第 9 章 对网络安全协议的典型攻击

与第一部分类似，在阐述网络安全协议(也常称为密码协议)之前，本章首先阐述和分析几类非常典型的“黑客成就”，从反面讨论网络安全协议失败的例子。这些例子都是近二十年内被发明的各种安全协议，在建立之初都被“证明”过是“安全”的，而多年之后却被指出有严重的安全漏洞。本章目的是建立起协议分析的依据(虽然并不完整)，使读者初步学会如何分析对一个网络协议是否存在常见的攻击。除此之外，这些例子本身也是有趣的，对其中某些修改过的协议，后来已严格证明确实是安全的。

9.1 一个例子、一段历史

第一个有趣而又非常简单的例子是 Needham-Schroeder 协议，这是一个基于公钥体制的身份验证协议[1]，在其发表以后的很长一段时间内都被认为是安全的，直到该协议发表 17 年后的 1995 年才被 Gavin Lowe 发现了一个严重的安全漏洞。改正后的协议称为 Needham-Schroeder-Lowe 协议。

原始的 Needham-Schroeder 协议非常简单，表示如下。A、B 是两个用户或进程的名字（ID），N_a、N_b 是随机数，K_A、K_B 分别表示 A、B 的公钥，$\{m\}_K$ 表示以 K 对消息 m 加密后的密文，X,Y 表示字 X 和字 Y 的联结，这些都是典型的消息表达式。Needham-Schroeder 协议的消息交换过程为：

$$A \rightarrow B : \{N_a, A\}_{KB}$$

$$B \rightarrow A : \{N_a, N_b\}_{KA}$$

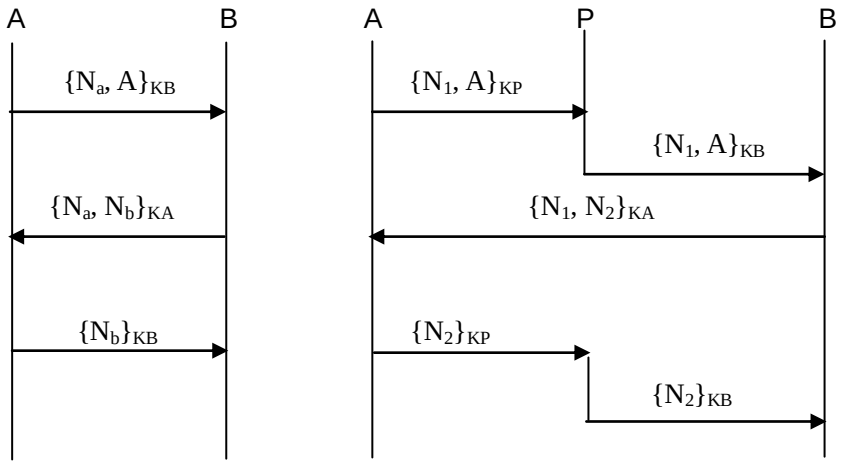
$$A \rightarrow B : \{N_b\}_{KB}$$

该协议也可以用分布式计算系统中进程的 Lamport 图表达，见图 9-1(a)，这种图形表

[1] 历史上有两个 Needham-Schroeder 协议，都发表于 1978 年，我们这里讨论的是基于公钥体制的 Needham- Schroeder 协议，另一个 Needham-Schroeder 协议基于对称加密体制，早年就被发现有被重放攻击的漏洞，改正后发表于 1987 年，与同时发表的 Otway-Rees 协议很相似。对称 Needham-Schroeder 协议的安全漏洞在 1989 年被 BAN 逻辑重新发现。改正后的对称 Needham-Schroeder 协议就是 Kerberos 协议早期版本的基础。

耐人寻味的是，在发表 BAN 逻辑的同一篇论文中，对公钥 Needham-Schroeder 协议分析的结果并没有发现其安全漏洞，但这只能说明早期分析方法和技术不够成熟，“没能发现安全漏洞”和严格证明“不存在安全漏洞”是截然不同的事情！

