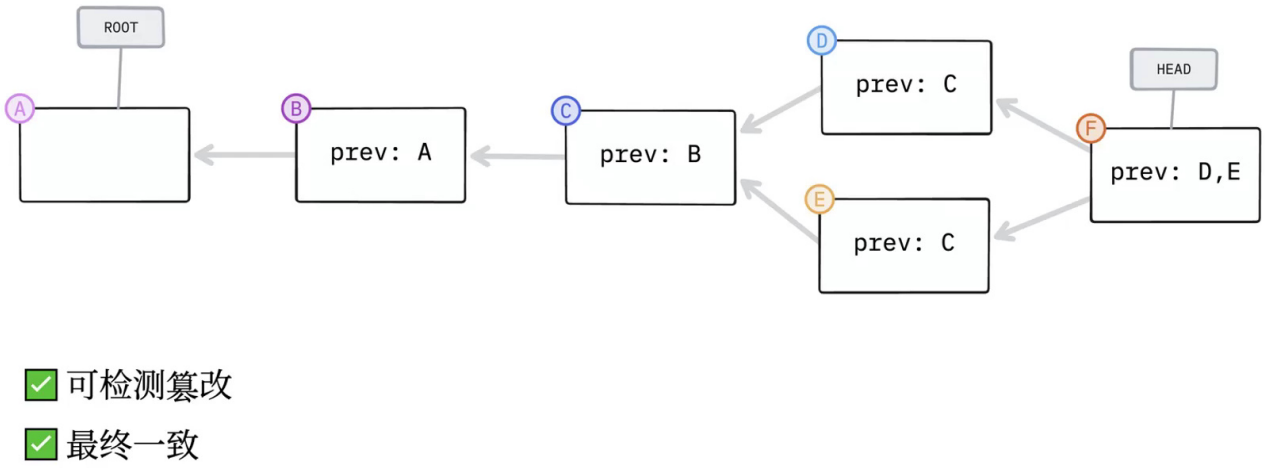
00:27:58 · Herb Caudill

For a tamper-evident history, the first thing you might think of is a hash chain. We start with the first link, A. The next link, B, contains a cryptographic hash of A. Then C has a hash of B, D has a hash of C, and E has a hash of D. E is the head of the chain. If anything changes in one of those earlier links, it breaks all of the hash references that follow it. This is how blockchain works. It makes tampering detectable because we have a trusted reference to check against.

00:28:41 · Herb Caudill

The problem is that a chain is linear. It doesn't give you a way to merge concurrent branches. Blockchain solves this by requiring you to spend a lot of energy to resolve conflicts in the correct way. But how would we do this in this setting?

哈希图



✅ 可检测篡改

✅ 最终一致

Hash graphs · Frame 00:30:19

00:29:03 · Herb Caudill

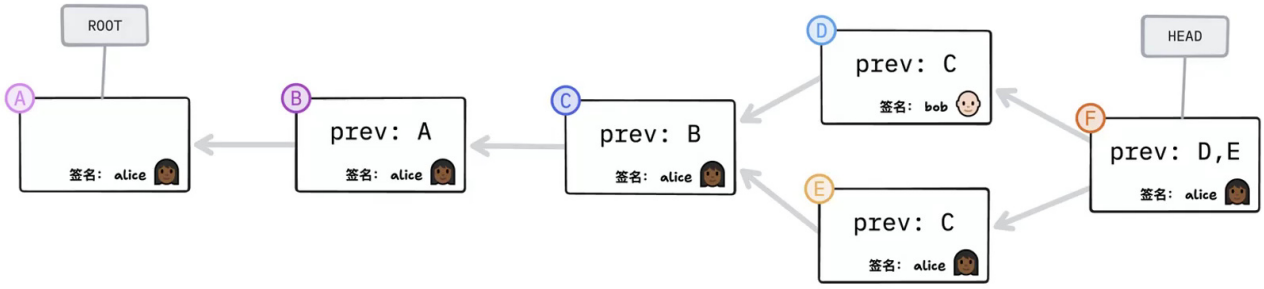
A hash graph adds branching, and branching allows us to represent concurrent changes. We start again with A. B refers back to A, and C refers to B. Now suppose one person adds link B and another person adds link E at the same time. They both refer back to C. The next person who knows about all the links that came before can refer back to two links instead of just one, and that effectively merges the graph.

> Editorial note: both automatic transcripts identify this branch as B; the accompanying diagram labels it D. The diagram shows D and E as the two concurrent branches.

00:29:50 · Herb Caudill

With suitable conflict resolution, this lets you replicate changes with eventual consistency. This is a CRDT: it lets peers merge their changes and reach the same results. Automerge uses a history of changes to support this kind of replication. This is exactly how Automerge works. Git also has a branching history, although you have to resolve any conflicts manually. So this is a well-understood way of doing this.

签名图



✅ 可检测篡改

✅ 最终一致

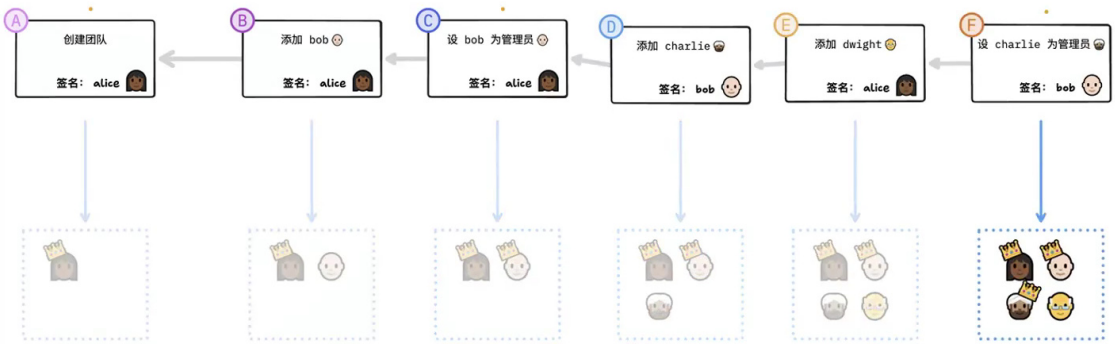
✅ 可认证

Signature graphs · Frame 00:31:23

00:30:27 · Herb Caudill

We still need a way to verify who made each change. A signature graph is a hash graph with the added twist that each operation is cryptographically signed by its author. Alice signs the first operation. She also signs B and C. Bob signs D on one branch, Alice signs E on the other branch, and then Alice signs F, which joins the branches. We can verify who signed each operation, as well as detect changes to the history. Any change to anything that happened here will be obvious: the signatures will fail.

签名图



Group operations in a signature graph · Frame 00:32:41

00:31:13 · Herb Caudill

We now have the three things we're looking for: tamper-evident, eventually consistent, and authenticated. But we still need rules that say which operations are allowed. We'll get back to that. The signature graph records operations that change the team. This is how it actually works in practice.

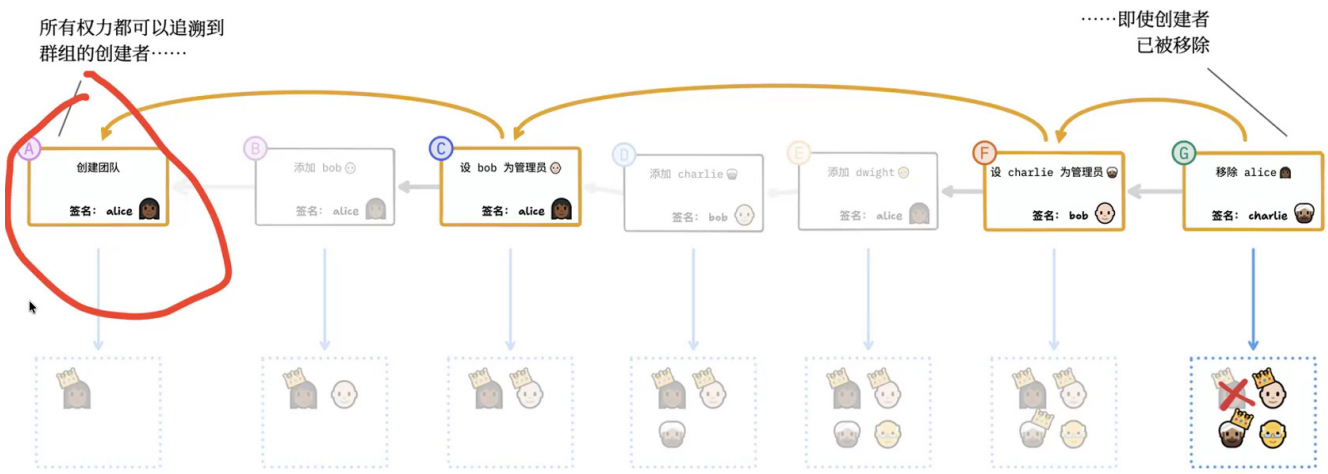
Membership, delegation and key rotation

00:31:52 · Herb Caudill

For example, Alice creates the team, so she gets to decide who's on it. At this point, she is the absolute dictator of the group. She's the only member, so it doesn't matter, but she is the absolute dictator. She adds Bob. It is still the case that only Alice is an admin in this group. But Alice can then add a link to the chain promoting Bob to an admin, delegating some of her authority to him.

