BeeKEM 与本地优先环境中的密码学密钥协商

Derek Yen · 2026-10-06 · 中文逐字稿

中文逐字稿（英文发言已翻译）。根据 2026 年 10 月 6 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。公式记号按幻灯片核对。时间对应交付视频。

分享者与研究背景



研究题目与合作者 · 画面 00:00:05

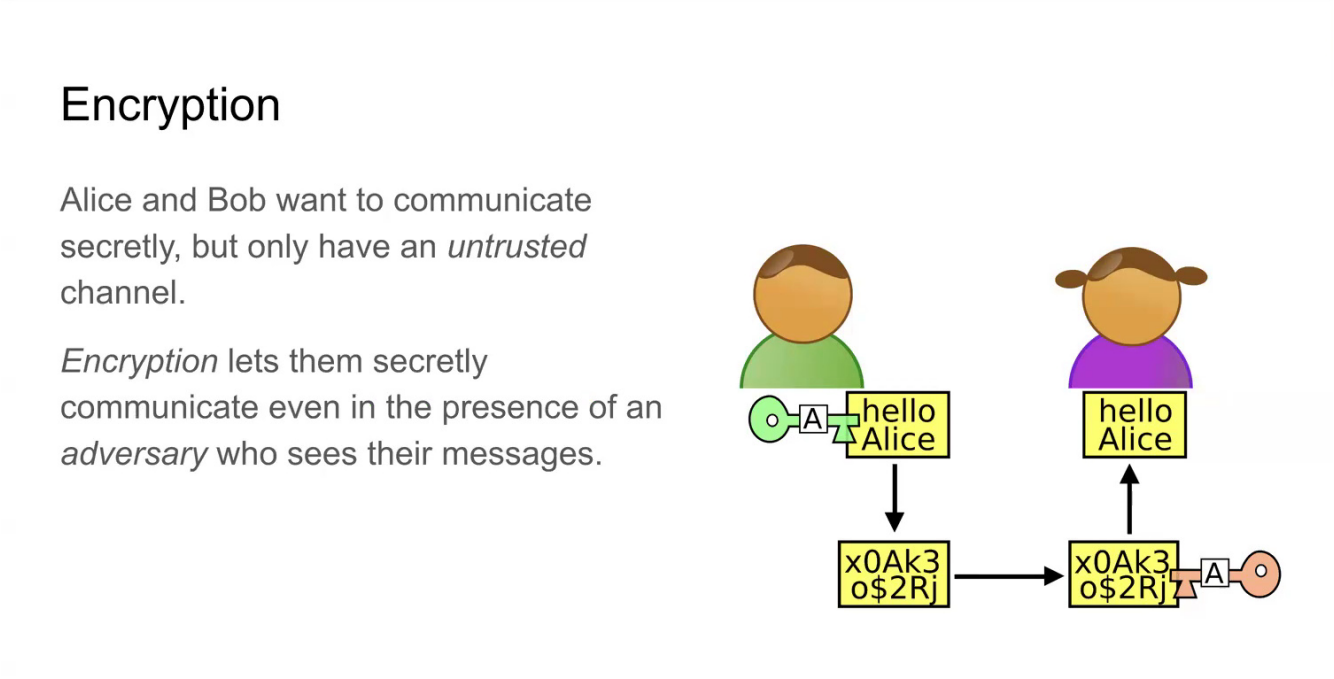
00:00:00 · Derek Yen

大家好，我叫 Derek，是纽约大学的三年级博士生。很高兴有机会介绍我们开发 BeeKEM 的工作，以及更广泛的、面向本地优先环境的密码学密钥协商。这是学术界与 Ink & Switch 产业研究团队共同完成的工作，合作者包括 Andrés、Liangrun、Martin、John、Sunoo 和 Brooke。先简单说一下今天的安排。

00:00:29 · Derek Yen

我会先介绍密码学密钥协商的基本背景，再谈谈从通常假设的中心化环境转向本地优先这样的去中心化环境时，会发生哪些变化。接着概括介绍我们的方案 BeeKEM：它是一种专门面向本地优先应用设计的去中心化持续组密钥协商方案。最后介绍我们开发的 BeeKEM 实现及其性能评估。

从加密到密钥协商



不可信信道上的加密通信 · 画面 00:01:25

00:00:59 · Derek Yen

首先从最基础的内容讲起，希望无论大家的密码学背景如何，都能理解。什么是加密？假设 Alice 和 Bob 想秘密通信，却只能使用一个不可信的信道。加密是一类算法，让他们即使面对能够看到传输消息的攻击者，也能秘密通信。

00:01:28 · Derek Yen

这张图展示了加密的基本思路：把一条可读的消息，比如“Hello, Alice”，通过一个称为密钥的秘密值，转换成看起来随机的文本。我们把这种文本称为密文，而它又可以被解密，恢复成原始消息。所以，密码学应用通常要求用户持有密钥，也就是某种密码学秘密，有点类似于电脑的密码。

Key agreement

Cryptography requires users to possess *keys,* or cryptographic secrets.

Key exchange or *key agreement* algorithms let two users establish a shared key.

- Underlying primitive for TLS, SSH, Secure Messaging

➔ What if there are more than two users? What if we have a group of users, e.g. secure group messaging?

Pairwise key exchange: n choose 2 = $O(n^2)$ protocols; O(n) adding new user

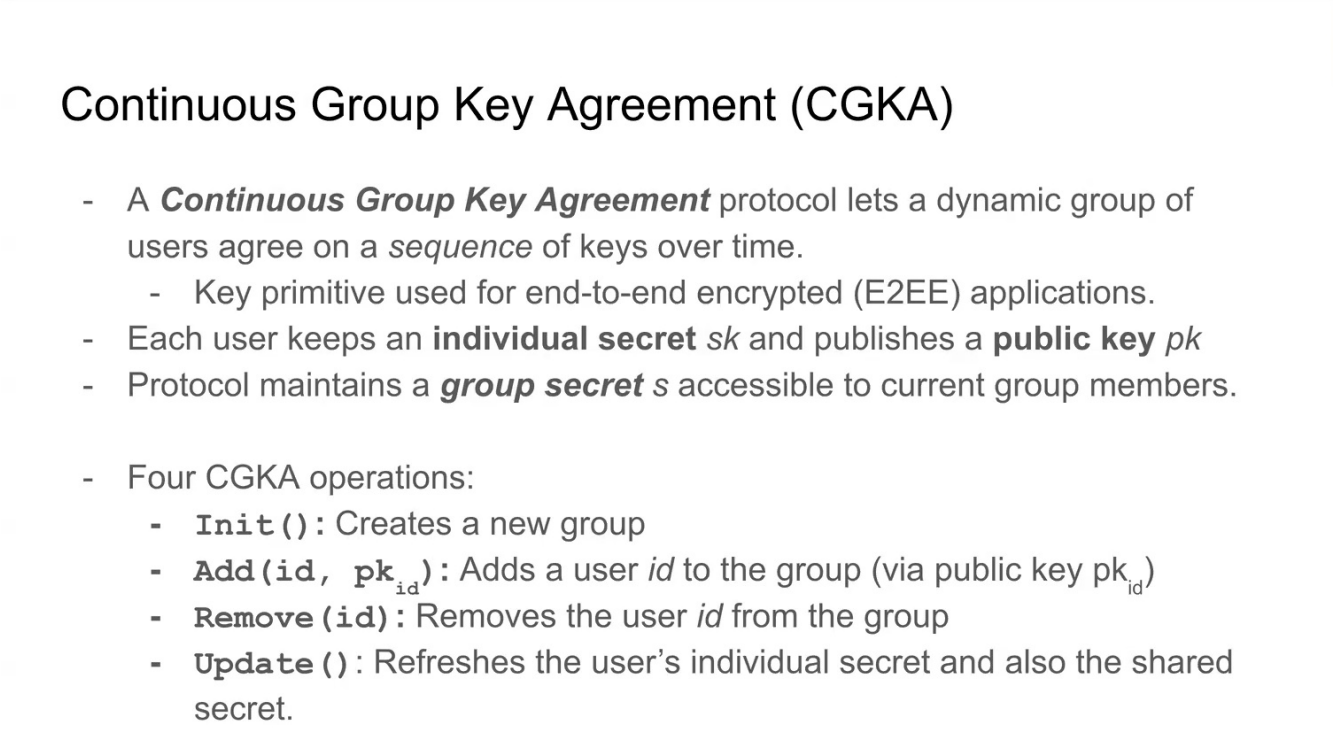
两方密钥协商与群组扩展 · 画面 00:03:15

00:02:06 · Derek Yen
怎样让应用中的不同用户持有这些密钥？这就要用到密钥交换或密钥协商算法、协议。两个用户可以通过公共网络通信，建立一个共享密钥；即使攻击者看到了那些数据包——准确地说，是那些消息——也不能得到这个共享密钥。大家可能听说过 Diffie–Hellman 密钥交换，它就是这样的密钥协商算法。

00:02:35 · Derek Yen
传输层安全协议（TLS）、SSH 和安全通信应用，都使用了这样的底层原语。接下来很自然会遇到一个问题：如果需要协商密钥的不是两个人，而是一组用户呢？比如端到端加密的群聊。当然，可以让所有用户两两运行密钥交换协议，但这样就需要 n² 次协议调用。

00:03:13 · Derek Yen
当组里有几百、几千名用户时，这就不太高效了。加入一名新用户时，他还得与每名现有成员交换密钥，成本是 O(n)，也不太可行。所以，一个自然的目标是为群组找到更好的方法，最好让添加成员等操作的成本低于线性增长。

持续组密钥协商 CGKA



CGKA 与四种基本操作 · 画面 00:05:25

00:03:39 · Derek Yen

这确实是一个成果丰富的研究方向。它定义了一类密码学原语，也就是一种称为“持续组密钥协商”（Continuous Group Key Agreement，CGKA）的密钥协商。顾名思义，CGKA 协议让一组用户随着时间推进，协商出一系列密钥，这就是“持续”的含义。这还是一个动态的用户组：成员不必在开始时就固定下来。

00:04:03 · Derek Yen

你可以添加新用户，让他们获得组秘密；也可以移除用户，让他们失去访问组秘密的能力。我稍后会解释，为什么需要的是一系列密钥，而不是一个密钥。CGKA 是用于端到端加密应用的原语，除了消息通信，也可能用于其他应用。