示法是一种很直观并且可以精确化的表示方法，读者应能熟练使用。



(a) Needham-Schroeder 协议　　(b) 对 Needham-Schroeder 协议的一种攻击

图 9-1　Needham-Schroeder 协议及其安全漏洞

直观上看，A、B 通过 Needham-Schroeder 协议向对方证实自己的存在。A 将自己生成的随机数 N_a 用 B 的公钥加密后发送到 B；B 生成一个自己的随机数 N_b 后连同 A 的随机数 N_a 一起传递回 A，整个消息以 A 的公钥加密；A 以自己的私钥解密该消息，验证其中包含的第一个数确实是自己在第一个消息中向 B 传递的那个随机数（否则 A 立刻终止该协议），然后将该消息中的第二个数以 B 的公钥加密后传递回 B；B 以自己的私钥解密该消息，验证其中所含的恰是其在第二条消息中生成的自己的那一个随机数 N_b。

因为所有消息都以接收方的公钥加密，因此只有接收方才能识别消息的内容。因为两个随机数同时出现在第二条消息中，因此 A、B 都能肯定对方与自己同属于当前的协议会话（这是随机数在协议中的重要用途之一，所谓随机数，其基本属性就是每次随机、独立地生成，因此可以合理假定协议会话中出现的所有随机数是互不相同的[2]，本书今后在讨论形式方法时将始终遵循这一假定）。

如果上面所叙述的一切验证都成立，则协议将正常结束，A、B 在协议结束时都应得出结论“对方存在于刚刚完成的协议会话中，并且该会话中的两个随机数 N_a、N_b 只有自己和对方知道”。这就是 Needham-Schroeder 协议所追求的安全性质（但直到 BAN 逻辑发表之前，一个协议的安全性质却从未被精确地表达出来过）。

[2] 更精确地讲，是如此生成的两个随机数相等的概率随数的位数大小指数地减小，从而可以忽略，一个经常使用的方法是按均匀分布随机生成 n 位二进制数，这样两个随机数相等的概率=2^{-n} 。

这里要强调的是，在以上协议（实际上在任何协议）中，每个参与者都是而且只能是根据其自身所发送-接收到的所有消息进行推断，密码协议的目的就是使得在协议进行到特定时刻时参与者能根据这些消息推断出特定的结论。更具体地讲，Needham-Schroeder 协议的目标就是使 A 在接收到第 2 条消息时，能根据前两条消息+$\{N_a, A\}_{KB}$−$\{N_a, N_b\}_{KA}$（+号表示发送，−号表示接收，本章今后沿用这一表示）推断结论“B 存在于当前的会话中，且 N_b 由 B 生成”。同样，在接受到第三条消息时，B 能根据−$\{N_a, A\}_{KB}$+$\{N_a, N_b\}_{KA}$−$\{N_b\}_{KB}$ 推断“A 存在于当前的会话中，且 N_a 由 A 生成”。

果真如此吗？毕竟，以上这些“论证”不是严格而精确的分析。首先，该协议的安全目标的精确含义究竟如何表达？其次，该协议的行为本身应该如何精确表示？最后，如何根据这两者精确地验证或证明该模型确实有所要求的性质？这些问题都是形式分析方法和计算密码学方法要精确回答的问题，已超出本教程的水平。不过现在暂时回到直观的论证上来，看看 Needham-Schroeder 协议是如何被破解的。

值得特别注意的是第二条消息$\{N_a, N_b\}_{KA}$，它要表达的意思是“A、B 当前存在于一个时效由随机数 N_a、N_b 界定的会话中”，但注意该消息表达式中实际上没有任何信息可以使接收方据此推断发送方是 B 而不是其它参与者。细心的读者或许会争辩说该消息中的项 N_a 来自 A 发送的第一条消息$\{N_a, A\}_{KB}$，既然该消息以 B 的公钥加密，“当然”只有 B 能解密从而提取出 N_a，因此当 A 看到 N_a 出现于$\{N_a, N_b\}_{KA}$ 中时，“当然”能据此推断出是 B 而不是其它进程发送了该消息。