00:32:32 · Herb Caudill

At that point, Bob can add Charlie. Say Alice adds Dwight, and Bob makes Charlie an admin. Charlie can now even remove Alice, because she delegated the authority that eventually reached him.

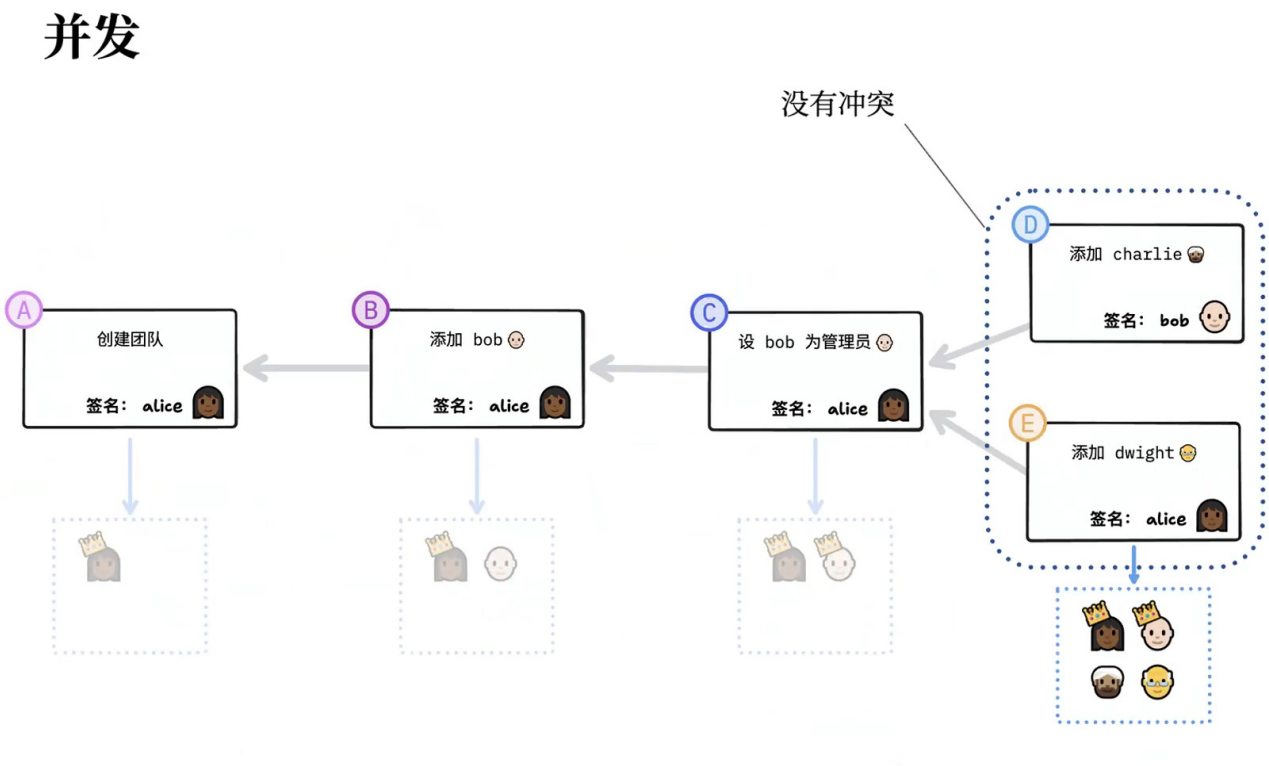


Member removal and key rotation · Frame 00:33:45

00:32:55 · Herb Caudill

The key insight for me is that the root of the graph, this first link in the chain, is what anchors authority in the group. Alice creates the team, so she gets to decide who's in it. Even if she's removed from the group, we still trace every action in the group back to the original founding link of the chain. The root of the graph gives us an anchor for authority. Every valid addition, every removal, and every further delegation ultimately traces back to the authority that comes from the fact that Alice is the one who created the group.

Concurrent additions and removals



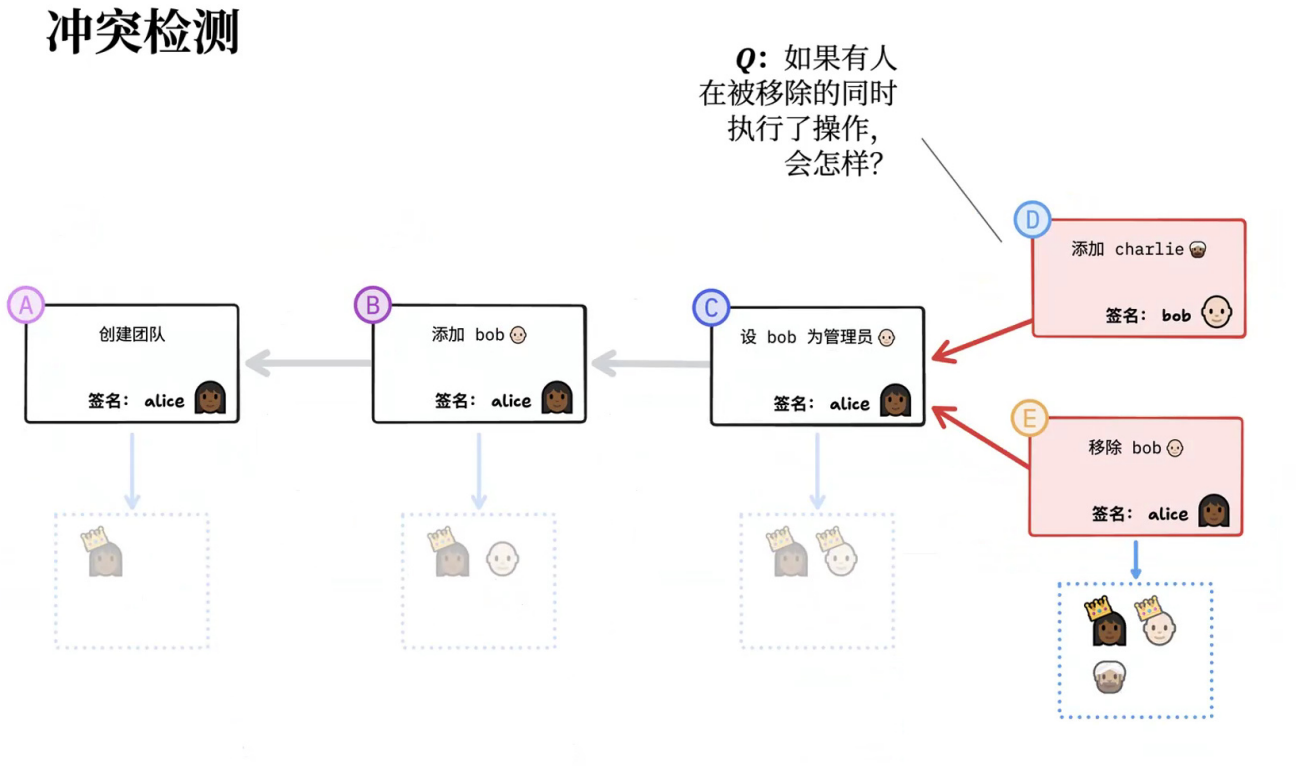
Concurrent non-conflicting operations · Frame 00:35:09

00:34:05 · Herb Caudill

To support changes while peers are disconnected, we need to be able to handle concurrency. The problem with a regular CRDT like Automerge is that it doesn't supply the permission rules we need in this context.

00:34:34 · Herb Caudill

In a very simple situation, you could use something like Automerge. Alice creates the team, adds Bob, and makes Bob an admin. Alice and Bob go offline, so each device now has its own copy of the history. Bob adds Charlie, and Alice independently adds Dwight. When their devices reconnect, they just merge the two histories. There's no conflict between these operations, and we can apply them in either order. So far, there's nothing here that you couldn't handle with a regular CRDT.