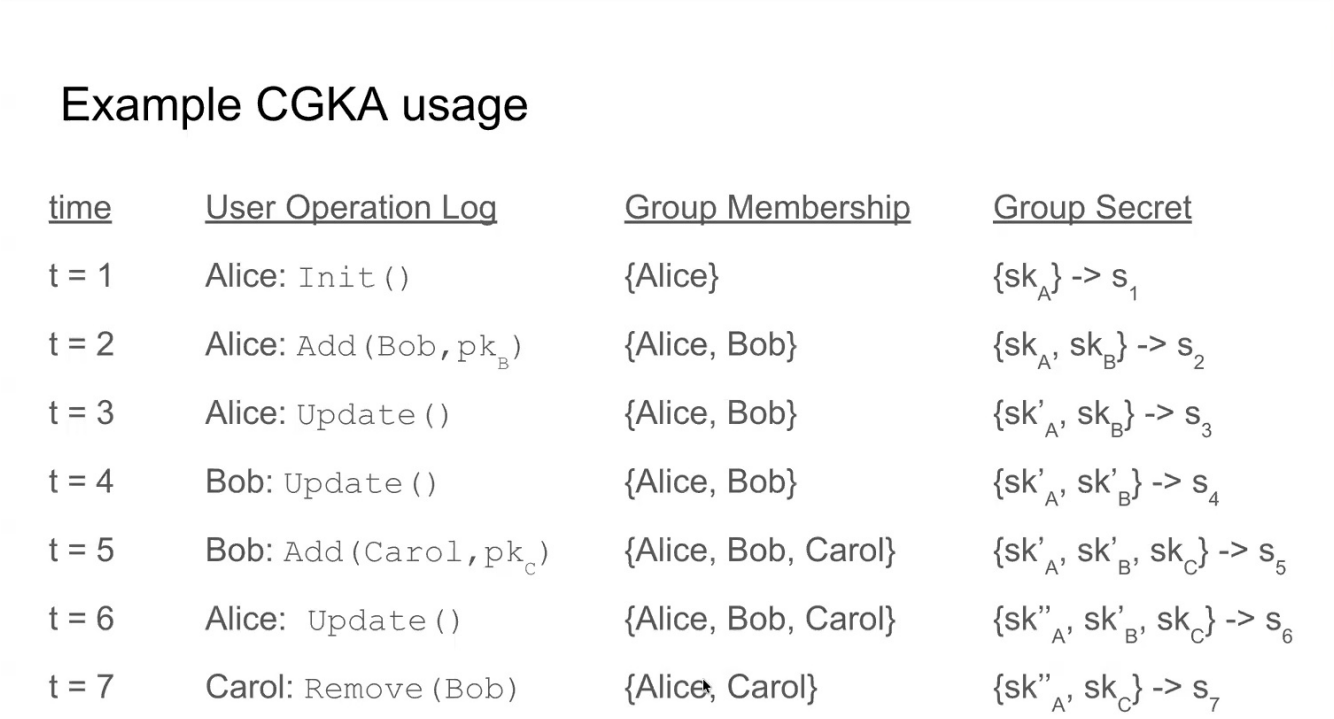
00:04:29 · Derek Yen

在这样的方案中，每位成员都有自己的秘密值，我把它记为 sk，同时发布一个公钥 pk。CGKA 协议维护一个组秘密，记为 s，它应当只有当前的组成员能够获取。更正式地说，持有某个 sk、且其对应公钥 pk 属于这个组的人，才能获取组秘密。CGKA 有四种基本操作。

00:04:58 · Derek Yen

init 创建一个新组；add 根据公钥 pk_id，把标识为 id 的新用户加入组；remove 从组中移除用户。第四种操作是 update，它让发起操作的用户刷新自己的秘密值和公钥，同时刷新共享秘密 s。我稍后会解释为什么需要这个操作，这也关系到为什么我们需要随时间变化的一系列密钥。

四种操作怎样改变组秘密



组成员与组秘密随操作变化・画面 00:06:45

00:05:30・Derek Yen

这里举一个 CGKA 的使用例子。左边是 t = 1、2 等离散时间点，以及不同用户发起的操作日志。右边两列分别列出当前组成员和组秘密，也标明哪些私钥能够获取相应的组秘密。

00:05:56・Derek Yen

比如，在 t = 1 时，Alice 用 init 创建一个新组。组里最初只有她，她可以通过自己的秘密值 sk_A 获取组秘密 s_1。在 t = 2 时，Alice 把 Bob 加入组，用他的公钥 pk_B 标识他，而 Bob 持有对应的私钥 sk_B。这时组里有 Alice 和 Bob，sk_A 与 sk_B 都可以获取组秘密 s_2。

00:06:24・Derek Yen

在 t = 3 时，Alice 调用 update，生成新的秘密值 sk_A′，组秘密也随之改变。这时 sk_A′ 和 Bob 原来的秘密值 sk_B 都能获取 s_3。这个过程可以继续：Bob 可以更新自己的密钥，可以把 Carol 加入组，最后也可能被移出组。产生的这一系列组秘密，分别对应不同的、能够访问它们的用户集合。

前向保密性与失陷后安全性

00:06:57・Derek Yen

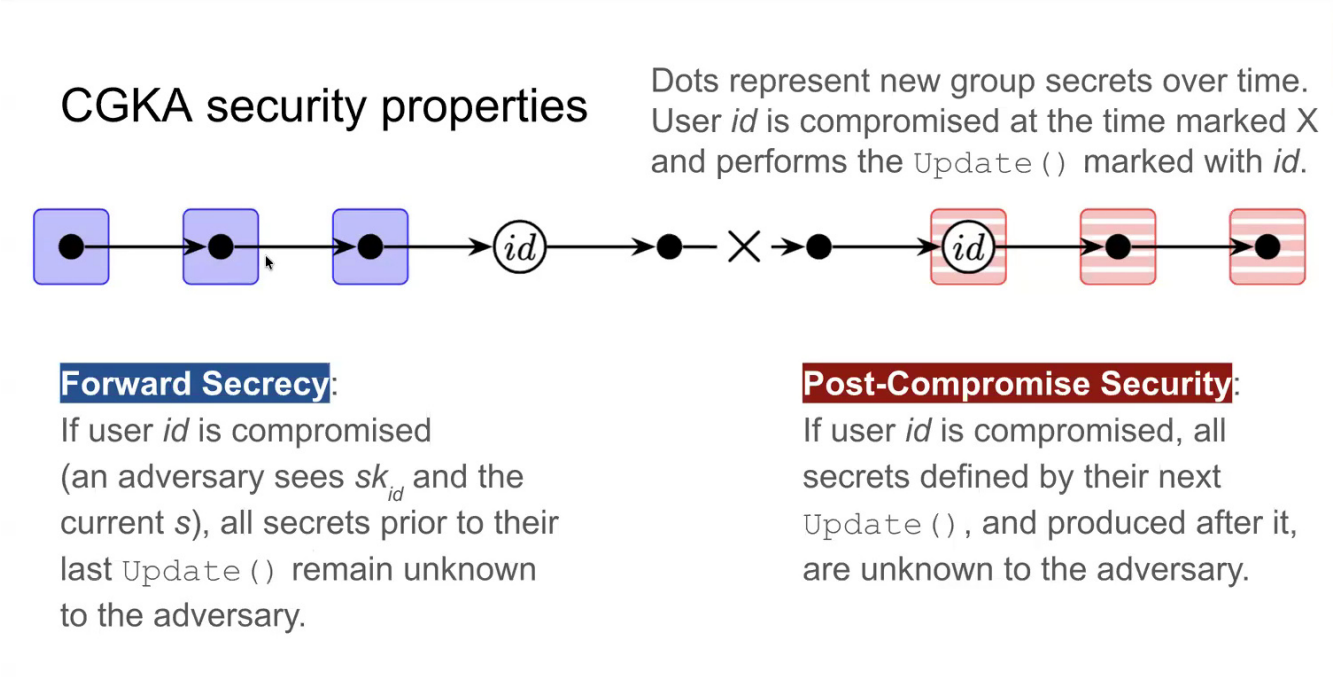
接下来谈谈我们希望 CGKA 具备哪些安全性质。这与前面所说的、随时间定义一系列密钥有关。直观上，既然组成员会变化，秘密值也必须变化。例如，把 Bob 移出组时，就必须把组秘密从 s_6 换成其他值；如果继续使用 s_6，Bob 早已能够通过自己的私钥获取它。

00:07:28 • Derek Yen

因此，执行这些操作时，组秘密必须改变。类似地，当我们把新用户加入组时，也得让他能够以某种方式获取组秘密。随时间更换秘密还有第二个作用：增强安全性，抵御试图攻入组的攻击者。具体来说，假设有一个“快照攻击者”，他能够在某一时刻获得某个用户的秘密状态。

00:08:01 • Derek Yen

通过随时间更换组秘密所带来的安全性质，我们可以把这次入侵造成的损失限制在一定范围内：泄露一个秘密值，只会暴露部分历史消息，而不会暴露全部消息。面向未来，持续轮换秘密也让协议能够自我恢复。一次秘密泄露，不应让攻击者读到此后所有通信。



前向保密性与失陷后安全性 • 画面 00:09:45

00:08:30 • Derek Yen

我用这张图更具体地解释，同时给出这两个性质的正式名称：前向保密性和失陷后安全性。图中展示了秘密值随时间的演变，每个点代表一个版本的组秘密。箭头表示它们随时间变化，从左到右是时间前进的方向。

00:08:56 • Derek Yen

假设组里有一位标识为 id 的用户，在标有 × 的位置泄露了自己的状态，也就是有人以某种方式取得了他的秘密。正式地说，攻击者在这个时刻知道了该用户的私钥 sk_id，以及当前的组秘密 s。图中两个标有 id 的节点，表示由这名用户，也就是遭到入侵的用户自己执行的更新。