问题是，这一切需要在一个严格的形式体系和概念框架中精确地证明。实际上，以上“论证”并不成立，图 9-1(b)中的攻击恰是对 Needham-Schroeder 协议这一漏洞的巧妙利用。

A、B 以及其它符号如前述，P 表示攻击者，假定 P 以前破译出过一个合法参与者的私钥，其对应的公钥为 K_P。当 A 期望向 P（从现在起对 P 及其破译出私钥的参与者不加区分，用同一符号表示）证实身份时，向 P 发送消息$\{N_1, A\}_{KP}$，P 提取出 A 的随机数 N_1 后以 A 的身份向 B 发送消息$\{N_1, A\}_{KB}$，B 根据协议的规定向 A 发送消息$\{N_1, N_2\}_{KA}$，注意如前所述，A 并不能根据该消息本身推断出该消息的实际发送者是 B 而不是 A 所期望的 P，于是 A 接收该消息后能正确地通过验证并向 P 发送消息$\{N_2\}_{KP}$，P 提取出 N_2 后向 B 发送$\{N_2\}_{KB}$。至此，根据 Needham-Schroeder 协议，A 断定“P 存在于 N_1, N_2 所界定的会话中，且 N_1, N_2 仅有 A、P 知道”，B 断定“A 存在于 N_1, N_2 所界定的会话中，且 N_1, N_2 仅有 A、B 知道”，而事实上，两者的结论都是错误的。

在实际应用中，该错误可能导致严重后果，因为随机数 N_1, N_2 可能在身份验证之后用

于合成会话密钥，当 B 以这样合成的密钥向 A 发送敏感消息时，P 能轻易地破译这些消息，这当然不是协议设计者所期望的结局。

对 Needham-Schroeder 协议的改正非常简单，这就是 Needham-Schroeder-Lowe 协议：

$$A \rightarrow B : \{N_a, A\}_{KB}$$

$$B \rightarrow A : \{N_a, N_b, B\}_{KA}$$

$$A \rightarrow B : \{N_b\}_{KB}$$

Needham-Schroeder-Lowe 协议的安全性质目前已经被严格证明。

9.2 更多的例子

上一节较详细地解释了对 Needham-Schroeder 协议的一种攻击，在发现该攻击之前该协议曾长期被认为是安全的。事实上，这类例子是很多的，本节将给出更多的例子，同时我们也试图通过这些例子总结出一些经验性的原则，从直观的角度初步分析为什么会有安全漏洞。

9.2.1 其它身份鉴别协议的例子

Denning-Sacco 协议 这是一个基于公钥体制的会话密钥交换协议，参与的进程是准备进行会话的双方 A、B 和一个双方共同信任的服务器 S，A、B 的公钥分别是 K_a、K_b，对应的私钥分别为 d_a、d_b，且 A、B 分别有证书 CA、CB。在下面的分析中 CA、CB 的具体形式无关紧要，消息交换过程如下：

$$A \rightarrow S: \quad A,B$$

$$S \rightarrow A: \quad CA,CB$$

$$A \rightarrow B: \quad CA,CB,\{\{K, T_a\}_{da}\}_{Kb}$$

在最后一条消息中，A 生成会话钥 K 和一个表示时效的随机数 T_a，两者相关联后共同以 A 的私钥签字，再以 B 的公钥加密，目的是想提供对 K 的发送方的身份鉴别和对 K 的保密：只有 B 能从该消息提取出正确的 K，并且能判定该消息来自于 A。

然而，消息 $CA,CB,\{\{K, T_a\}_{da}\}_{Kb}$ 对该涵义的表达并不充分，具体地说，消息中的子项 $\{\{K, T_a\}_{da}\}_{Kb}$ 可以被一个意图假冒发送者身份的进程所重用。设想 C 是这样一个进程，在与 A 以 Denning-Sacco 协议获取了会话密钥 K_{ac} 后，C 再与 B 进行一次 Denning-Sacco 协议，C

