Designing a Secret Handshake: Authenticated Key Exchange as a Capability System

摘要

基于能力的安全性（Capability Based Security）是设计分散式访问控制系统的概念上的框架，目前还没有一个用于建立安全的双向通信并且同时也形成一个能力系统的广泛实现的协议。我们调查了各种密钥交换协议不能成为能力系统的原因，然后提出一种以能力系统设计的安全密钥交换协议。在该协议中，服务器的公共密钥形成一个访问能力。使用预认证步骤，我们在服务器之前验证客户端，但仍然在 4 次传递内完成相互验证。所有长期密钥都对任何未经认证的活动者保密。

1. Introduction

密钥交换算法的发展标志着现代密码学的开端[7]。他们的发展是由于双方之间的安全通信的需求而推动的，但是设计一种用于在两个认证方之间交换共享密钥的实用并且安全的协议是并不重要的[8]。

密钥交换的大部分研究已经产生了协议的整个“家族”[10]。目前广泛使用的协议往往是分层的而且可以配置的（TLS，SSH）。这并不利于应用程序开发人员，要获得对这种密码系统充分细微的理解需要相当大的研究。为开发人员提供更多的选择也就是为他们的应用程序增加严重的安全漏洞提供更多的机会。因此，最近的观念已经转向提供具有容易理解且使用安全的安全属性的简单的构造[3]。我们将这一理念应用于认证密钥交换的设计—秘密握手（secret handshake）。由于可以设计具有如此多的可能属性的密钥交换算法，我们采用能力系统的框架[14]，并允许其驱动这个设计。TahoeLAFS [15]是这个决定的灵感。我们发现目前没有可用的握手协议充分满足能力系统的需求。有趣的是，能力系统需要比其他可用协议提供的更高的隐私程度。

我们将描述各种密钥交换协议，并检查其设计不能形成能力系统的原因。我们将抽象地讨论这些协议，提供足够的细节来展示其加密属性。在某些情况下，这意味着忽略实际实现所需的详细信息，例如处理随机数。最后，我们将介绍我们基于能力的密钥交换协议。

2. Capability System

在能力系统中，访问资源的权利被授予能力持有者（cap）。基于能力的安全性早于现代密码学，但其概念清晰地映射到通过加密技术启用的分散式访问控制系统。[1]

举个例子，拥有解密文件的密钥意味着你有能力读取该文件。 这可以以分散的方式实现：加密的文件通过对等网络（peer to peer network）广泛分发，但是 read-cap 仅限于信任的几个网络。与访问控制列表（Access Control List, ACL）对比，文件保存在服务器上，只有那些可以正确验证过的才能访问该文件：就像一个夜店的保镖一样，需要一个受信任的门卫来管理访问列表。一个分散式的 ACL 是不可能的。

[1] There are limitations to a cryptographic capability system. It is possible to implement what *Capability Myths Demolished*[12] describes as the *Capabilities as Keys* model. In this model, actors may delegate via any channel they can write to. Contrast this with the *Object Capability* model, which incorporates a strict type system that can distinguish between data and capabilities, and thus prevent capabilities from being mistakenly written as data.

授权（Delegation）是通过给其他行动者发送一个 capability 来授予你的一个访问权限给他们的行为。在一个表现良好的能力系统中，除了通过授权之外，没有其他办法可以获得特定的访问权限。

意外授权（Unintended Delegation）—如果你把汽车钥匙给代客泊车的人，那就是授权。如果你把你的钥匙留在你的车里，然后它被盗了—这就是一个意外授权，或者称为误授权（misdelegation）。这是不一样的，这就好像有人强行进入你的车，并点着你的车。如果有人“偷”了你的车，因为你失误给了他们钥匙，这是你自己的错。不要把误授权加入协议本身是我们作为协议设计者的责任。[2]如果一个系统在协议级别存在意外授权，那么它不会是一个性能很好的能力系统。