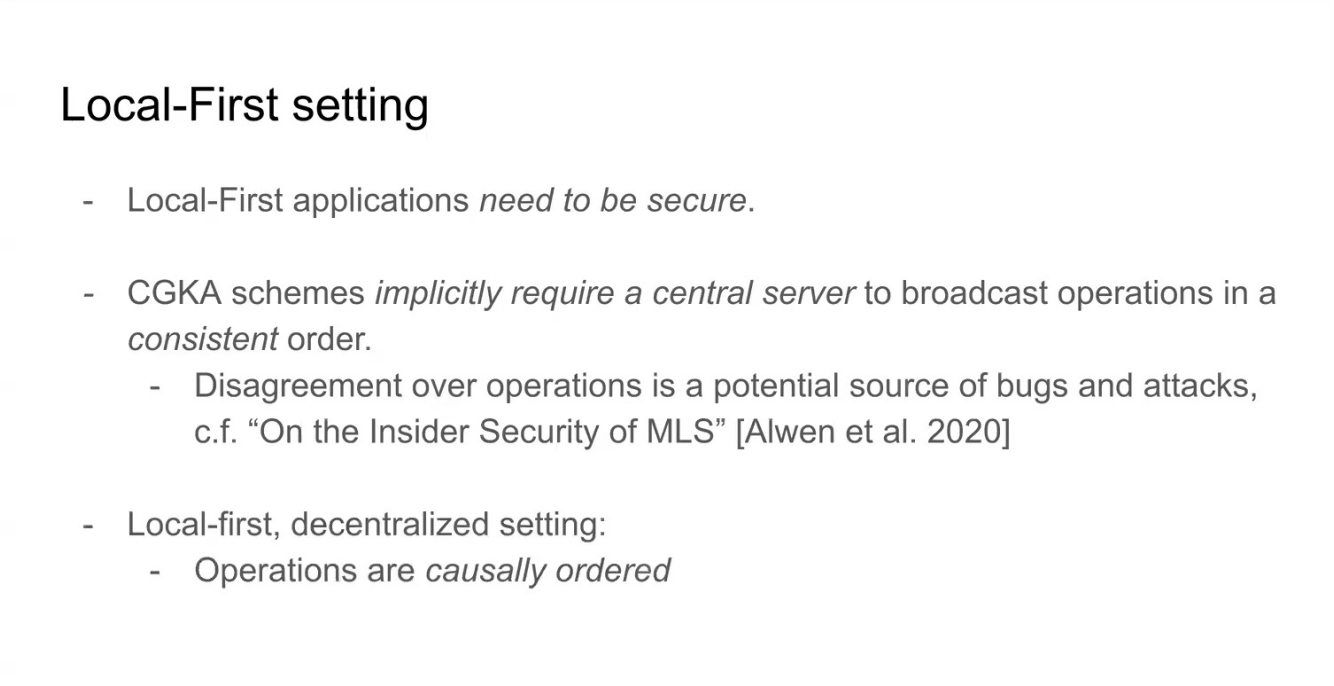
00:09:24 • Derek Yen

稍微正式地说，前向保密性要求：当攻击者攻破用户 id，看到他的私钥 sk_id 和当前组秘密 s 时，该用户上一次更新之前的所有组秘密，仍然应当对攻击者保密。图中蓝色背景标出的就是受前向保密性保护的组秘密。即使用户在 × 处遭到入侵，上一次更新之前定义的这些秘密仍然受到保护。

00:09:59 • Derek Yen

失陷后安全性则面向未来：用户 id 遭到入侵后，由他的下一次更新定义的组秘密，以及此后产生的组秘密，都应当不为攻击者所知。图中红色横纹背景标出的，是应当受到失陷后安全性保护的组秘密。还要注意，没有着色的那些组秘密不受这两种性质保护，攻击者可能获取它们。接下来，我会从隐含假设中心化环境的 CGKA，转向本地优先环境，看看有哪些变化。

端到端加密为什么仍依赖服务器



本地优先环境中的因果序 • 画面 00:11:35

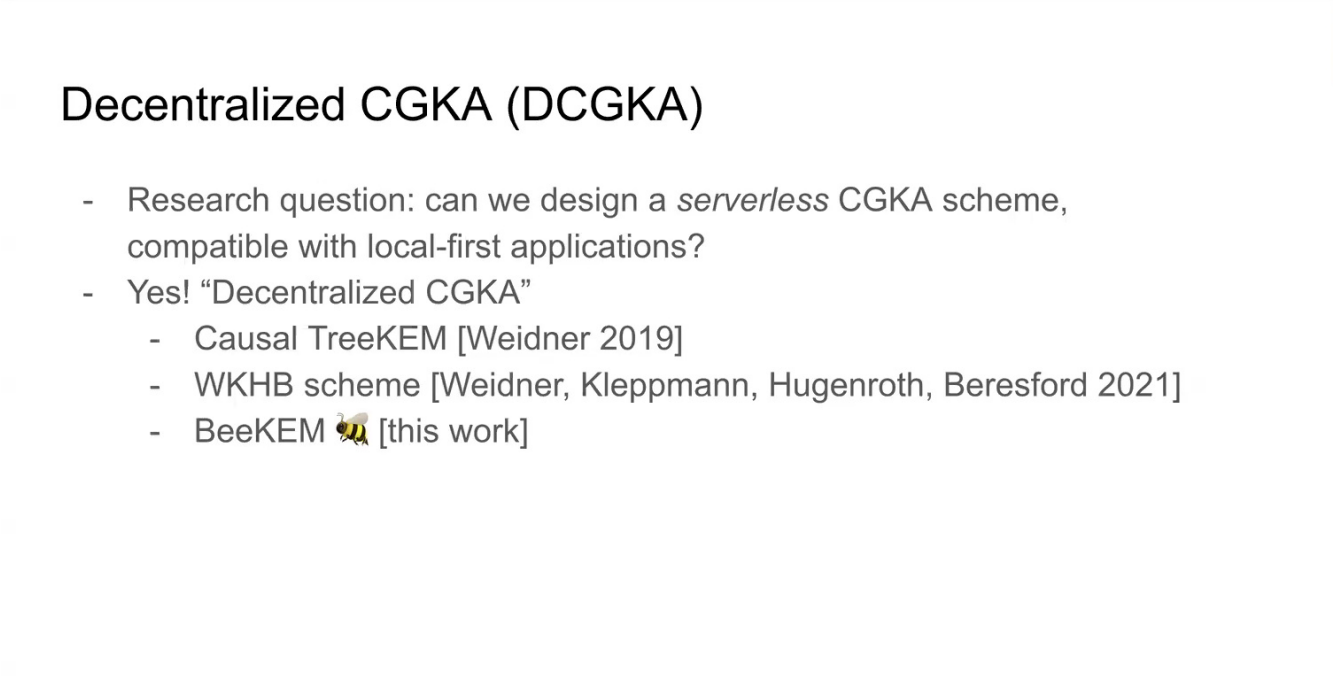
00:10:38 • Derek Yen

本地优先应用和其他应用一样，都需要保护用户数据。不过，这里有个问题：我们前面讨论的 CGKA 方案，隐含要求一个中央服务器以一致的顺序广播操作。端到端加密的理念是，中央服务器不应该读到通信内容；但这不等于协议可以在没有中央服务器的情况下运行。

00:11:10 • Derek Yen

中央服务器在操作排序中起着关键作用。如果参与者对操作的顺序存在分歧，或者方案允许出现这种分歧，就可能导致协议的正确性问题，甚至产生一些可行的攻击，已有研究讨论过这些问题。在本地优先的去中心化环境中，操作不再具有全序，而是按因果序组织。网络还可能发生分区，用户也可能长时间离线。

从 CGKA 到去中心化的 DCGKA



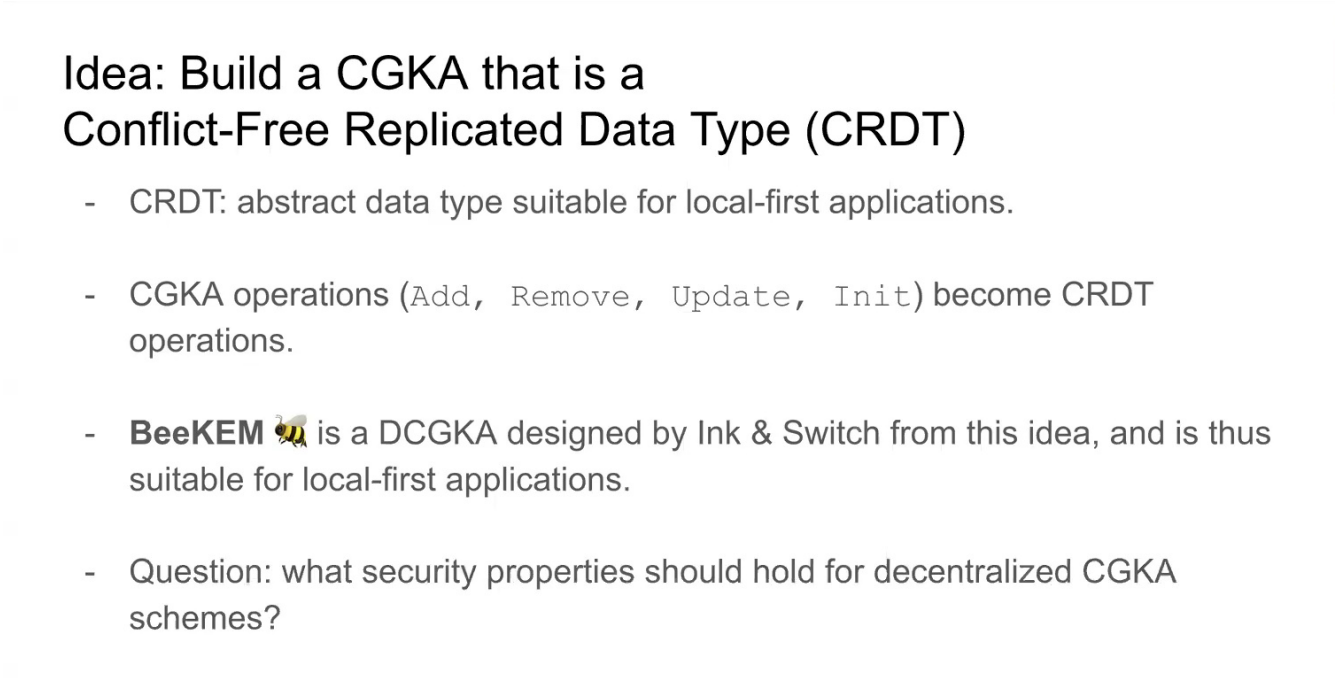
去中心化组密钥协商的研究脉络 · 画面 00:12:45

00:11:45 · Derek Yen

所以，我们不能继续假设中心化环境中的许多便利条件都成立。一个自然的研究问题是：能否设计不依赖中央服务器的 CGKA 方案，使它适用于本地优先应用？答案是可以，这就是去中心化 CGKA，简称 DCGKA。我们知道的先前方案有两个：2019 年的 Causal TreeKEM，以及 2021 年提出、我们称为 WKHB 的方案；后者还形式化定义了 DCGKA。