向 B 发送的最后一条消息是：

$$C \rightarrow B: \quad CB,CC,\{\{K_{ac}, T_a\}_{da}\}_{Kb}$$

于是，根据 Denning-Sacco 协议 B 将判定 K_{ac} 来自 A，而实际上 C 也知道该密钥，当 B 以 K_{ac} 加密其敏感数据与 A 会话时，将立刻被 C 破译。

对 Denning-Sacco 协议可以改正如下：

$$A \rightarrow S: \quad A,B$$

$$S \rightarrow A: \quad CA,CB$$

$$A \rightarrow B: \quad CA,CB,\{\{A,B,K,T_a\}_{da}\}_{Kb}$$

Woo-Lam 协议 这是一个基于对称加密体制的身份鉴别协议，S 是被共同信任的服务器进程，进程 B 期望验证进程 A 当前的确存在，A、B 双方的消息交换过程为（注意被验证的 A 是协议过程的发起方）：

$$A \rightarrow B: \quad A$$

$$B \rightarrow A: \quad N_b$$

$$A \rightarrow B: \quad \{N_b\}_{Kas}$$

$$B \rightarrow S: \quad \{A,\{N_b\}_{Kas}\}_{Kbs}$$

$$S \rightarrow B: \quad \{N_b\}_{Kbs}$$

N_b 是随机数、K_{as}、K_{bs} 分别是 A、S 之间和 B、S 之间的共享密钥。B 解密最后一条消息$\{N_b\}_{Kbs}$，将提取出的明文与它在第二条消息中发送的随机数比较，如果相同则 B 判定 A 当前确实存在。

果真如此吗？如果最后一条消息$\{N_b\}_{Kbs}$ 中的 N_b 确实来自第三条消息$\{N_b\}_{Kas}$ 中的 N_b，则 A 确实存在，但注意 B 在接收到消息$\{N_b\}_{Kas}$ 时并不能对其明文做任何推断（请思考为什么？），需要借助于 S 从密文$\{N_b\}_{Kas}$ 提取出 N_b 后再以形式$\{N_b\}_{Kbs}$ 传递给 B，而观察整个协议，S 并没有任何依据能判定什么样的 N_b 的密文才是合法的（以正确的密钥加密）。因此，直观的分析表明 B 并没有充分的依据判定最后的消息$\{N_b\}_{Kbs}$ 中的 N_b 确实是 A 发送的。

以上分析可以归纳为：从 B 的观点看，消息 4 到消息 5 是多对一的关系，即形如$\{A',\{N_b\}_{Ka's}\}_{Kbs}$ 和$\{A'',\{N_b\}_{Ka''s}\}_{Kbs}$ 的消息 4 都对应相同形式的消息 5 $\{N_b\}_{Kbs}$，利用这一弱点，可以构造出对该协议的一种攻击如下。

A、B、C 是三个进程，A 当前并未运行，C 向 B 同时发起两个 Woo-Lam 协议过程，其中一个假冒 A 的身份，另一个以自己的真实身份运行，并截获所有发向 A 的消息：

$$C \rightarrow B: \quad A \qquad \text{C 假冒 A 的身份}$$

C → B： C

B → A： N_b C 截获该消息

B → C： N'_b

C → B： $\{N_b\}_{Kcs}$ C 假冒 A 的身份

C → B： $\{N_b\}_{Kcs}$

B → S： $\{A,\{N_b\}_{Kcs}\}_{Kbs}$

B → S： $\{C,\{N_b\}_{Kcs}\}_{Kbs}$

S → B： $\{N''\}_{Kbs}$

S → B： $\{N_b\}_{Kbs}$

N”是 S 以 K_{as} 解密$\{N_b\}_{Kcs}$的结果，一般地有 N”≠N_b、N'_b，但注意在以上协议过程中，B 不能根据最后一条消息本身准确判定$\{N''\}_{Kbs}$、$\{N_b\}_{Kbs}$中的哪一个和$\{N_b\}_{Kcs}$对应，即$\{N''\}_{Kbs}$、$\{N_b\}_{Kbs}$分别属于两个并发协议过程中的哪一个。根据其中有一条对应、另一条不对应的情况，B 会得出结论认为当前 A 在运行而 C 没有（请读者特别注意 C 故意将 B 发送给 A 的随机数 N_b 重复用于两条消息$\{N_b\}_{Kcs}$中），结果与事实恰好相反。