Concurrent removals and conflict detection · Frame 00:36:03

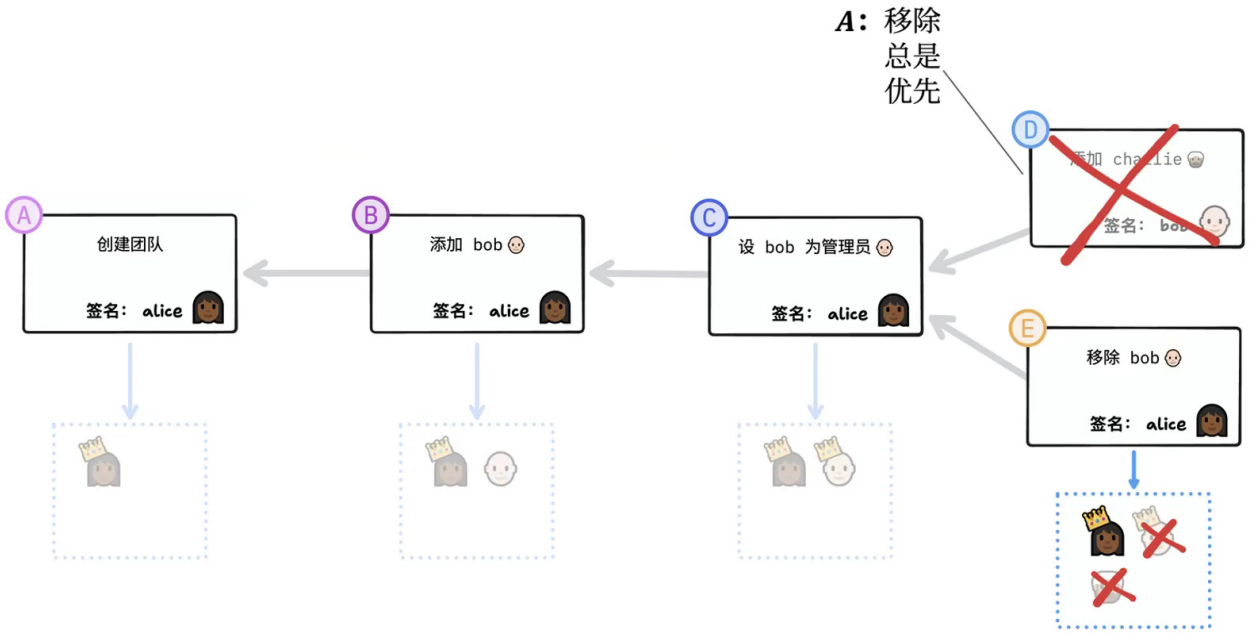
00:35:17 · Herb Caudill

But suppose Alice creates the team, adds Bob, and makes Bob an admin. What happens if Bob takes an action while Alice is removing him at the same time? Suppose Bob adds Charlie to the group, and while they're still disconnected, Alice removes Bob. When they try to merge their histories, we have a conflict. Is Charlie in the group or not? It depends on what order you take those two items in, or which ones you preserve. It's important to resolve this conflict correctly. Otherwise, Bob could get around his removal by adding another identity that he controls.

00:36:10 · Herb Caudill

The solution I came up with is to resolve the conflict by prioritizing the removal. This is what Martin Kleppmann calls strong removal. It means that Bob's concurrent addition of Charlie doesn't take effect, because he was being removed at the same time.

冲突解决

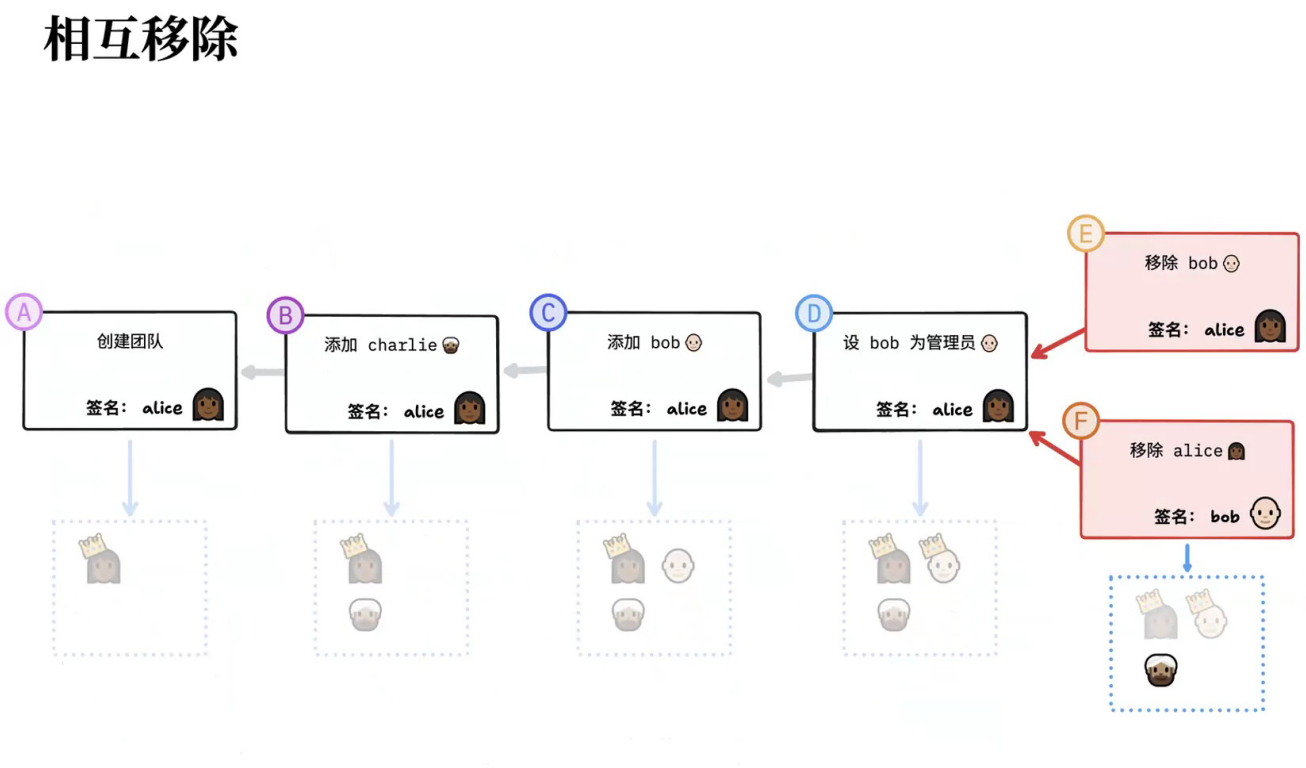


Strong removal · Frame 00:36:53

00:36:33 · Herb Caudill

A generic CRDT like Automerge can't provide that kind of logic. In fact, Automerge wouldn't even see these as a conflict. A conflict in Automerge is when you try to change the same property at the same time, whereas here, two different things happen that are in logical conflict. Basically, this means that any system you have is going to need a custom CRDT designed specifically for group membership.