通配符（Wildcard）—在某些纸牌游戏中，有一个指定的“万能牌”，可以当做任何其他牌一样使用。如果一个 token 启用了必须单独授权的一组权限，则该 token 就是通配符。我知道没有故意设计包含通配符的密码系统。在能力系统中意外地存在通配符应该被认为是设计失败。

性能良好（Well Behaved）—如果一个能力系统支持授权，但不存在意外授权和通配符，则该能力系统性能良好。

3. Notation

A → B 表示从客户端传递到服务器的消息，A ← B 表示一个响应。了解一方在什么时候认证是非常重要的，以及该方也要了解这一点。如果 A 或 B 被替换为?，则意味着对于接收者来说，该方还没有被认证。如果 Alice 收到一个消息? ← B，那意味着她现在知道 Bob 是谁，但是 Bob 不知道她是谁。这个握手直到双方知道他们在对谁说话，并且知道对方也知道的时候，才完整。因此，一次握手直到两次传递分别标记为 A → B 和 A ← B 的时候才会结束。

当一个密钥对是一个函数的参数时，它则被表示为下标的形式。当一个密钥在一个消息中被发送时，只有公共密钥被使用，为了使其明确，我把它写成 A_p。本文中研究的协议都没有通过线路发送密钥。

Alice 和 Bob 的长期密钥由 A，B 表示，短期密钥由 a，b 表示。a • b 表示从 a 和 b 导出的共享密钥。再次，当这是一个函数参数时，它被表示为下标的形式，$Box_{[a \cdot b]}(msg)$。请注意，Box 是经过身份验证的加密，它具有 mac 以及加密属性。|表示串联。

最后，每当接收到消息时，它立即被验证，如果无效，握手被中止。

4. Prior Art

4.1. Authenticated Key Exchange - STS, TLS, SSH

Station to Station[8]协议很简单，而且是几种流行协议的基础。[3]

协议 1（Station to Station）

---

[2] AES is an example of a protocol that has been shown to contain a misdelegation[2] (secure AES implementations are possible, but suffer from poor performance). A good protocol design cannot prevent a malicious implementer, but it can mitigate the effects of an incompetent one. Thus, a good protocol design at least prevents a malicious implementer from *feigning incompetence*[5]. Thankfully, maliciousness is not as common as incompetence.

[3] TLS (when used with ciphersuites that contain DHE or ECDHE), and SSH share their basic design with STS, but using a hmac to prove they know $a \cdot b$ instead of encryption.

$$
\begin{aligned}
? &\rightarrow ? : a_p \\
? &\leftarrow B : b_p, Box_{[a\cdot b]}(B_p|Sign_B(a_p|b_p)) \\
A &\rightarrow B : Box_{[a\cdot b]}(A_{pub}|Sign_A(a_p|b_p)) \\
A &\leftarrow B : Box_{[a\cdot b]}(okay)
\end{aligned}
$$

Alice 给 Bob 发了一个新的短期密钥，Bob 也创造了一个短期密钥，并且签字了两个密钥，然和和他的公钥一起把它们发送回去。Alice 然后把她的公钥和一个签名打包以证明她的身份。最后，Bob 打包一个标准消息，以表明他已经接受。[4]

任何一方都不能保证消息的新鲜度，除非它是对已知新值的加密回复，即他们新创建的值。因此，Alice 第一次传送中发送她的身份是毫无意义的。Bob 最早直到第三次传递才能确定她是 Alice。

然而，因为 Alice 可以知道 Bob 的第一条消息是新鲜的，所以 STS 和许多其他协议发送认证给服务器作为第二遍传递。为了抵制身份错误绑定的攻击，我们需要证明对方拥有共享的秘密 a • b。认证加密，$Box_{[a\cdot b]}$，或 mac 会完成它[10，第 3.1 节]。

STS 是否适合能力系统？答案是不。STS 做的第一件事就是对服务器进行身份验证，但是能力系统应该问的第一个问题是客户端是否具有这个资源的能力。由于不管怎样，Alice 都需要知道 B_p 从而对 Bob 进行认证，所以它成为一个访问 cap 的绝佳候选人。