对 Woo-Lam 协议的改正如下：

A → B： A

B → A： N_b

A → B： $\{N_b\}_{Kas}$

B → S： $\{A,\{N_b\}_{Kas}\}_{Kbs}$

S → B： $\{A,N_b\}_{Kbs}$

即将其最后一条消息改为$\{A,N_b\}_{Kbs}$，使之明确包含 B 期望验证其存在性的那个进程的 ID。

SSL 协议 早期的 SSL(Secure Socket Layer)协议按以下过程交换会话密钥 K：

A → B： $\{K\}_{Kb}$

B → A： $\{N_b\}_K$

A → B： $\{CA, \{N_b\}_{da}\}_K$

K 是 A 生成的会话密钥，d_a 是与公钥 K_a 对应的私钥，N_b 是随机数，CA 是 A 的证书，具体形式与下面的分析无关。随机数 N_b 的目的是用以表达 K 的时效，但注意以上第三条消息并不能完全表达“K 仅用于 A、B 之间的会话，且 K 的时效与 N_b 相同”这一命题。考虑以下两个交错的 SSL 协议过程：

A → B： $\{K\}_{Kb}$

$$B \rightarrow C: \{K\}_{Kc} \quad B 假冒 A$$

$$C \rightarrow A: \{N\}_K$$

$$A \rightarrow B: \{CA, \{N\}_{da}\}_K$$

$$B \rightarrow C: \{CA, \{N\}_{da}\}_K$$

在协议终止时，A 认为 A 与 B 当前有会话钥 K，C 则认为 A、C 之间当前有会话钥 K，但 C 并未意识到 B 也知道 K。

改正后的 SSL 协议如下：

$$A \rightarrow B: \{K\}_{Kb}$$

$$B \rightarrow A: \{N_b\}_K$$

$$A \rightarrow B: \{CA, \{A,B,K,N_b\}_{da}\}_K$$

9.2.2 加密的运用

在设计密码协议时,无论对称密码体制还是公钥体制都被以各种方式广泛应用，这里以 Kerberos 协议为例来解释加密在密码协议中的主要应用方式。

Kerberos 协议(早期版本)过程如下，其中 T_s、T_a 是时戳，L 是会话钥 K 的寿命，K_{as}、K_{bs} 是会话双方 A、B 与其共同信任的验证服务器 S 之间的共享密钥：

$$A \rightarrow S: A,B$$

$$S \rightarrow A: \{T_s, L, K, B, \{T_s, L, K, A\}_{Kbs}\}_{Kas}$$

$$A \rightarrow B: \{T_s, L, K, A\}_{Kbs}, \{T_a, A\}_K$$

$$B \rightarrow A: \{T_a+1, A\}_K$$

在向下讨论之前，先借此协议介绍一种很有用的协议进程表示法。在分析协议时，重要的是明确每个参与协议的进程从自身观点都“看到”了什么、“没有看到”什么，即进程按特定时间顺序接收/发送的一组消息（以后称为进程上的消息序列或 strand），以及这些消息的内部结构。

按照在 9.1 节引进过的表示法，进程 A 上的消息序列（或称 strand-A）是

$$+A,B \ -\{T_s, L, K, B, x\}_{Kas}, +x, \{T_a, A\}_K \ -\{T_a+1\}_K$$

式中的+表示发送，−表示接收，x 表示变元，其它符号为常元。变元表示该进程对其值完全无知，是可以被攻击者潜在地赋予任何值的消息项，而常元表示其值可以被进程完全确定的消息项。以上面的表示为例，A 发送的第一条消息是明文，故表示为+A,B；A 接收的消息$\{T_s, L, K, B, \{T_s, L, K, A\}_{Kbs}\}_{Kas}$ 中，由于整个消息以 A 和 S 的共享密钥加密，故 A 能解密之从而