When administrators remove each other

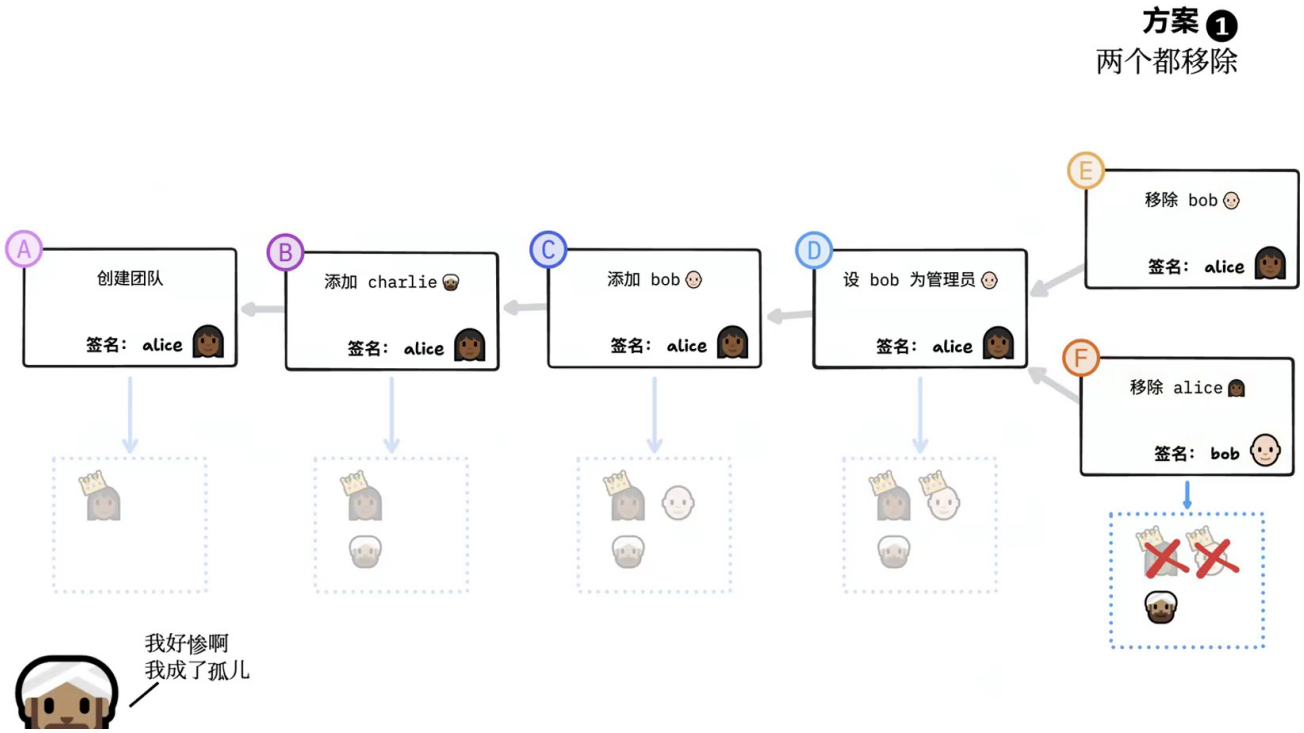


Mutual removal by administrators · Frame 00:37:46

00:37:07 · Herb Caudill

There are other situations where you might have conflicts. The obvious one you're probably thinking of is: what happens when two admins remove each other concurrently? Alice creates the team, adds Charlie, and so on. Alice and Bob are now disconnected. Bob tries to remove Alice, and Alice tries to remove Bob. Now what? We have an even more difficult conflict to resolve.

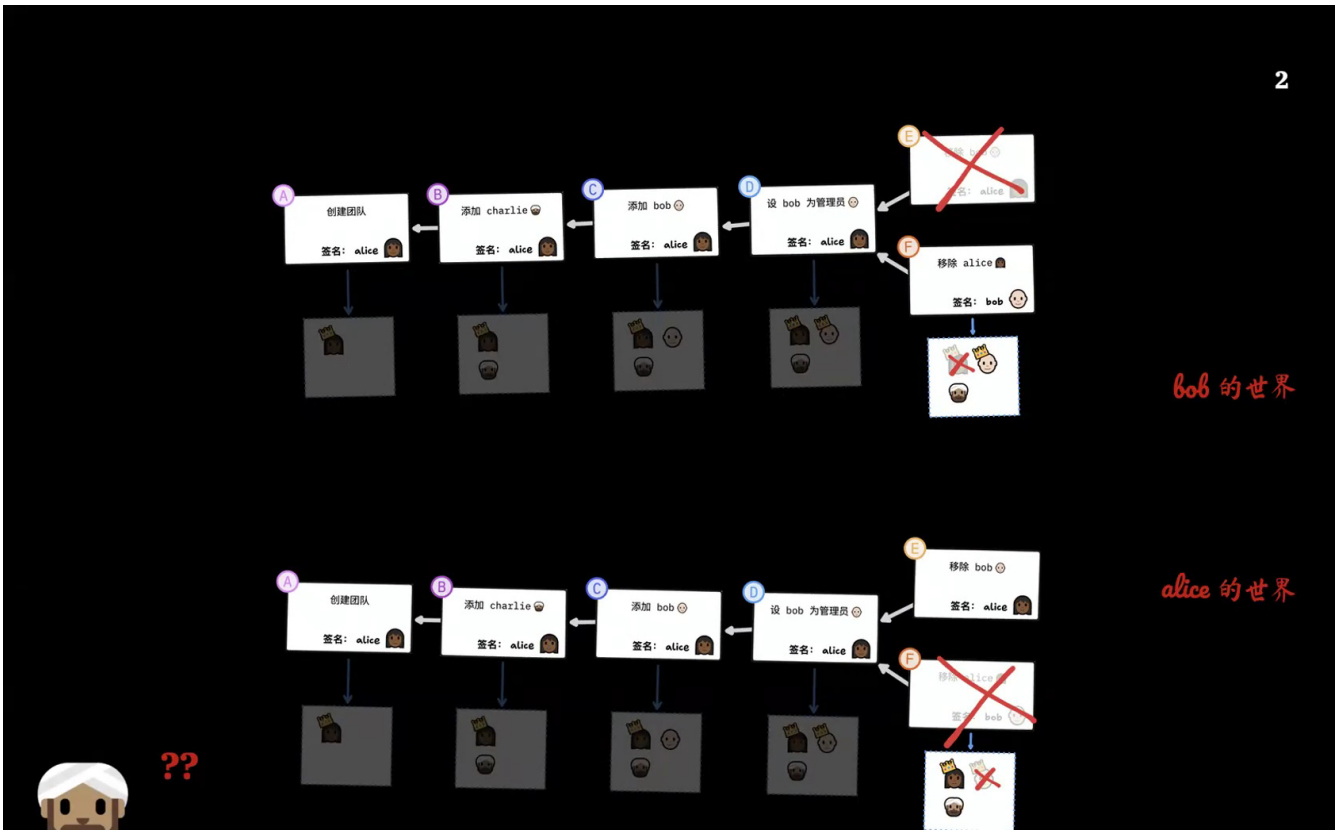
相互移除



Option one: remove both · Frame 00:38:20

00:37:43 · Herb Caudill

One solution would be to use the same approach as before, strong removal, and remove both admins. The problem is that Charlie, who is not an admin, is now the only member of the group. That's not the outcome you wanted, because the group is now useless. This creates an opening for what is essentially a denial-of-service attack: a rogue device can leave the group without an admin rather than accept its own removal.



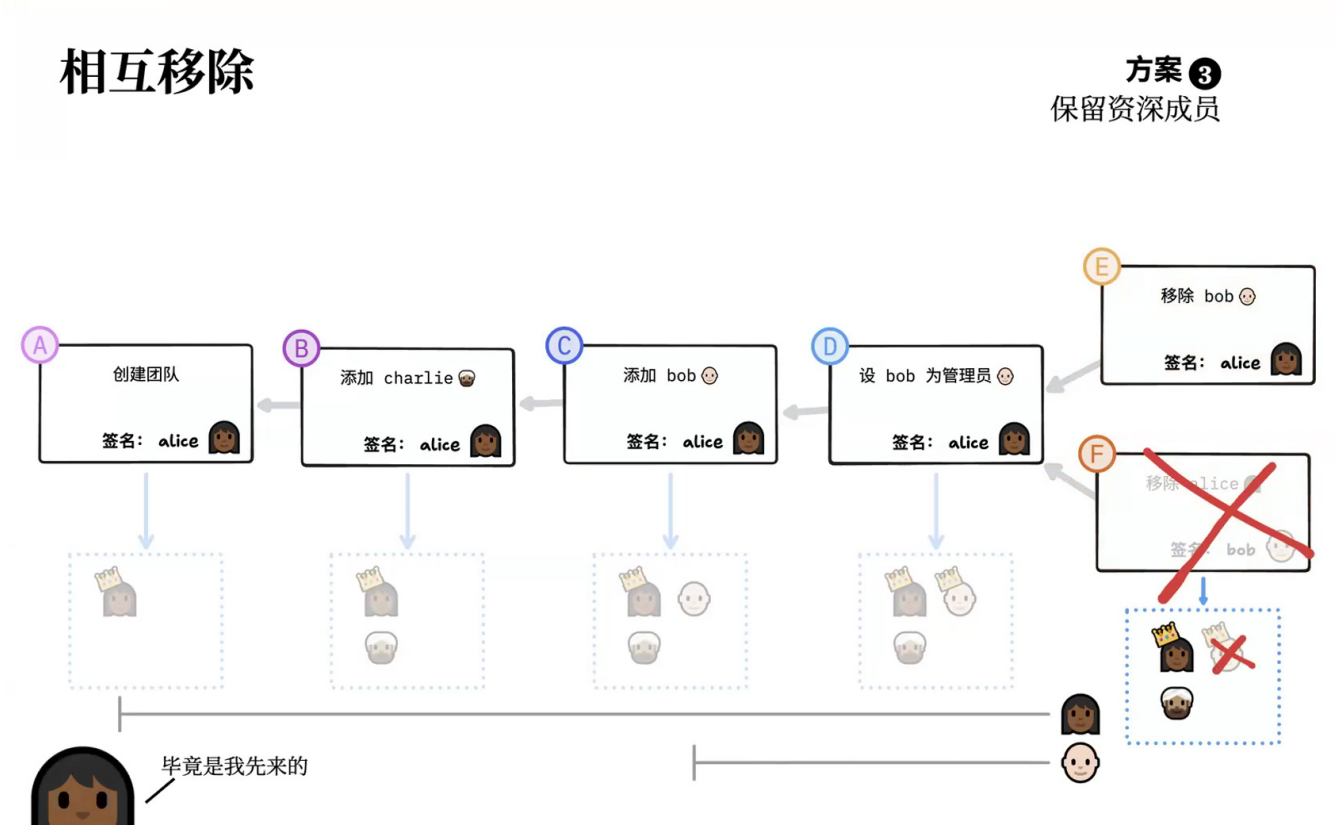
Option two: fork the group · Frame 00:38:45