00:12:22 · Derek Yen

据我们所知，今天介绍的 BeeKEM 只是第三个这样的协议。中心化环境中也有一些协议尝试以不同方式减少对服务器的依赖、提高稳健性。BeeKEM 是第一个真正去中心化的——不，应该说是第三个真正去中心化的方案，但它是第一个在某些方面达到最优渐近复杂度的方案。BeeKEM 的核心思路，是借鉴无冲突复制数据类型的设计。我看过这个系列前面的几场分享，里面已经提到 CRDT；不过，考虑到有些人可能没参加，我还是会简要回顾一下。



把 CGKA 构建为 CRDT · 画面 00:13:55

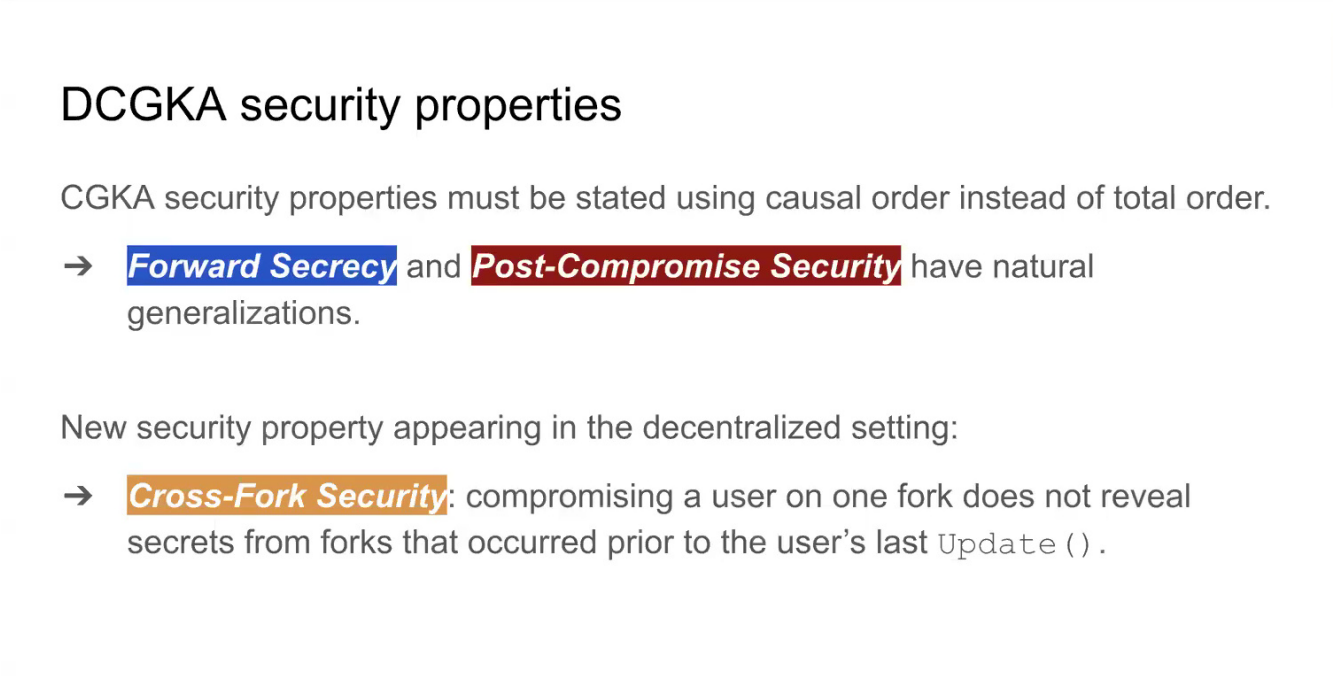
00:13:08 · Derek Yen

CRDT 是一种抽象数据类型，在很多方面适用于本地优先应用。它让多个副本共同维护一种数据结构，同时各个用户可以在本地使用自己的副本。在这里，我们把 CGKA 的操作作为 CRDT 的操作，因此前面那四种基本 CGKA 操作，也就是这个 CRDT 使用的操作。

00:13:43 · Derek Yen

BeeKEM 正是按这个思路设计的：构建一个本身就是 CRDT 的 CGKA 方案，而且从一开始就面向本地优先应用。设计密码学协议时，一个自然的问题是：去中心化 CGKA 应具备哪些安全性质？我们希望保留中心化 CGKA 通常要求的前向保密性和失陷后安全性，但表述需要稍作调整。因为这些性质用到了某次更新“之前”和“之后”的概念，而现在操作之间已经没有全序了。

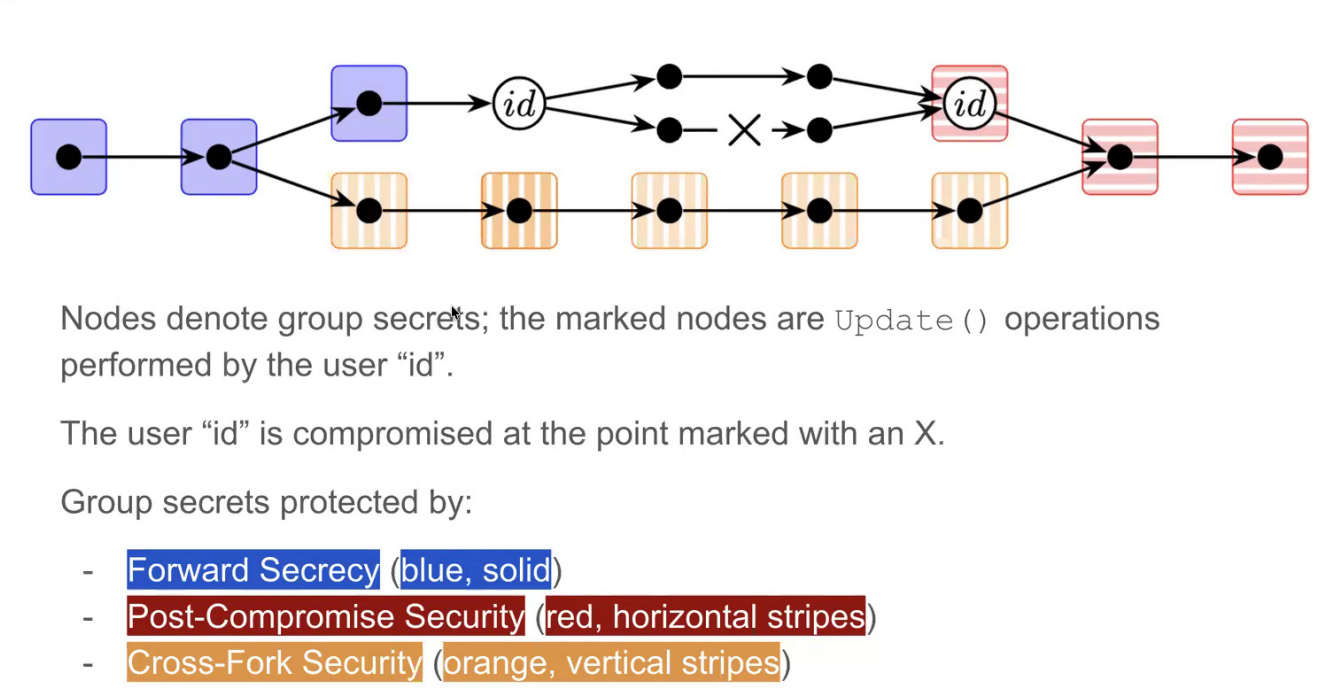
跨分支安全性



跨分支安全性的定义 · 画面 00:14:50

00:14:20 · Derek Yen

不过，把全序换成因果序，就能自然地推广这两个性质。去中心化环境还会出现一种新的安全性质，因为并发操作可能形成分支。我们把针对这些分支的性质称为“跨分支安全性”（cross-fork security）。不太正式地说，在一个分支上攻破某个用户，不应该暴露那些在该用户上一次更新之前就已分出的其他分支上的秘密。



三种安全性质保护哪些组秘密 · 画面 00:15:45

00:14:48 · Derek Yen

为了说得更具体一些，我再用一张图。这和之前的图一样，但现在也把它看作操作图：每个节点可以有多个后继。严格来说，实现中记录的是前驱，不过这里把箭头反过来画，更便于理解。和之前一样，每个点代表 DCGKA 方案中的一个组秘密版本。

00:15:19 · Derek Yen

两个标有 id 的点，代表某位用户发起的更新。该用户在 × 处把自己的状态泄露给了攻击者。和之前一样，蓝色实心框标出的组秘密受到前向保密性保护，红色横纹框标出的组秘密受到失陷后安全性保护。除此之外，橙色竖纹背景标出的组秘密，还受到跨分支安全性保护。

BeeKEM 的树与操作历史

00:15:50 · Derek Yen

也和之前一样，有些组秘密不受这些性质中的任何一种保护。特别是，图中在用户 id 上一次更新之后才出现的那个分支，并不受跨分支安全性保护。现在介绍了 CGKA 和 DCGKA 的基本概念，我来概括说明 BeeKEM 的构造。我们的方案要求用户在设备上维护两个对象。