然而，通过首先验证 Bob，并将他的公共密钥发送给未认证的客户端，将公共密钥授权给与 Bob 发起协议的任何人。由于这种不必要的授权，STS 作为能力系统并不是很有用。[5]

4.2. Encrypted Key Exchange - CurveCP

CurveCP [1]仅依赖于 nacl 的[3] Box 原语实现认证，它使用 curve25519 密钥。curve25519 密钥通过标量乘法组合以产生共享密钥，如 Diffie Helman 密钥[7]，而不是签名密钥。CurveCP 通过显示它们能够生成共享密钥来对每一方进行身份验证，并使用嵌套 box 来保护该身份验证免受窃听。

协议 2（CurveCP）

[4] Often the description of a handshake protocol ends as soon as each party is authenticated, but before the client *knows* it is authenticated. If this is the case, the client could receive an authentication error (or dropped connection) at the application layer, which is awkward. Thus, they do not know they've been accepted until receiving the first encrypted message. For this reason I've presented STS as a 4 pass protocol, even though the original paper describes it as a 3 pass protocol.

[5] Readers familiar with SSH will know the typical workflow for setting up a new server and accessing it for the first time. A server is created (on a cloud service such as AWS), with the user's public key in the authorized keys file. The user connects with the ssh command, and since this is the first connection, they will be prompted to accept the server's *key fingerprint*. Most users will just hit Y, since that is easier than checking. This allows a window for a man-in-the-middle attack, rendering ssh effectively a *trust on first use* system. On the other hand, if SSH was a capability system, the user would need to copy/paste the cap, completely avoiding this weakness. The user is likely to copy/paste the IP address anyway, so if the address and key where combined into one token it would be no more awkward. SSH requires an inconvenient security discipline, only followed by the most paranoid users, but a capability system would provide naturally better security at the same level of convenience.

$$
\begin{aligned}
? &\rightarrow B : a_p, Box_{[a \cdot B]}(okay) \\
? &\leftarrow B : Box_{[a \cdot B]}(b_p) \\
A &\rightarrow B : Box_{[a \cdot b]}(A_p || Box_{[A \cdot B]}(a_p)) \\
A &\leftarrow B : Box_{[a \cdot b]}(okay)
\end{aligned}
$$

Alice 将短期密钥，a_p和一个打包的标准消息发送给 Bob。Bob 把他短期密钥打包返回给了 Alice 的短期密钥。这使得 Alice 确信了 Bob 的身份，一个在该点上失败的中间人。Alice 打包（[a • b]）她的长期密钥，A_p，另一个打包（[A • B]）包含她的短期密钥。如果 Bob 对此感到满意，他会用一个标准的消息来回复，并将其打包加入到短期密钥中。[6]

在 CurveCP 中，知道 Bob 的公共密钥作为访问 Bob 的能力。不幸的是，这个协议有两个问题。

1. 如果 Bob 移动到不同的地址，重播攻击可以不知道他的公共密钥就确认他在哪。由于第一遍传递是加密的，只有 Bob 能够响应，如果它被重播到一个新的服务器，该服务器做出响应，则表明该服务器是 Bob。获取这些信息可能会鼓励重播攻击者寻找其他安全漏洞。
2. 更糟糕的是，第 3 步没有包括交换双方的新值。因此(a, $Box_{[A \cdot B]}$(ap)）形成了对 Bob 验证为 Alice 的可重用能力[6]。另见[8][第 5.1 节，只签署自己的指数]。
3. 最坏的情况是，CurveCP 包含一个通配符能力。由于使用了一个 curve25519 密钥进行身份验证，所以知道 B_{secret}[6]的行动者可以将任何人伪造成 Bob—在文献中这称为 Key Compromise Impersonation（KCI）。如果 Conrad 获得 B_{secret}，他可以把任何人伪装成 Bob。为了模仿 Alice，Conrad 将连接到 Bob，创建一个短期密钥 a，将其打包给 Bob，当 Bob 回应时，创建一个 $Box_{[a \cdot b]}(A_p||Box_{[A \cdot B]}(a_p))$，除了像 Bob 要打开它时一样使用($A_{public}$, B_{secret})，而不是像 Alice 要密封它时一样使用(A_{secret}, B_{public})。如果 Conrad 拥有 B_{secret}，他可以模仿 Bob 的任意密钥。因此，B_{secret}是一个通配符，而 CurveCP 不是一个性能良好的能力系统。