00:38:26 · Herb Caudill

Another solution is to fork the world into two versions of the same group and have the remaining members decide which one they want to be in. In a way, this is brilliant because it reflects the reality of a decentralized system, but it leaves the members with a decision to make. It's like when the Catholic Church was split for a few years and there were two popes, one in Rome and one in Avignon. Every Catholic had to decide which pope they believed was the correct one. So that's not a great solution either.



The church-split analogy · Frame 00:39:08

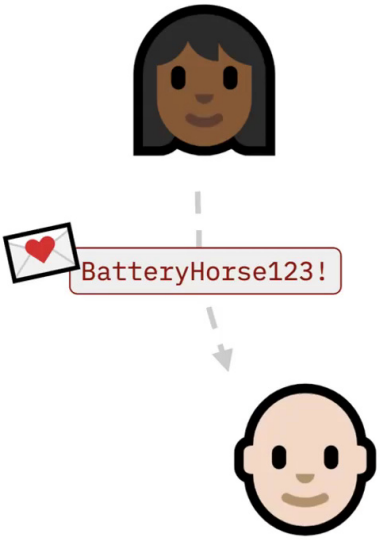


Option three: seniority · Frame 00:39:32

00:39:14 · Herb Caudill

The solution I ended up with is to use seniority as a proxy for who's more likely to be right. These lines show how long Alice and Bob have been in the group. Alice has been in it longer, so her removal of Bob takes effect. That was the choice I made in my library.

What anchors identity and authority



Identity rooted in human relationships · Frame 00:40:41

00:39:39 · Herb Caudill

We started with two big questions about how to find our footing in a local-first world. The first was how to anchor identity, and the second was how to anchor authority. This invitation protocol involves a lot of cryptographic subtleties, but at its core is the interaction where Alice gives Bob an invitation code. When I first designed this system, that seemed like cheating. It didn't seem like a principled solution.

00:40:18 · Herb Caudill

But the more I think about it, the more I think this is correct. That pre-existing side channel, whether it's email, talking on the phone, or meeting in person, represents the fact that the initial trust exists. That's not nothing. It really is the core of where trust comes from in this situation. This trust exists because Alice and Bob already have some relationship in the world. It's that existing relationship that gives us an anchor for identity.

没有服务器……

Q.

身份的锚点是什么?

Q.

权力的锚点是什么?

The two questions revisited · Frame 00:41:19



Authority rooted in creation · Frame 00:42:16

00:41:12 · Herb Caudill

That's our first answer to what anchors identity. What anchors authority? To go back to these chains: What anchors authority is the act of creation. Whether it's a company that you founded, a project that you started, or a document that you created, creating something gives you that initial authority over it. It doesn't mean that you have to keep the power all to yourself. You can delegate your authority to other people.

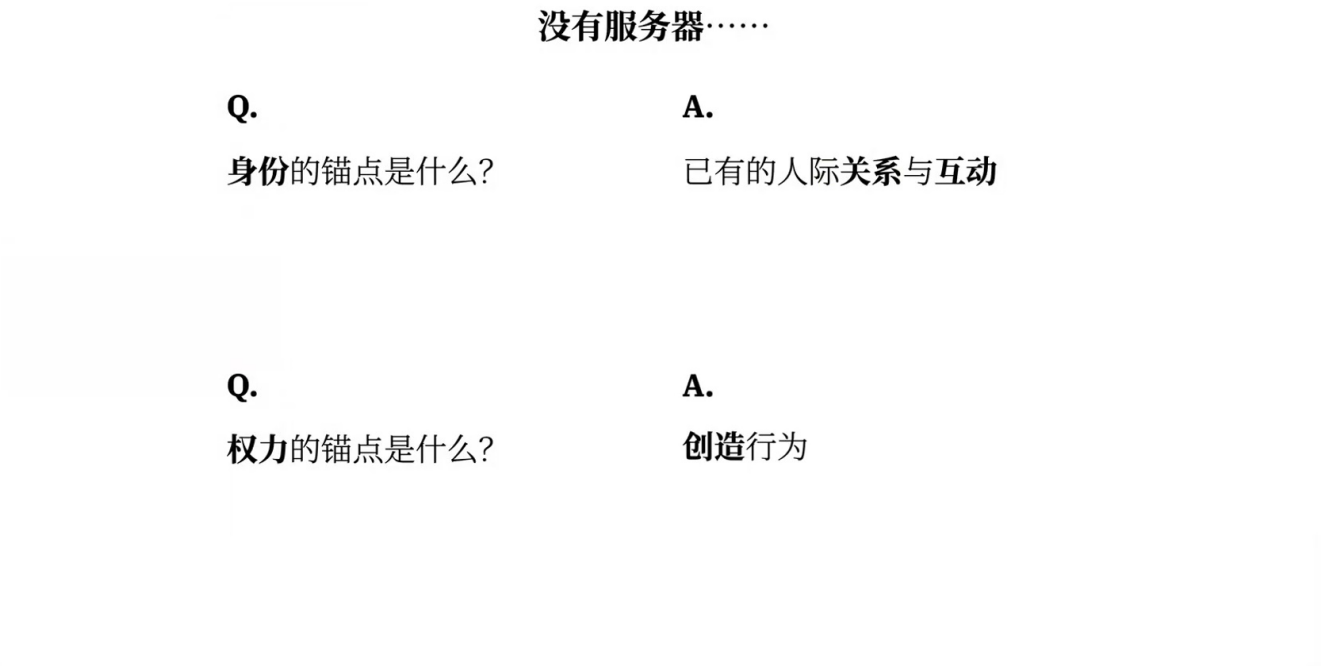
00:41:56 · Herb Caudill

It doesn't even mean that you have to keep that authority until the end of time. You might create something and then leave it, just like the founder of a company can eventually leave the company and the company carries on. But even in the case of a company, that authority ultimately traces back to the person who created it, even if they're not there anymore. Even if there are people in the company now who are more powerful than the founder, it all comes from that founder's delegation of authority.



auctor 拉丁语	使之增长的人 发起者、促成者、行动者、创始人 作者身份、能动性
auctoritas 拉丁语	影响力、威望、名声 被赋予的权力 责任、意见、判断
auctorité 古法语，*842–1400*	
autorite 中古英语，*1100–1500*	
authority 英语	执行规则或发布命令的权力