00:16:22 · Derek Yen

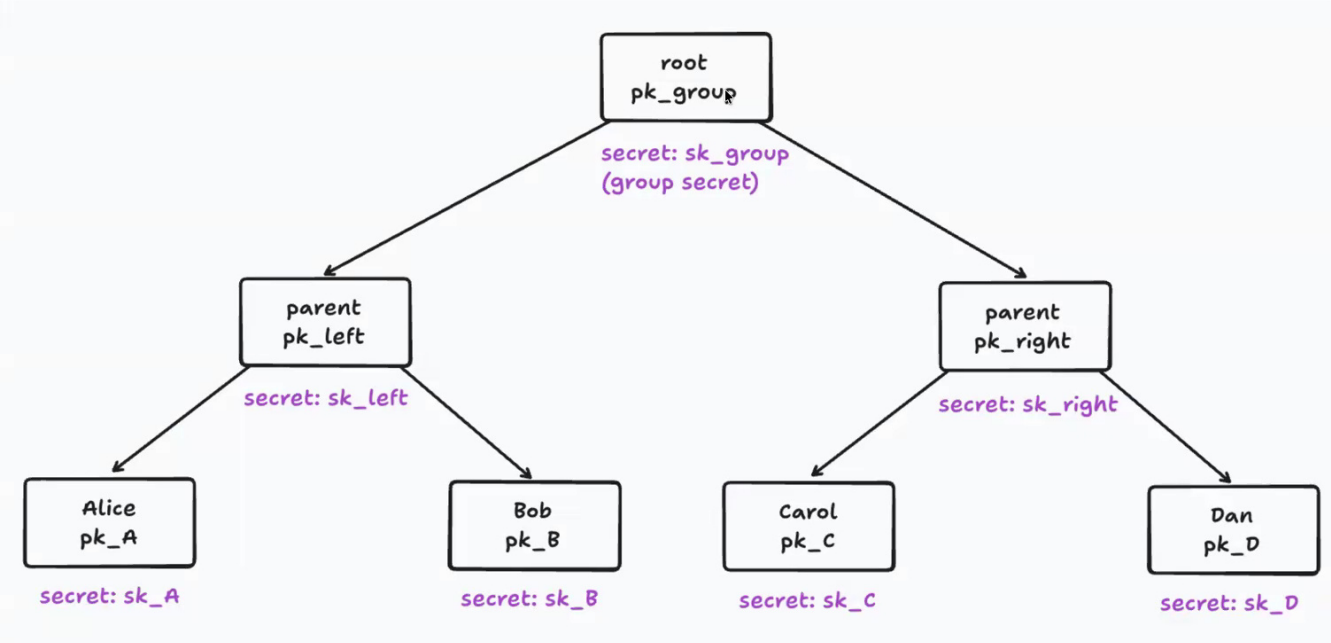
第一个是树，第二个是操作历史图。操作历史图的用法类似于通常的 CRDT：用户记录对底层数据结构的所有操作，而树就是这里真正要维护的数据结构。树既编码了当前的组成员，也让用户能够计算当前的组秘密。CGKA 的各项操作会以不同方式修改树；用户则维护完整的操作历史，包含自己执行过的操作，以及从其他人那里收到的操作。

00:16:59 • Derek Yen

每当用户执行新操作，就会通过例如 gossip 网络这样的方式，向其他人传播它。前面的 CRDT 分享可能已经介绍过操作图，因此这里多讲一点 BeeKEM 的树结构。每位组成员都对应树上的一个叶节点，叶节点标注了该用户的公钥，比如 pk_A。

00:17:25 • Derek Yen

树的每个内部节点也有对应的公钥和私钥。我们希望维持一个不变量：只要拥有某个子节点的私钥，就能够获取其父节点的私钥。如果能始终维持这个不变量，就能直接得到一个结论：所有用户都可以获取树根处的秘密。



四名成员组成的 BeeKEM 树 • 画面 00:18:15

00:17:50 • Derek Yen

因此，我们把树根处的秘密作为组秘密，因为所有叶节点都能获取它。这张图给出了一个包含四名用户的 BeeKEM 树：四个叶节点分别对应 Alice、Bob、Carol 和 Dan。黑色表示随树公开的信息；紫色表示秘密信息，它们不一定对所有人可见。

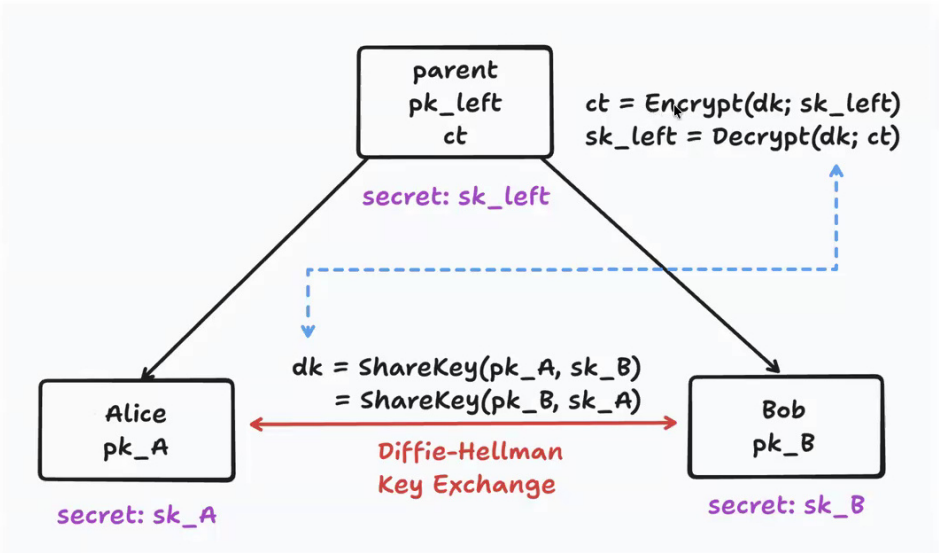
00:18:20 • Derek Yen

例如，Carol 的节点只公开 pk_C，而她把对应的私钥 sk_C 留在自己手里。左右两个内部节点分别有公钥 pk_Left、pk_Right，以及各自的秘密值。树根有公钥 pk_Group 和秘密值 sk_Group，后者就是组秘密。下面具体说说如何维持这个不变量，我把它再次列在了图的下方。

用 Diffie–Hellman 逐层解开秘密

00:18:54 • Derek Yen

内部节点的私钥以密文的形式存储，用另一把密钥加密，但要保证持有任意一个子节点秘密值的人都能解开它。具体来说，我们在两个子节点之间使用两方密钥交换协议，得到一把用于解密父节点秘密值的密钥。这里涉及几个环节，接下来用图说明。



Invariant: if you have a child's secret key, you can get the parent's secret key.
- **Corollary**: all users can access the secret at the tree root.

通过子节点密钥解密父节点 • 画面 00:19:45

00:19:20 • Derek Yen

这张图只保留了刚才大树的左子树，我们先看一个节点上发生了什么，再把同样的过程沿树向上递归。假设我们是 Alice，手里只有 sk_A。父节点上存储着它的公钥，以及一个密文 ct；这个密文加密的正是父节点的私钥 sk_Left。

00:19:48 • Derek Yen

加密它用的密钥，我记为 dk，意思是派生密钥。dk 是 Alice 与兄弟节点运行密钥交换后能够得到的另一个秘密值。如果你之前不了解 Diffie–Hellman，在这里只需要知道：它让你把对方的公钥和自己的私钥结合起来，计算出一个共享秘密；关键是，对方也能对称地计算出同一个值。

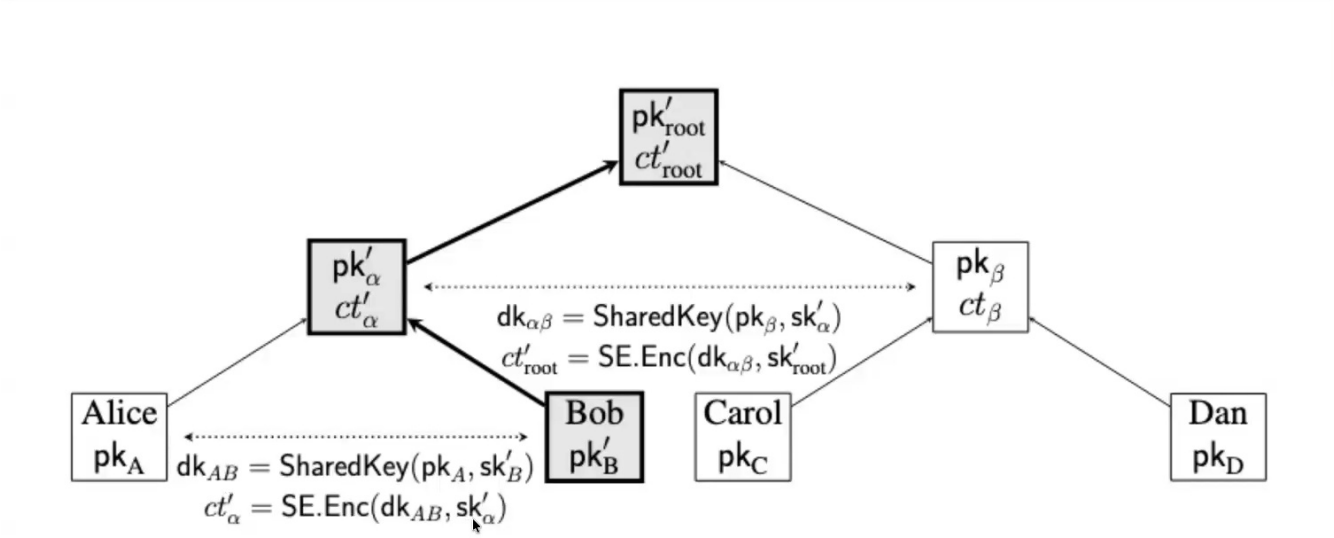
00:20:20 • Derek Yen

图中写出的关系是：dk = ShareKey(pk_A, sk_B) = ShareKey(pk_B, sk_A)。Alice 持有 sk_A，也能看到 Bob 的 pk_B，所以可以计算第二个表达式。Bob 则可以用 sk_B 和 Alice 的公钥 pk_A 得到同一个 dk。这是某些密钥交换算法的特殊性质；BeeKEM 具体使用的是 Diffie–Hellman。

00:20:49 • Derek Yen

因此，Alice 用 sk_A 得到 dk，再用 dk 解开密文，获得父节点的秘密值 sk_Left。树的每一层都有这种结构，于是我们就能维持“子节点的私钥可以导出父节点私钥”这个不变量。实际使用时，沿树逐层向上，就能获取根部的秘密。最后再介绍一下这个方案中的更新操作，因为你可能也会问：为什么要采用这种结构？