能完全确定其中的项 T_s, L, K, B 的值，但注意项$\{T_s, L, K, A\}_{Kbs}$是以 B 和 S 的共享密钥加密的，A“看不到”该项内部的各个项的值，因此以变元 x 表示；A 在第三条消息中不加改变地、完整地发送该加密项，因此表示为+x。

同理，B 上的消息序列是 $-\{T_s, L, K, A\}_{Kbs}, \{T_a, A\}_K +\{T_a+1, A\}_K$ ，S 上的消息序列是$-A,B +\{T_s, L, K, B, \{T_s, L, K, A\}_{Kbs}\}_{Kas}$ 。注意 S 拥有所有密钥，因此能“看到”所有消息项，从而在 S 的消息序列中只有常元，没有变元。

以上表示法在运用 strand-图模型分析协议时经常使用，在讨论 strand-图模型的章节将精确定义，但读者应该在直观上先建立一个正确的理解，明确其所表达的概念。

回到 Kerberos 协议，对其消息中加密的用法解释如下：

在消息 S→A：$\{T_s, L, K, B, \{T_s, L, K, A\}_{Kbs}\}_{Kas}$ 中，最外层以 A、S 的共享密钥加密，既是用于保密（只有 A 能解密），也是用于身份鉴别（只有 S 与 A 共享密钥 K_{as}，使接收方能判定该消息中的项均来自于 S）。这都是加密的典型用法。

除此之外，消息$\{T_s, L, K, B, \{T_s, L, K, A\}_{Kbs}\}_{Kas}$ 中同时包含项 T_s、L、K 和 B，是通过加密将它们相关联，表达一种由公共信任方所约定的关系，例如在这里$\{T_s, L, K, B, x\}_{Kas}$ 是对 A 表达“钥 K 用于与 B 的会话，且其时效由参数 T_s、L 决定”，$\{T_s, L, K, A\}_{kbs}$ 则是对 B 表达“钥 K 用于与 A 的会话，且其时效由参数 T_s、L 决定”。

在消息 A→B：$\{T_s, L, K, A\}_{Kbs}, \{T_a, A\}_K$ 中，第一项表达关联，第二项$\{T_a, A\}_K$ 除表达关联之外，还用以验证接收方是否持有该消息的加密钥 K，并通过解密第四条消息证实。

最后注意消息$\{T_s, L, K, B, \{T_s, L, K, A\}_{Kbs}\}_{Kas}$ 是两重加密的，即对密文$\{T_s, L, K, A\}_{Kbs}$ 再行加密，考虑到加密的计算代价，这种多重加密方式应该尽量避免使用。

9.2.3 时　效

密码协议的协议状态往往是有时效要求的，典型的如在线协商会话密钥的协议，要求产生的会话密钥 K 必须是本次协议协商的结果，而非以前结果的重放。在寻求这类时效性目标时，常常用到随机数。下面的例子说明，随机数必须以正确的方法使用，单纯的随机性并不足以保证时效性。

第一个例子是对称 Needham-Schroeder 协议，借助于一个公共可信任的服务器 S 生成 A、B 之间的会话密钥 K：

A→S：A,B,N_a

S→A：$\{N_a,B, K, \{K, A\}_{Kbs}\}_{Kas}$

A→B：$\{K, A\}_{Kbs}$

B→A：$\{ N_b\}_K$

A→B：$\{ N_b+1\}_K$

N_a、N_b 分别是 S 和 B 生成的随机数，最后两条消息的目的是用来使 A 向 B 证明 A 知道密钥 K。注意 K 是由 S 生成的，并且 S 生成的消息项$\{K, A\}_{Kbs}$ 明确地将 K 与 A 关联，似乎除非能破译 B、S 之间的共享密钥 K_{bs}，否则 B 通过最后两条消息总能证实当前知道密钥 K 的一定是 A。