4.3. Deniable Key Exchange – OTR, Noise, TextSecure

有一类密钥交换协议使得可拒绝性成为设计目标[4,11,13]。这样做的依据是，当你进行非正式沟通时，你不会创造出你说你做过什么的证据。

这些协议出于与 STS 相同原因，即未验证的客户端在第二次传递时了解到服务器密钥，不适用于能力系统。因为这很容易修正，所以一个可拒绝的密钥交换是否可以成为一个良好的能力系统，值得进一步研究。虽然 TextSecure [11]取代 OTR [4]，但是因此使用第三方引入器，而不是直接类似于密钥交换，所以我们将检查 noise[13]协议来代替。

这里我们将继承共享密钥的符号来表达在多对密钥之间共享的密钥。a • b|a • B 表示与 a • B 连接的 a • b 的散列。只有能构建双方组件密钥的一方可以构造复合密钥。

协议 3（noise）

[6] Again, actually a 4 pass protocol.

$$
\begin{aligned}
? &\rightarrow ? \ : a_p \\
? &\leftarrow B : b_p, Box_{[a\cdot b]}(B_p), Box_{[a\cdot b|a\cdot B]}(okay) \\
A &\rightarrow B : Box_{[a\cdot b]}(A_p), Box_{[a\cdot b|A\cdot b]}(okay) \\
A &\leftarrow B : Box_{[a\cdot b|a\cdot B|A\cdot b]}(a_p|b_p)
\end{aligned}
$$

Alice 给 Bob 发送她的短期密钥。Bob 用他的短期密钥，一个包含他的长期钥匙的 box 和一个多密钥的 box 作为他是 Bob 的证据来回复。Alice 用多密钥 box 作为证据证明她知道 a_{secret} 和 A_{secret} 来回复。Bob 然后发回一个打包的消息，显示他也能够得出这个共享的秘密。[7]

要注意，与 CurveCP 不同，共享密钥不是在长期密钥之间导出的，而是仅在一个短期密钥和一个长期密钥之间导出，因此 B_{secret} 不是通配符。

即使 Alice 怀疑 Conrad 可能损害了她的长期密钥，A，她相信他肯定不会知道她的短期密钥。Conrad 不知道 a_{secret}，所以他不能建立 a•B，除非它真的是 Bob。为了向 Bob 提供相同的保证，他们最终得到了一个三向密钥 a•b|a•B|A•b，就像 TextSecure [11] 中的一样。这似乎是合理的，除了 b_{secret} 现在是一个通配符！（虽然，因为它是短期的，期望更难获得的）

1. 握手被保护免受窃听—但是任何连接到服务器的人都将被发送公钥，所以 B_p 作为一个能力并没有什么用。
2. Key Compromise Impersonation 仍然是可能的，但是变得更难了—为了冒充 Bob 的一个任意密钥你必须知道 Bob 的短暂和长期秘密密钥。对于任何被动阅读 Bob 的记忆的人来说，这是可能的—第一眼看起来似乎是一个不太可能的命题，但实际上如果 Bob 正在租用的虚拟机上运行，这正是他将要处理的情况。在一个表现良好的系统中，知道 Bob 的秘密密钥应该可以让你像 Bob 一样验证身份，但不能像 Alice 那样进行认证。

如果协议只在物理硬件上运行，则可以避免问题 2，但这是不合理的。由于它仍然包含通配符，因此该协议无法形成性能良好的能力系统。而且，可拒绝性属性感觉上难以理解。更高级协议会提供通信的证据吗？无论如何，一条安全通道不仅仅是用来进行非正式的社交通讯，而且可拒绝性对于能力系统来说似乎并没有任何特别的优势。