一次更新怎样沿树传播



Bob 发起更新时的密钥与密文 • 画面 00:22:25

00:21:22 • Derek Yen

为什么要把这些秘密放在密文里？当用户调用 update 时，会从自己的叶节点一直到根，在沿途每一层生成一对新的公私钥。图中展示的是 Bob 发起更新的情况。他先更新自己的节点，生成新的公私钥对，不过只把新公钥 pk_B′ 公开。

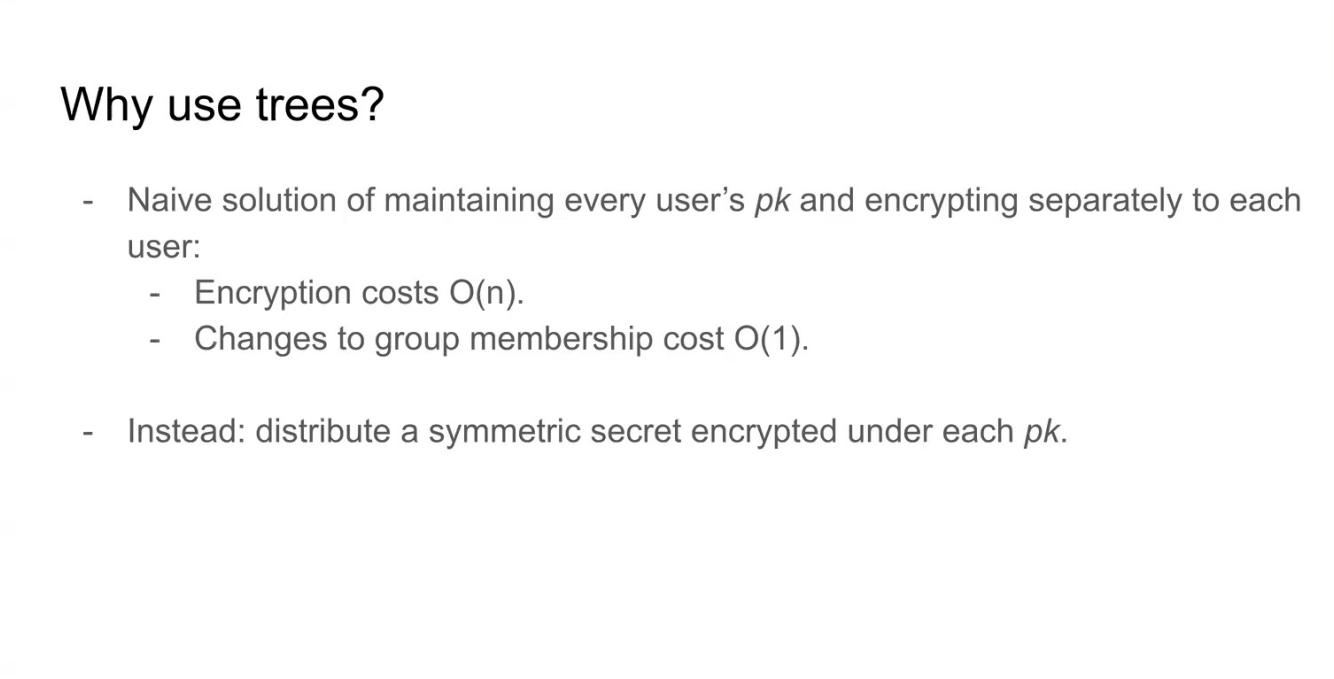
00:21:53 • Derek Yen

然后，他沿树向上，在父节点生成一对新的公私钥，公钥记为 pk_α′。接下来，需要把新私钥 sk_α′ 编码成 Alice 也能获取的形式：Alice 必须还能凭自己的私钥 sk_A 访问它，才能维持前面的不变量。具体来说，Bob 使用 Diffie–Hellman 的共享密钥算法计算 dk_AB，再用 dk_AB 加密 sk_α′，生成新密文，放到这个父节点上。

00:22:30 • Derek Yen

再往上一层，重复同样的结构：在根部生成新的秘密值 sk_root′，也就是新的组秘密，然后用派生密钥 dk_αβ 加密它。这个派生密钥来自左侧内部节点的新私钥 sk_α′ 与另一侧内部节点的公钥 pk_β。由这种结构，也可以看出为什么这些操作能够达到对数级成本，不过这里还有几个限定条件。

为什么使用树结构



树结构中的加密与成员变更成本 · 画面 00:23:55

00:23:08 · Derek Yen

顺便解释一下，为什么要用树。设想一种朴素的 CGKA 方案：每位成员公开一个公钥，发送时分别为每个用户加密。这样加密效率不高，成本是 O(n)，因为要为 n 名用户分别加密。但另一方面，修改组成员却非常高效。

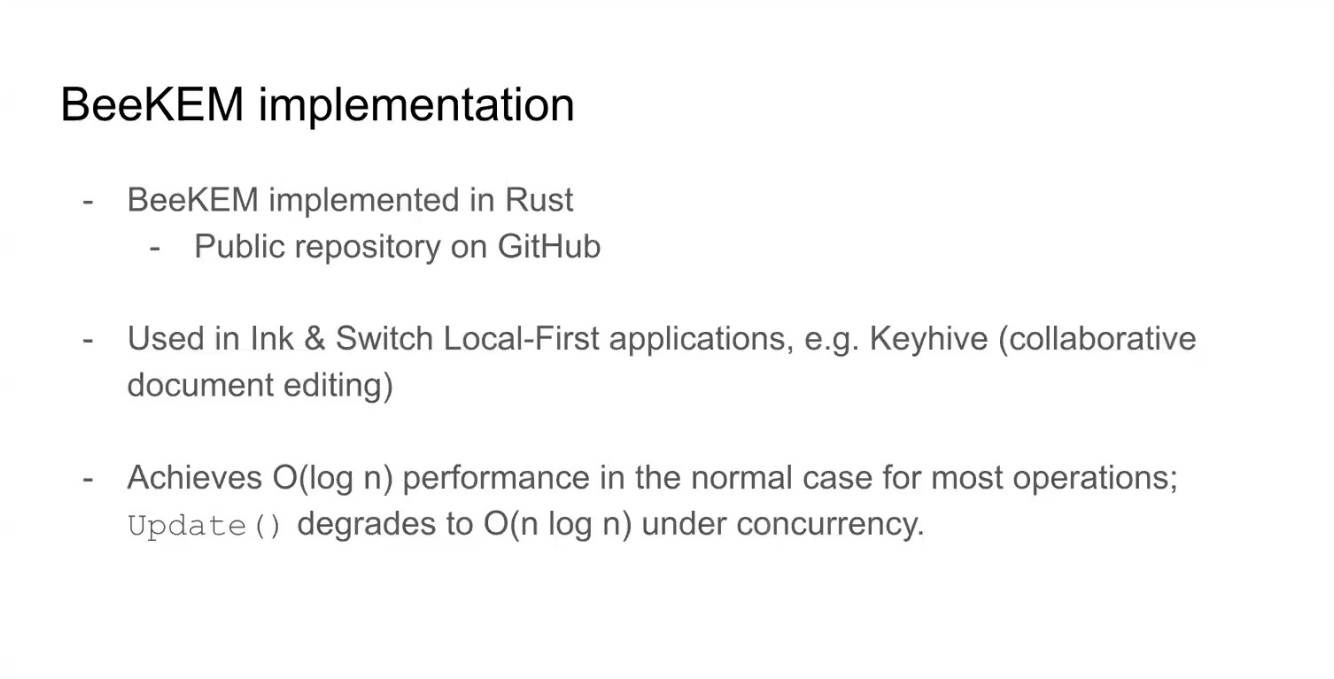
00:23:33 · Derek Yen

成员变更的成本只有 O(1)。用户更新公私钥对时，也只需要发布一条消息。如果想改进加密效率，可以分发一个对称密钥，用它加密消息，再分别用每位用户的公钥加密这个对称密钥。这基本上是一棵扁平的树。这样，为整个组加密消息的成本就是 O(1)。

00:24:01 · Derek Yen

但问题是，变更组成员或组秘密现在需要 O(n) 的成本，因为必须针对每个现有公钥重新加密新的组秘密。添加或移除用户时，也要轮换组秘密。直观上，BeeKEM 或更早的 TreeKEM 这类基于树的 CGKA，是把这个折中方案组织得更好：消息加密仍是 O(1)，而通过改变结构，让 CGKA 操作达到对数级复杂度。