Authorship and authority · Frame 00:42:51

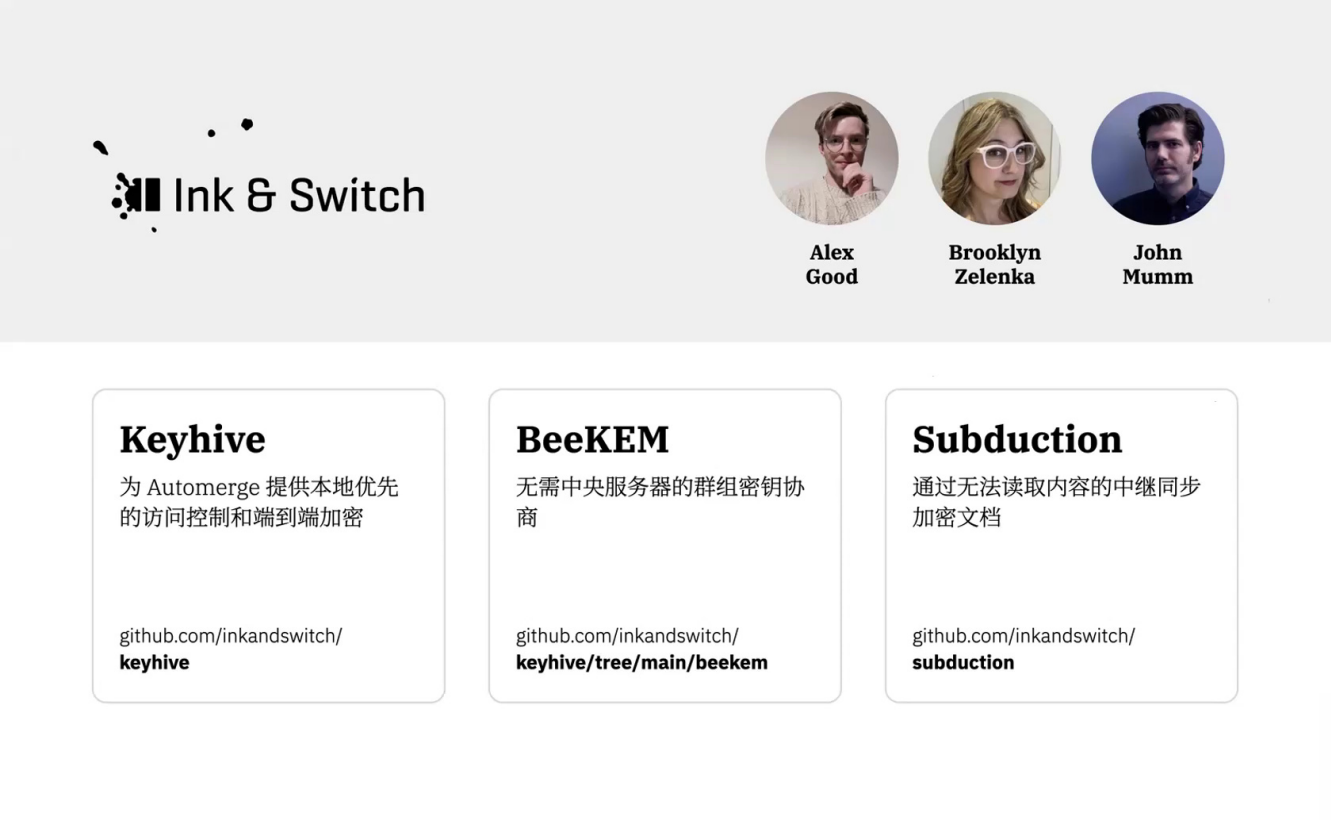


Answers about identity and authority · Frame 00:43:11

00:42:36 · Herb Caudill

In English, the connection is right there in the word. Authorship and authority both come from this idea of creating something and having that power over the thing you created. These are the answers I came up with and implemented in code in my library: human relationships and interactions anchor identity; the act of creation anchors authority. That feels like a correct and satisfying answer to those questions.

Keyhive, BeeKEM and Subduction



Keyhive, BeeKEM and Subduction · Frame 00:44:21

00:43:20 · Herb Caudill

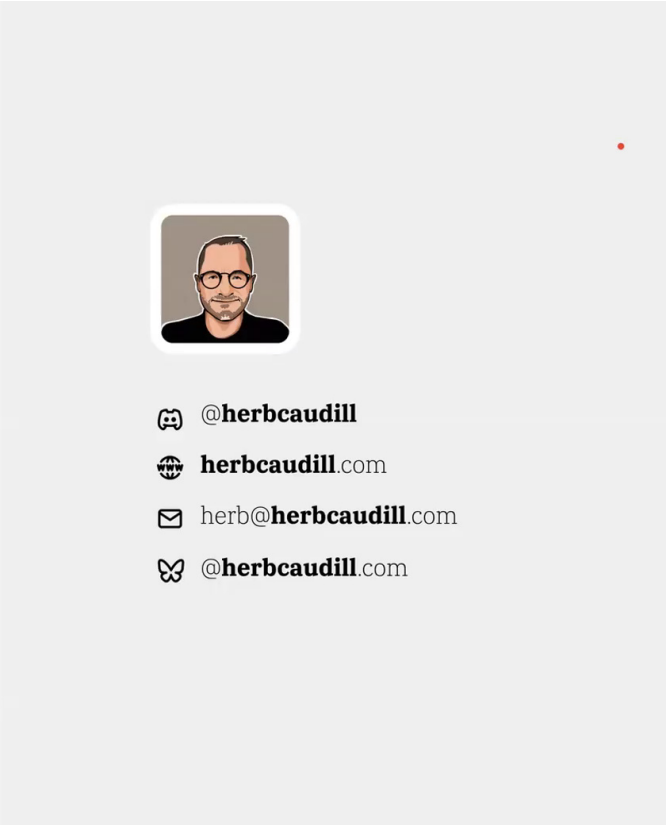
The spiritual successor to my library is something called Keyhive. It takes essentially the same ideas and implements them in a much more production-ready way. Although I should say that's the intent: Keyhive is still in alpha. But I've used it, it works, and it should be live and ready to be used very soon.

00:43:58 · Herb Caudill

There are a couple of other pieces to this. You have something called BeeKEM, which is a separate library—or a separate package within this library—that creates a cryptographic structure for dealing with all the keys and the history of keys within a group. And you have Subduction, a sync protocol and a sync server for relaying encrypted documents without having access to their contents.

00:44:36 · Herb Caudill

All of this is very cool technical work. People much smarter than me are working on it now—real cryptographers, as opposed to someone who learned cryptography during COVID. I'd encourage you to check these projects out. That's all I have. This is my contact information; I'd be happy to discuss any of these things further.



Contact details and questions · Frame 00:45:31

00:45:07 · Herb Caudill

I see there's a question in the chat.

00:45:12 · Host

Yes.

Q&A: contribution and authority

00:45:17 · Herb Caudill

Does that mean, if there's a contribution scoring system for each user in the chain, that we could also use contributions to determine power? If the chain is equipped with a contribution score, right?

00:45:31 · Herb Caudill

There are different solutions. I think mutual removal is the hardest question here. The solution I came to is about the depth of the links going back to when a person was added. If I was added ten links ago and you were added five links ago, I have seniority and I win in a mutual-removal situation.

00:46:11 · Herb Caudill

There are different ways to solve this. You could also have a system where you say Bob is Charlie's boss, even though Charlie was hired earlier, so Bob is going to win that conflict. You could find a way to encode those relationships. But obviously that becomes more complicated.

00:46:38 · Herb Caudill

I believe Keyhive makes mutual-removal resolution pluggable, so you can override the logic it uses. I'm not sure whether that's actually the case, though. But certainly there are different solutions to this.

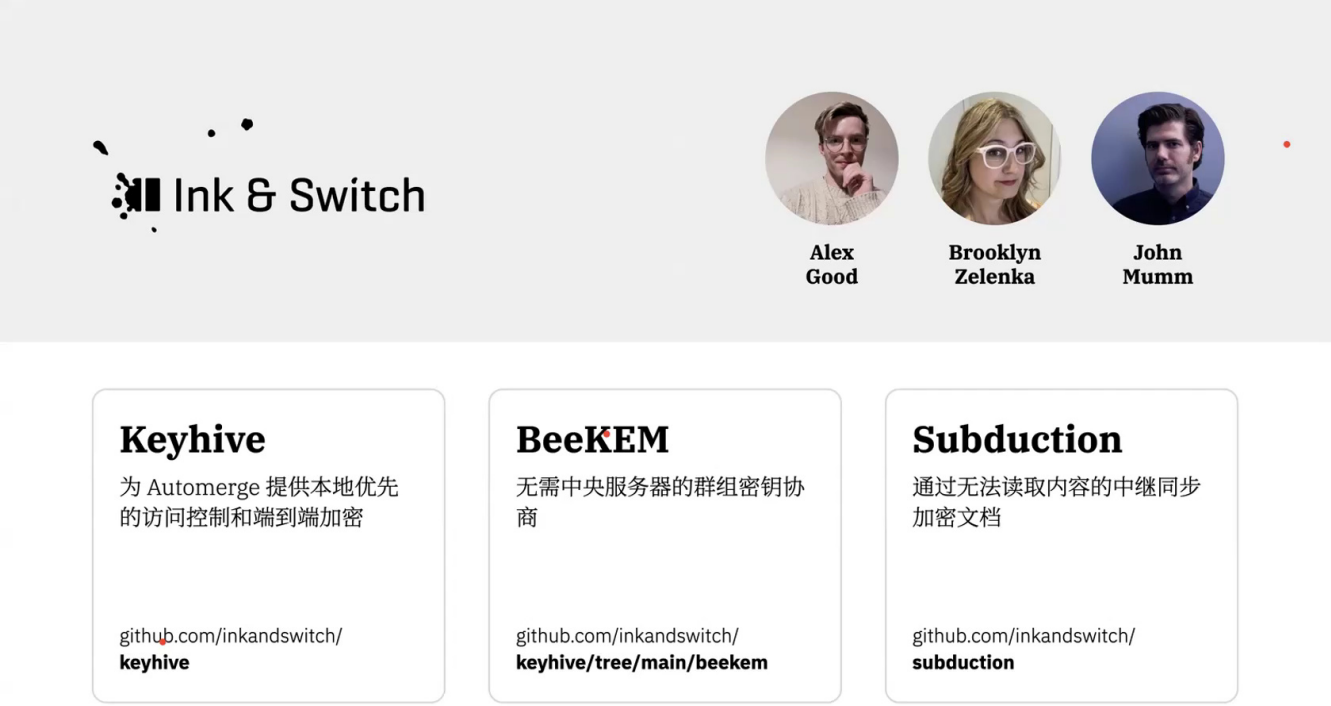
Cryptographic primitives and document permissions

00:47:09 · Herb Caudill

We have another question: ‘Are there new cryptographic primitives that could make it easier to build trust?’