5. A New Design

如果我们从 noise 和 TextSecure 握手中获得一些想法，但是转换过来使得 Alice 首先进行身份验证，那么我们就会开始得到一个看起来像能力系统的东西了。

如果 Alice 预先验证 Bob，那么 Bob 可以通过又一次传递来验证 Alice。有两个初始传递来防止重播攻击，我们有一个 4 次传递协议。这不会比上述更糟，即使我们在第三次传递之前不认证任何人。

为了“预验证”，Alice 发送了她的身份证明，以及她要连接的意图给 Bob。预认证即可以通过加密也可以通过签名来实现。

协议 4（Deniable Capability Handshake）

[7] like STS, it’s really a 4 pass protocol

$$
\begin{aligned}
? &\rightarrow ? : a_p \\
? &\leftarrow ? : b_p \\
A &\rightarrow B : Box_{[a\cdot b|a\cdot B]}(A_p) \\
A &\leftarrow B : Box_{[a\cdot b|a\cdot B|A\cdot b]}(okay)
\end{aligned}
$$

Alice 把其短期密钥发送给了 Bob，他也用自己的回应了她。然后，Alice 打包她的长期密钥给 Bob，使得只有他才能打开它。Bob 通过打包标准信息表明他已经接受，使得只有 Alice 或 Bob 才能读取它。

要求 Alice 先认证是不寻常的，但我认为这是一个公平的协议。Bob 已经允许自己可以被公开寻址而把自己置于不利地位。Alice 先认证只会是公平的。通过加密她的认证，她不需要向任何人揭露她的身份，除了那个真正的 Bob。同样地，如果 Bob 选择不接电话，那么 Alice 将无法推断出他是否真的是 Bob。也许这只是他不想和她说话？也许这只是一个错误的号码？这样可以保护鲍勃免受骚扰。

如果 Bob 对他的客户端的身份不关心，他可能会允许任何人知道 B_p 进行身份验证。客户端可以通过使用另外一个临时身份来保持匿名。

在这个设计中，Bob 的公共密钥是一种能力，但它仍然具有 noise 和 TextSecure 所具有的通配符（KCI）问题。

密钥交换是保密和转发安全性所必需的，但需要签名来避免通配符。通过签名，我们将会有一个真正性能良好的能力系统。

由于我们需要交换和签名密钥，因此身份可以通过一对签名和交换密钥来表示。nacl 使用 ed25519 密钥进行签名，并使用 curve25519 密钥进行交换。然而，nacl 还提供了将签名转换为交换密钥的功能，因此身份可以表示为，在必要时可以将其转换为交换密的钥签名密钥.

协议 5（Capability-based Handshake）

$$
\begin{aligned}
? &\rightarrow ? : a_p \\
? &\leftarrow ? : b_p \\
H &= A_p|Sig_A(B_p|hash(a\cdot b)) \\
A &\rightarrow B : Box_{[a\cdot b|a\cdot B]}(H) \\
A &\leftarrow B : Box_{[a\cdot b|a\cdot B|A\cdot b]}(Sig_B(H))
\end{aligned}
$$

Alice 和 Bob 交换短期密钥，然后 Alice 私下向 Bob 预验证。鲍勃通过私下签署她的预认证来证明他已经接受。请注意，a_{secret} 和 b_{secret} 是解密握手的能力，但由于它们不是通配符，我们仍然拥有一个性能良好的能力系统。

设计变得越来越好。我们拒绝窃听，重播，中间人，KCI 攻击，并提供转发保密。没有通配符，也没有意外授权。但是还有一些小问题要解决。

实现方案在变化，所以拥有一个协议 ID 和与握手相关的版本通常是有帮助的。而且，如果两个初始传递中的任何一个被篡改，那么在接收到第一个认证通过之前，它将无法被发现。防止基于此协议构建的不同应用程序可能会相互干扰也可能是很好的。