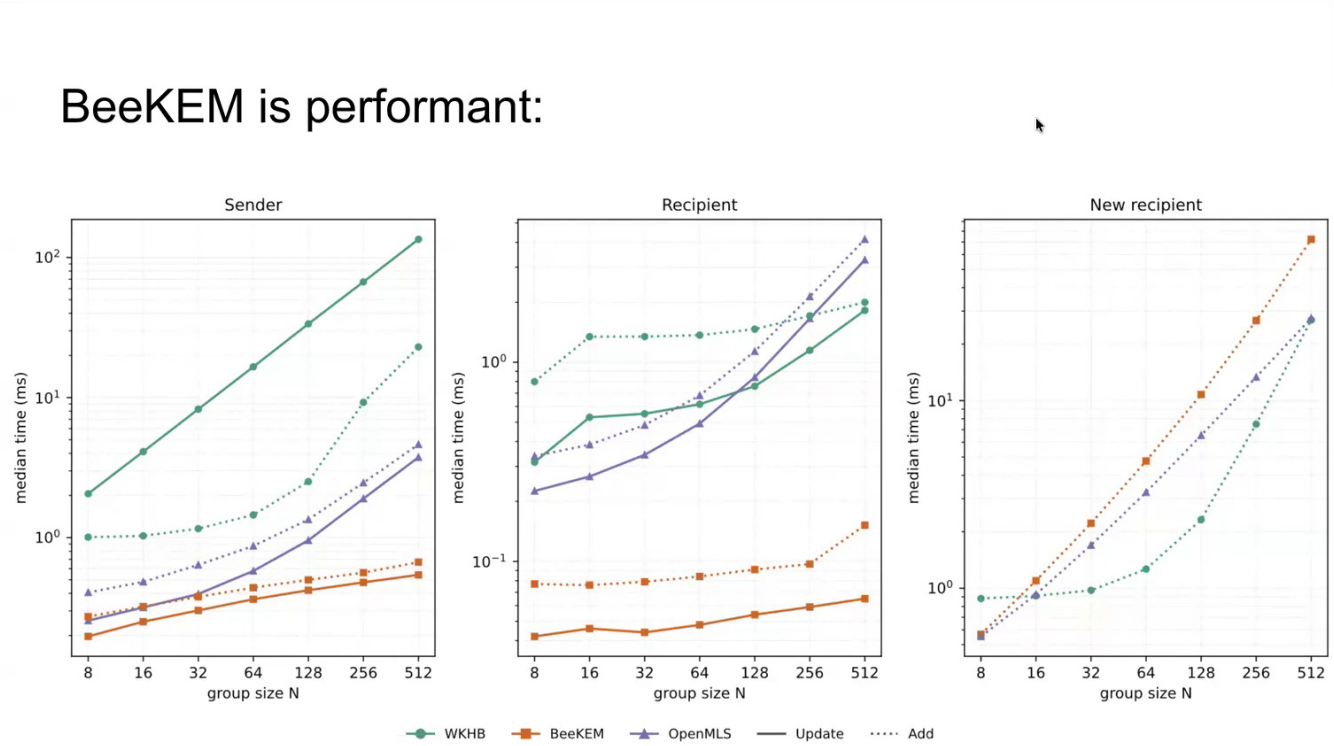
Rust 实现与性能比较



Rust 实现及并发更新的成本 • 画面 00:24:55

00:24:37 • Derek Yen

最后谈谈实现。我们用 Rust 实现了 BeeKEM，代码公开在 GitHub 上。Ink & Switch 的一些本地优先应用已经在使用它，比如协作文档编辑应用 Keyhive。BeeKEM 在通常情况下具有对数级的渐近性能。不过，尤其是 update，在存在并发时，有时成本会提高到 O(n log n)。



BeeKEM、WKHB 与 OpenMLS 性能比较 • 画面 00:26:05

00:25:08 • Derek Yen

这部分细节比较复杂，可以会后再谈；这就是我前面提到的限定条件。接下来看看性能数据。这里有三张图，纵轴是执行操作所需时间的中位数，横轴是组大小，借此观察 BeeKEM 的渐近表现。图中比较的是 CGKA 的两种操作：update 和 add。

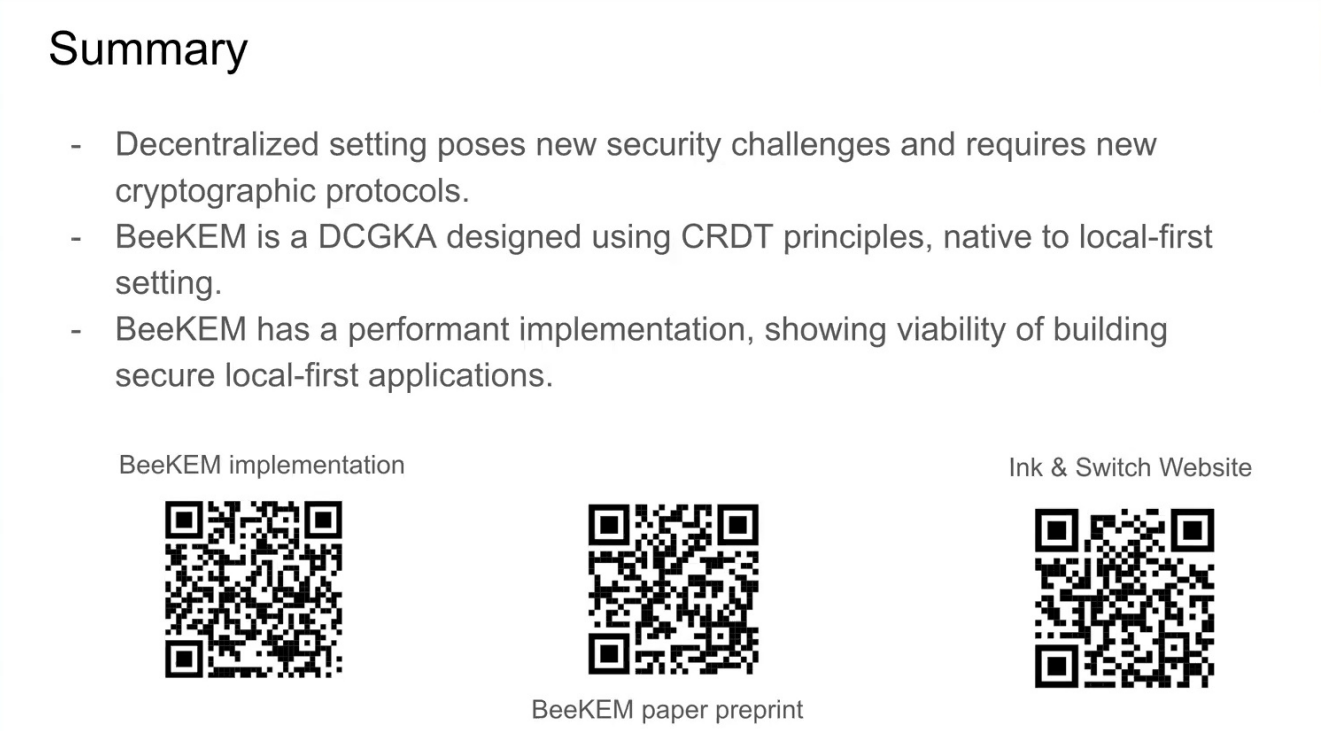
00:25:39 • Derek Yen

实线表示 update，虚线表示 add。我们把 BeeKEM 与另外两种代表性方案作比较：带方形标记的橙色线是 BeeKEM；绿色线是 WKHB，它是此前先进的 DCGKA 方案之一；紫色线是 OpenMLS，也就是 MLS 协议的一个实现，其中的 CGKA 是 TreeKEM，它可能是目前最常用的中心化 CGKA 方案。从图中可以看到，在大多数测试范围内，BeeKEM 的性能与另外两种方案相当，甚至更好。

00:26:26 • Derek Yen

左图和中图分别是发送方与接收方的成本，两者需要执行的操作不同，所以成本也不同。右图是新接收方加入组的成本，这里 BeeKEM 反而稍差一些，原因也不难理解：CRDT 的设计要求存储更多元数据。总结一下，去中心化环境带来了新的安全挑战，也需要新的密码学协议。

研究小结与项目资源



代码、预印本与研究团队 · 画面 00:27:30

00:26:56 · Derek Yen

大多数密码学协议隐含地只考虑中心化环境，因此不能直接用于本地优先应用。我们的 DCGKA 方案 BeeKEM 按照无冲突复制数据类型的原则设计，因而原生适用于本地优先环境。它有一个性能良好的实现，展示了构建安全本地优先应用的可行性。最后是几个链接：左侧二维码指向 GitHub 上的 BeeKEM 实现。

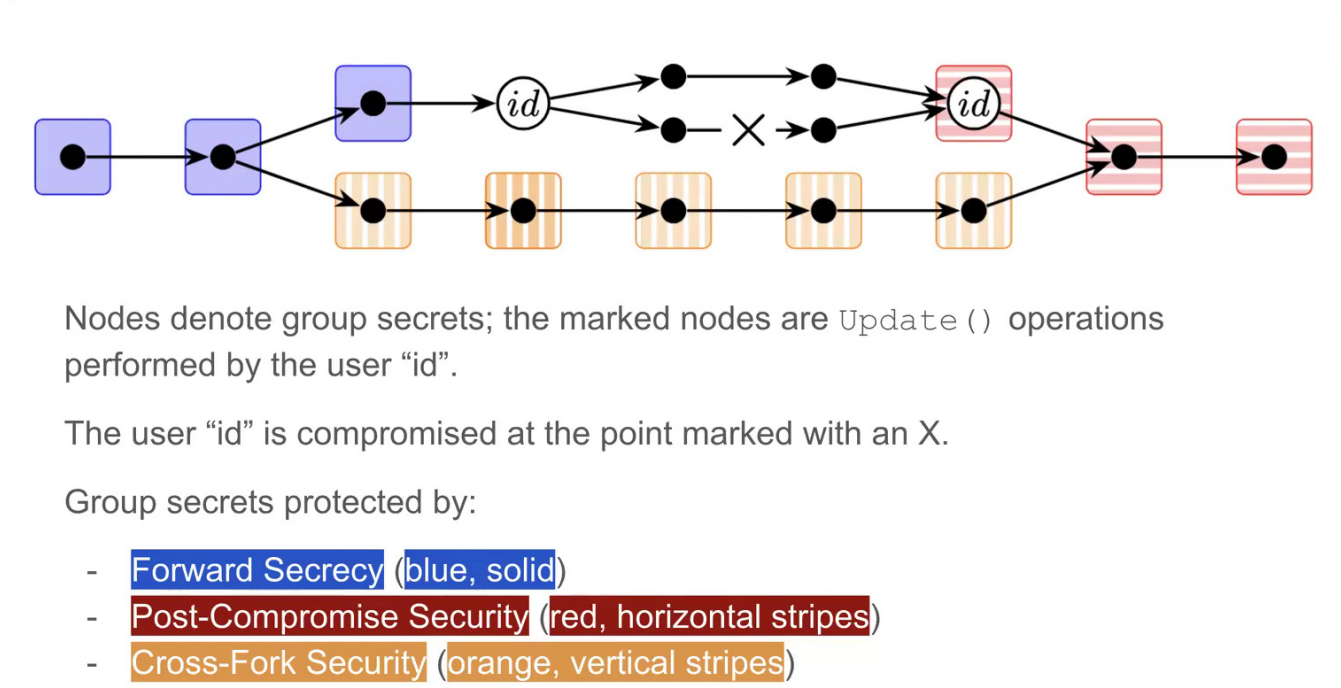
00:27:26 · Derek Yen

中间是我们的论文预印本，介绍 BeeKEM 的算法、安全性证明和性能评估，目前正在投稿。右侧是 Ink & Switch 的网站，这是一个在本地优先领域做了很多工作的产业研究团队。如果之前没有听说过他们，很值得了解。谢谢大家，接下来可以提问。