00:47:18 · Herb Caudill

What I found was that, in my library, I didn't need to come up with any new cryptographic primitives. The Keyhive team did come up with some interesting new primitives; the main one is BeeKEM.



Q&A: revisiting Keyhive · Frame 00:48:41

00:47:49 · Herb Caudill

I won't describe it, primarily because I don't think I could describe it accurately. But it's a very cool way of building a tree of relationships between cryptographic keys. It's very efficient—much more efficient than what I ended up cobbling together in localfirst/auth.

00:48:23 · Herb Caudill

Another question: ‘The example is about member management. Is a similar method used for granting document access rights?’

00:48:29 · Herb Caudill

Yes. In Keyhive, basically every document is a group. You can have a fairly complex delegation graph. You can say, ‘Here's this document,’ and associate it with five people; that association pertains to the document.

00:49:17 · Herb Caudill

The internal system is very elegant: documents, groups, devices, and individuals are all basically the same kind of thing. If you think about it, a group consists of individuals, and an individual consists of a group of devices. I might have a laptop, a phone, and two different browsers, for example.

00:49:59 · Herb Caudill

A document can be accessed by multiple people, so all these things are treated as the same sort of internal construct. That's all I'll say about it, because I don't want to misrepresent it. It would be great to get Alex Good or Brooklyn to meet with you and talk more about the solutions they came up with while working on this.

00:50:43 · Host

Hi, Herb.

Comparing with Secure Scuttlebutt

00:50:45 · Herb Caudill

Yes?

00:50:45 · Host

I have a question. Someone in our community introduced a protocol called SSB, Secure Scuttlebutt. It's a kind of person-to-person network. I'm interested in how it differs from your product, like Keyhive, and in how you think about trust models—how you work through and formulate those models.

00:51:38 · Herb Caudill

Right. There are a lot of similarities with Secure Scuttlebutt. I did look at SSB when I was working on this.

00:51:49 · Host

Oh.

00:51:50 · Herb Caudill

I actually don't remember what it was about SSB that made it unsuitable for this particular context. But there was something; I think it probably had to do with some of the specific rules for managing group membership. There's also another issue with SSB…

00:52:26 · Host

Yes.

00:52:26 · Herb Caudill

It's a very inefficient protocol in the sense that you kind of have to have everything in the universe before you can do anything. My experience when trying to use it was that you sit there waiting for a spinner for a very long time, because it has to bring in every document it knows about…

00:52:51 · Host

Mm-hmm.

00:52:51 · Herb Caudill

…before you can do anything. But it's a very cool project. SSB is a very cool project, and I know some of the people who created it.

00:53:03 · Host

Mm-hmm.

00:53:04 · Herb Caudill

It's a very cool project, and it might work for your use case.

00:53:13 · Host

Thanks. From what you're saying, it sounds as though you prioritize efficiency in establishing trust. That's my impression.

00:53:29 · Herb Caudill

Could you say that again? Prioritize the…?

00:53:31 · Host

The efficiency. Okay.

00:53:35 · Herb Caudill

Right, yes. I think it's more about a more efficient way of establishing that trust in the first place.

00:53:47 · Host

Cool, thanks.

Social relationships and invitation security

00:53:56 · Herb Caudill

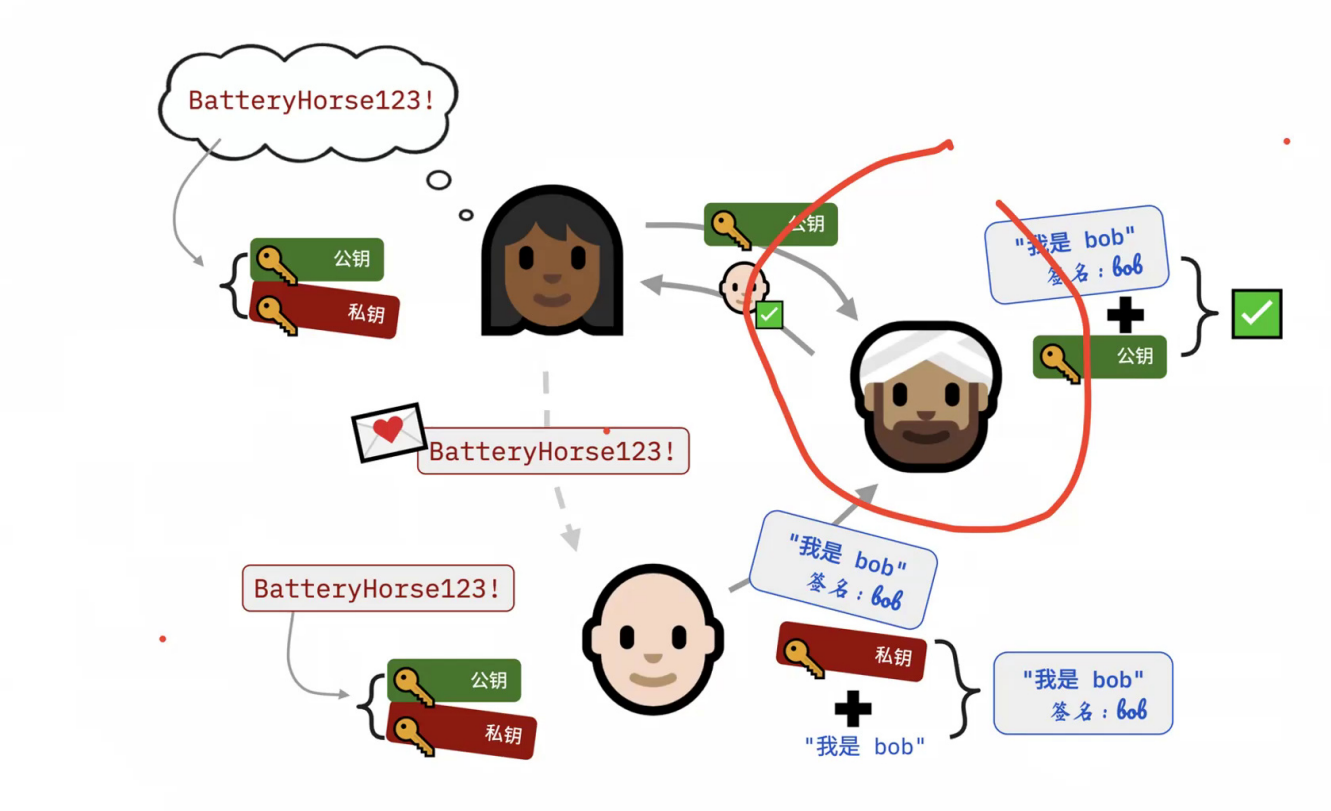
Another question: ‘I really like the idea of grounding authority in real-world social relationships. Do you think your work in international development helped you recognize the importance of offline social scenarios?’

00:54:08 · Herb Caudill

Well, yes, maybe. For sure.

00:54:17 · Question read aloud

I have a question here; let me translate. ‘Is it possible for an asymmetric PAKE to cause the introducer to be attacked?’



Q&A: the invitation protocol with annotations · Frame 00:56:31

00:54:36 · Herb Caudill

Do you mean the third party in this scenario? Like Bob, in this case? I'm not entirely sure—I might not have gotten a good translation of that question.

00:55:05 · Host

I think this is the question from that audience member: the introducer being attacked. Do you mean Bob being attacked? I'm asking because he seems not to have fully understood what you mean.

00:55:33 · Host

Hello? Is the person who asked that question there?

00:55:40 · Herb Caudill

I'll answer…

00:55:42 · Host

Yes.