最重要的是，我们必须小心避免造成选择协议攻击（Chosen Protocol Attack）[9]的弱点。如果签名密钥也在其他地方使用，则可能会重复使用此协议的签名，可能会导

致意外授权。所有这些问题都可以通过协议能力的应用密钥（K）来解决。可以通过使用 K 作为 hmac 的密钥来认证短期密钥。通过在每个签名中包含 K，证明该签名属于该协议，从而减轻 CPA。至关重要的是，如果有任何其他情况下使用签字密钥，则会采取类似程度的关注，以确保对签名的明确解释。

K 是访问该协议的一个实例的能力[8]。窃听者无法从密文中提取 K，但可以确认 K 的一个正确猜测。在 K 广为人知的情况下，一个知道 K 篡改初始传递但是创建一个有效的 $hmac_K$ 的攻击者，当接收到第三次传递时，它们的干扰仍将被检测到。

协议 6（Secret Handshake）

$$
\begin{aligned}
? \rightarrow ? &: a_p, hmac_K(a_p) \\
? \leftarrow ? &: b_p, hmac_{[K|a\cdot b]}(b_p) \\
H &= A_p|sign_A(K|B_p|hash(a\cdot b)) \\
A \rightarrow B &: box_{[K|a\cdot b|a\cdot B]}(H) \\
A \leftarrow B &: box_{[K|a\cdot b|a\cdot B|A\cdot b]}(sign_B(K|H|hash(a\cdot b)))
\end{aligned}
$$

与以前相同，但是共享密钥作为 K 开始，并且随着更多的公共密钥被学习而被扩展。Alice 的认证，$A_p|sign_A(K|B_p|hash(a \cdot b))$，证明她拥有 A，并且该证明是针对这个协议（通过 K）和这次握手（通过哈希(a·b)）。如果 Bob 还没有 Alice 的密钥，我们将其授权给他（显然我们希望 Bob 有这样的能力，否则 Alice 不会联系他）

为了让 Bob 返回认证 Alice，他可以签署 Alice 发送给他的证明，并将其发送回来。H 已经加密地链接到之前的传递。另一方面，比起 A_p 从来不是另一个协议的应用程序密钥来说，签署 K 减轻 CPA[9] 更容易让自己说服自己。

Alice 和 Bob 现在可以使用他们的共享密钥，K|a·b|a·B|A·b，和一个批量加密协议来保护双向通信通道的安全。

6. Future Work

4 传递协议的延迟对于某些应用可能是被禁止的。用于对下一个会话的一次性密钥进行预配置的机制可以实现一个两次传递协议，一对给定的行动者已经建立了至少一次联系。

有些读者会想知道 Alice 如何学习 Bob 的公钥？能力系统的概念上的框架有一个简单的答案：有人将其授权给她。在实践中，在 how 和 why 上面有一个很大的设计空间。简而言之，可以存在一个集中的注册表，一个 DHT，一个八卦网络，或可以在文件中配置的访问 cap。上述的各种组合可能是完全不同的系统。

7. Conclusion

我已经描述了一种高度私有的，适合于能力系统的 4 传递握手协议。它不会遭受重播，窃听，中间人，或 KCI 攻击。能力系统为分发和限制访问提供了一个概念框架，因此，当合理化决策时，它是一个有用的指南。在像这个协议的加密能力系统中，公共密

[8] i.e. an application which is built on it. *K* should be updated if the implementation of the application changes incompatibly. If backwards compatibility is required, the new protocol version could be used on a new port, until the legacy version is fully deprecated. For an openly specified application *K* may be publicly known. For a private application it may be a closely guarded secret.

钥和私有密钥只是访问权限。这允许我们思考其他不可想象的想法，例如一个秘密公共密钥或者一个共享私有密钥的概念。通过这些概念，我们可以创建动态的访问和限制层。由于秘密握手在其他方面是无意义的，并且安全的双向通信是根本，如果它与合适的流批量加密协议相结合，那么它可能是分散式访问控制系统的强大构建区块。

Acknowledgements

Reference