问答：跨分支攻击是什么样

00:27:48 · 主持人

你刚才介绍了跨分支安全性。能否举一个具体例子，说明跨分支攻击会是什么样？如果协议不具备这个性质，攻击者实际上可以做什么？



问答中使用的安全性质示意图 · 画面 00:28:51

00:28:08 · Derek Yen

这是个很好的问题。我举一种攻击。BeeKEM 实现了一种跨分支安全性，而接下来这种攻击可以针对前两个 DCGKA 方案，也就是 Causal TreeKEM 和 WKHB。它们的工作方式差异很大，所以我会用比较概括的方式描述。假设 Alice 和 Bob 都是一个大组的成员，他们并发地各自发布了一次更新。于是出现两个新的组秘密，一个由 Alice 定义，另一个由 Bob 定义。

00:28:54 · Derek Yen

在 WKHB 和 Causal TreeKEM 这两个先前方案中，如果攻击者在这条分支上攻破 Alice，获取了组秘密，他实际拿到的状态，还可能被用来推导或计算 Bob 在另一条分支上的新秘密。我说得比较概括，因为在两个方案中，具体的攻击细节有所不同。但这个例子在技术上并不属于前向保密性或失陷后安全性的范围，因为这两个新秘密是并发产生的。

00:29:29 · Derek Yen

它们在因果序上没有先后关系，所以严格来说，前向保密性和失陷后安全性都不能直接适用。更广泛地说，跨分支安全性想表达什么？我想，最自然的说法是：在存在并发的情况下，我们也希望对其他分支作出某种安全保证，而这样的保证，并不能直接从传统的那两种安全性质推出。

00:30:13 · Derek Yen

我看看。

00:30:15 · 主持人

还有一个问题。

问答：安全性与长期离线的取舍

00:30:19 · Derek Yen

好，我把问题读出来，方便录音："你认为 DCGKA 最大的开放问题是什么？这个领域接下来应该往哪里走？"这是个好问题。有一点我刚才没有展开：如果去看 BeeKEM 的预印本，会发现我们实际实现的安全性质，比刚才在这里描述的要稍弱一些。我前面介绍的三种性质，是理想的安全性质。

00:30:51 · Derek Yen

最理想的 DCGKA 方案会具备这三个性质。但 BeeKEM 实际达到的安全性，由一个在代码中记为 kappa 的参数调节。这个参数体现了安全性与我们所说的"对并发的耐受能力"之间的取舍。先退一步说：对于 DCGKA，或者更广泛的去中心化、本地优先方案，你可能还希望它能容忍长期网络分区和长时间离线。

00:31:30 · Derek Yen

比如，我是群聊中的 Alice，去森林里露营度假三周，期间完全不碰手机。回来以后，我仍然希望能解密那些原本发给我的消息。如果应用加密了大量数据，等我回来之后，再把所有数据重新加密发给我，可能并不可行。

00:31:57 · Derek Yen

所以，我需要一种办法，能够取得离开那三周期间的组秘密。要实现这一点，Alice 就需要保留自己个人秘密值的许多旧版本，其中可能有很早以前的版本。这会稍微削弱安全性：如果 Alice 的设备被攻破，攻击者就能获得大量信息，包括许多过去的组秘密。kappa 描述的就是安全性与容忍网络分区、长时间离线的能力之间的取舍。

00:32:34 · Derek Yen

回到你的问题，我们在这项工作中指出，至少在 BeeKEM 里，并发情况下的可用性与安全性之间存在这种取舍。理想的方案不需要这样权衡。我们还想进一步研究，这种取舍究竟是这个环境固有的，还是存在某些办法可以克服它。

00:32:57 · Derek Yen

我们还定义了 BeeKEM 的一个变体，选择实现更强的安全性，但代价是对并发的容忍能力较差。总的来说，我认为需要更仔细地思考这两种相互牵制的性质，尤其是在本地优先环境中。我们确实希望有很强的安全性，但本地优先也有容忍网络分区的实际工程需求。至少按我们目前的理解，这两者之间存在一些张力。

00:33:30 · Derek Yen

所以，我认为这是未来研究中很重要的一个问题。这样回答能理解吗？刚才可能说得有点长。

问答：安全状态能否建模为 CRDT

00:33:53 · Derek Yen

我再读一个问题，方便视频里的观众理解："BeeKEM 和 CRDT 都需要处理并发操作，并在之后合并它们。你觉得安全状态本身，是否也可以更像复制数据类型那样来建模？"说实话，这个问题有些超出我的专业范围。

00:34:14 • Derek Yen

我主要从密码学这一侧参与，一直在学习 CRDT 和本地优先这些东西。所以，这个问题我还不太确定。

问答：BeeKEM 怎样诞生

00:34:25 • 主持人

还有一个问题：你们是怎样开始研究 BeeKEM 的？在这项工作中，你觉得最有意思、最有乐趣的部分是什么？

00:34:51 • Derek Yen

我刚才把 BeeKEM 当作一个独立项目来介绍，然后提到 Ink & Switch 的另一个应用 Keyhive 用了它。不过，实际发生的顺序并不是这样。在 Ink & Switch，我们先开发了 Keyhive，为了做这个协作文档编辑器，才开发出这个 DCGKA 方案。所以严格来说，是先有 Keyhive。这就是 BeeKEM 实际诞生的过程。据我理解，设计理念确实来自 CRDT。

00:35:26 • Derek Yen

这也是核心思路。广义上，BeeKEM 属于名为 ART，也就是异步棘轮树（Asynchronous Ratcheting Tree）的 CGKA 协议家族。最常用的中心化 CGKA 方案 TreeKEM 也来自这个家族。这大致是 BeeKEM 的技术渊源。另外，第一个 DCGKA 方案 Causal TreeKEM，也对 BeeKEM 有很深的影响，可以说是它的重要前身。抱歉，第二个问题是什么来着？

00:36:07 • 主持人

在这项工作里，你觉得哪些部分最有意思、最让你兴奋？

问答：密码学研究者的新发现

00:36:16 • Derek Yen

对我个人来说，正如前面提到的，我是作为密码学研究者进入这个领域的，本地优先对我来说是新东西。很有意思的一点是，我发现不少传统密码学技术与这个环境并不适配，或者不能直接用在本地优先环境中。比如，端到端加密通常被描述为服务器什么都看不到，那为什么我们还需要服务器？

00:36:49 • Derek Yen

最初了解 BeeKEM，并和 Ink & Switch 的合作者一起做安全性证明时，我才意识到，自己以前并没有充分认识到端到端加密对中央服务器的依赖。对一个密码学研究者来说，发现端到端加密仍然存在这种依赖，让我很惊讶。这也说明，密码学界去研究服务器到底承担什么角色，是很重要的。

00:37:27 • Derek Yen

一直以来，也有一条研究路线尝试对服务器建模，识别我们对服务器作出了哪些性质假设，并挑战这些假设。但我认为，本地优先所探索的、完全移除服务器的方向，研究还相对不足。对我来说，这是个很有意思的问题。

问答：文档历史与前向保密性

00:37:47 • Derek Yen

我再读一下聊天里的另一个问题：“文档似乎与聊天不同。在文档场景中，我们似乎需要放弃前向保密性。是这样吗？”

00:37:56 • Derek Yen

这个问题我和 Brooke 讨论过几次。它确实很好。我认为，这取决于具体应用。我不愿意笼统地说，文档就应该放弃前向保密性。但在大多数常见的合理场景中，确实如此，前向保密性可能没有那么适合。为了让看视频的朋友理解这个问题，我解释一下：协作文档编辑中，一个很自然的需求，是查看文档以前的版本。

00:38:33 • Derek Yen

至少，你需要看到文档的当前状态，而当前状态是由过去的版本演变而来的。但通常，你还希望看到比当前状态更细的历史。从形式上说，作为新加入这份文档的成员，如果想看到过去每一次具体的编辑，就需要取得加密那些历史编辑所用的密钥。

00:38:56 • Derek Yen

换句话说，新加入组的用户要取得旧版本的组秘密，就需要牺牲前向保密性。因此，如果你的要求是让新用户看到组的完整历史，那么是的，就得放弃前向保密性。但也可以选择另一种设计：保留前向保密性，而不让新用户看到组的历史。

00:39:22 • Derek Yen

有些应用可能更愿意选择后一种取舍。如果允许我代 Brooke 转述，我想 Brooke 和其他一些参与 Keyhive 的人的看法是：让用户能够查看文档的历史版本，比前向保密性更重要。就 Keyhive 本身而言，放弃前向保密性还有其他原因，但那些是 Keyhive 特有的。

00:39:51 • Derek Yen

这些问题都很好。

问答：用 BeeKEM 做聊天应用

00:39:54 • Derek Yen

“如果有人用 BeeKEM 做聊天应用，能做出哪些有意思、好玩的功能？”

00:40:00 • Derek Yen

我可能不是最适合回答这个问题的人。总体来说，聊天作为去中心化应用，有意思的一点是：相比其他应用，消息通信更能接受去中心化环境的一些限制。比如，去中心化环境的重要挑战是网络分区和并发。很多应用一旦有人离线一段时间，就没办法正常工作了。

00:40:45 • Derek Yen

我试着举个例子，想想哪个比较合适。假如某种应用要求每个发出的数据包都必须被接收，任何一个都不能丢，而且所有人都要收到，例如某种大型联合计算，那么它在本地优先环境中就比较难做。因为一次网络分区可能让计算彻底失败，必须重新开始。

00:41:24 • Derek Yen

这类事情在中心化环境中做起来会自然得多。消息通信的好处在于，它有某种“语义上的韧性”。比如，我给别人发消息，其中一条丢了——这也许不是个好例子——但至少可能察觉到消息缺失，因为我说的话突然接不上了。所以，聊天相对来说更自然一些。

00:41:51 • Derek Yen

不过，我还不确定怎样给这个问题一个一般性的答案。我想，目标可能是让 BeeKEM 支持你像在中心化环境中一样，构建各种聊天应用，尽管具体的安全性和其他性质会略有不同。抱歉，这可能不算一个很好的回答。