00:55:43 · Herb Caudill

…the most recent question while we're waiting: ‘How many peers can collaborate using Keyhive and Subduction today?’

Scale and independent communities

00:55:51 · Herb Caudill

I'm not sure what the limits are. But when we designed the parameters or requirements for Keyhive a couple of years ago, the target was to handle Wikipedia scale: the number of documents, users, and editors on Wikipedia.

00:56:33 · Herb Caudill

It was very ambitious in terms of scalability. I'm not sure whether they've hit that target yet. But it's definitely much more scalable than my library was. It was completely rewritten in Rust, again by programmers much more skilled than me.

00:57:09 · Herb Caudill

There's a response here. Okay.

00:57:26 · Host

Yes. Okay, there's a new question.

> Chat context (written in the meeting chat, not spoken aloud): a participant suggested that people who do not know one another offline could build trust and collaborate, forming communities around shared goals or subcultures and helping to create a more resilient world. The next answer responds to this comment.

00:57:47 · Herb Caudill

Yes, absolutely. That's definitely the hope here. In addition to being a better programming model that doesn't make us dependent on tech companies, it lets you build these communities from the ground up without depending on someone else's servers or resources. I think that's true.

Key rotation, identity conflicts and malicious members

00:58:39 · Host

There's a question about what happens if the scale is quite large. I think I would understand it as a Byzantine attack. [Unclear.]

00:59:00 · Herb Caudill

Right. So, if someone resets their private key…

00:59:06 · Host

Mm-hmm.

00:59:07 · Herb Caudill

What do you do if there's an identity conflict when the key is distributed? In my library, key rotation works like this: you have to be an admin to initiate a key rotation, or you can rotate the keys for your own devices.

00:59:32 · Herb Caudill

I think in Keyhive, which has a much more abstract and elegant approach, if you have administrative rights over a group, you can rotate those keys, and nobody else can do so.

00:59:58 · Host

Mm-hmm.

01:00:09 · Herb Caudill

Right. Sorry, I'm translating the chat.

01:00:39 · Herb Caudill

Again, identity is defined this way: if Alice invites Bob, Alice is stating that the person she invited is Bob. The person who sends the invitation is the person who does the naming.

01:01:06 · Herb Caudill

Once Bob accepts the invitation and supplies his own private key, we can use that private key to identify anything Bob has done. So, yes.

> Editorial note: both Soniox and the Zoom captions render these two references as “private key.” We preserve that wording here rather than silently changing the talk. In the speaker’s protocol description, private keys remain on the device; public keys are exchanged to verify identity and signatures. Read this distinction alongside the passage; it is not an instruction to disclose a private key. Speaker’s protocol description

01:01:36 · Herb Caudill

I suppose two people could each say, ‘This is Bob.’ The library doesn't treat that as a first-class concern. I think that's probably still an open area. Anyway, it's not a problem this library tries to solve—the question of ultimate identity.

01:02:08 · Herb Caudill

I've seen other projects, such as DXOS, a library built on Automerge and Keyhive. It adds a layer using DIDs—decentralized IDs—tied back to, for example, the Bluesky or AT Protocol world. You give up a little decentralization by referring to some trusted store of identities. That's one solution.

01:03:06 · Herb Caudill

With malicious behavior, I think it's up to the admins to remove bad actors from their group. That's the technological solution. Beyond that, there may be other ways to communicate, outside this system, that somebody is not to be trusted.

01:03:46 · Herb Caudill

Identity definitely gets tricky: actually knowing somebody's real-world identity and agreeing on it. There are distributed solutions, and there are central-server-based solutions.

01:04:16 · Herb Caudill

But the threat model behind Keyhive and localfirst/auth assumes that whoever invited a person has the authority to say who that person is. It doesn't deal with two people claiming to have invited the same person when they are actually different people. But that's an interesting problem.

01:05:01 · Host

Mm-hmm.

01:05:37 · Host

Okay. Any new questions? Does anyone have more questions?

01:06:07 · Host

Okay, if there are no other questions, then maybe… Oh, there's a new question.

Bulletin boards and identity revocation

01:06:30 · Host

The new question is: could we register a cryptographic signature or credential on a bulletin board in advance, and then, when needed, use it on the board to revoke that identity?

01:06:54 · Herb Caudill

If you do that, you're presupposing that you have a central bulletin board that everybody trusts.

01:07:05 · Host

Mm-hmm.

01:07:06 · Herb Caudill

That's essentially centralized public-key management. It's a legitimate solution, but it's a different problem space.

Peer discovery, sync and relay trust

01:07:39 · Herb Caudill

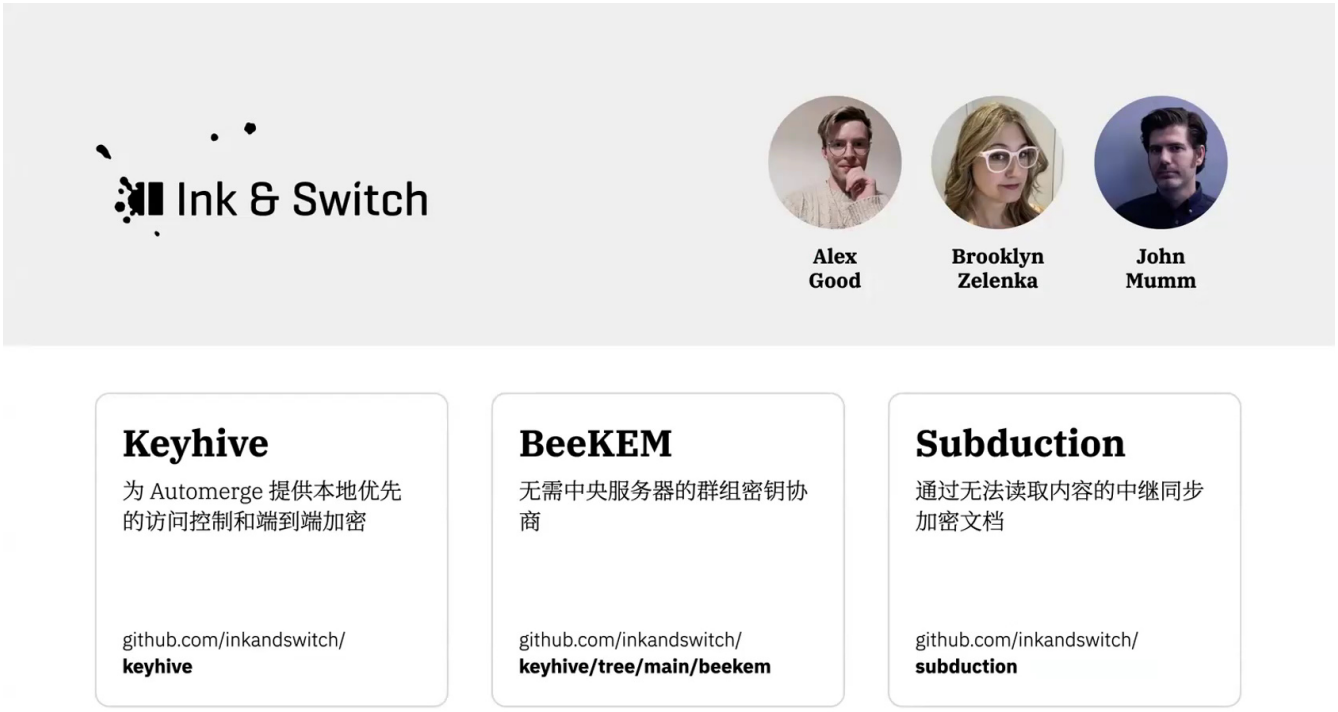
There was a question about how devices find each other and sync without a central server, and whether you need to trust relay services.

01:07:52 · Herb Caudill

There are ways to do that. Both my solution and Keyhive are network-transport agnostic, so they can work over whatever network protocol you have. If you have a trusted relay—or actually, the relay doesn't need to be trusted, because all the traffic is encrypted once you've established someone's identity.

01:08:38 · Herb Caudill

You can also do it completely peer-to-peer using something like iroh. There are many solutions to the question of how to interact with each other. Once you've authenticated with someone, the sync server doesn't need to be trusted: it can't see the contents of anything it stores or forwards. Everything you put on it is encrypted, and it doesn't have any of the keys.



Q&A: encrypted synchronization · Frame 01:10:06

01:09:26 · Herb Caudill

That's what Subduction is: a server and a sync protocol for encrypted documents whose contents the server cannot access or read.

01:09:58 · Host

Mm-hmm.

01:10:03 · Herb Caudill

Reticulum's network… Yes, I'm not familiar with Reticulum, but yes.