可惜，以上结论是错误的，关键在于这里的 K 没有时效保证，一个攻击者 P 可以用一个他已经破译出的旧会话密钥 K_0 来欺骗 B 并将自己伪装成 A，看下面的攻击途径：

P→S：A,B,N_a

S→P：$\{N_a,B, K, \{K, A\}_{Kbs}\}_{Kas}$

A/P→B：$\{K_0, A\}_{Kbs}$　　P 重放以前的消息$\{K_0, A\}_{Kbs}$

B→A/P：$\{ N_b\}_{K0}$

A/P→B：$\{ N_b+1\}_{K0}$

A/P 表示 P 伪装成 A 的身份进行操作，实际操作者是 P。注意以上攻击实际上并不需要前两条消息，P 只需要直接从第三条消息开始攻击，$\{K_0, A\}_{Kbs}$ 来自 P 重放以前的消息，这时 A 根本不需要去破译 K_{bs}。既然 P 知道 K_0，当然能顺利完成最后两条消息交换，从而向 B “证实”他就是 A！

这种错误发生的根源在于 K 没有显式地与一个表达时效的随机数关联（如果读者愿意将消息$\{ N_b\}_{K0}$ 理解为这种关联—毕竟 N_b 是一个随机数—那么上面的攻击说明这种关联的强度是不够的）。如果我们能认识到在这个协议里真正反映时效的随机数应该是 N_a 而不是 N_b，那么可以将协议改进如下：

A→S：A,B,N_a

S→A：$\{N_a, B, K, \{K, N_a, A\}_{Kbs}\}_{Kas}$

A→B：$\{K, N_a, A\}_{Kbs}$

B→A：$\{ N_b,N_a\}_K$

A→B：$\{ N_b+1, N_a \}_K$

最后两条消息显式地将 K 和 N_a 关联，以上攻击失效。

这里提醒读者注意一个微妙的细节：如果攻击者能够在未知 K 的情况下从$\{N_b,N_a\}_K$ 推断出$\{ N_b+1, N_a\}_K$，那么以上改进也无效，因此我们实际上是隐含地假设了这种情况不可能发

生，但实际上，这取决于在实现该协议时使用什么类型的加密运算$\{\ \}_K$，而事实是，并非任何安全的加密运算都能够满足这一要求[3]。这个例子再次说明在设计协议时明确陈述关于安全性的前提或假设是非常重要的，协议失败的原因往往就出在设计者隐含地做了不适当的假设。

实现时效性并非意味着一定要使用随机数，序列号也是可以使用的，但这类可预测的“随机数”在使用时需要特别小心，常常需要加密保护。下面的协议是一个例子：

$A \rightarrow S$：A, N_a

$S \rightarrow A$：$\{T_S, N_a\}_{Kas}$

S 是一个时间服务器，A 向 S 请求当前时间，N_a 是一个序列号，A 每次请求时将其值加 1，S 在回答请求时将当前时间 T_S 与 N_a 显式地关联，以此表达“T_S 是对序号为 N_a 的那个请求的回答”。注意在这里运算$\{\ \}_{Kas}$起类似于数字签名的作用，向 A 证实该回答来自 S。

这里的问题是，N_a 的变化是可预测的，一个攻击者可以用下面的方式以一个过去的时间欺骗 A：

$P \rightarrow S$：A, N_a+1　　A 的当前序号值应该是 N_a

$S \rightarrow A$：$\{T_S, N_a+1\}_{Kas}$　P 截获该消息

$A \rightarrow S$：A, N_a+1

$P \rightarrow A$：$\{T_S, N_a+1\}_{Kas}$　P 重放前面截获消息，这时的实际时间应该是 $T'_S > T_S$

要避免以上攻击，该协议有两种改进方法，或者将 N_a 取随机数，或者对 N_a 加密：

$A \rightarrow S$：$A, \{N_a\}_{Kas}$

$S \rightarrow A$：$\{T_S, N_a\}_{Kas}$

9.2.4 类型攻击

类型攻击是一种非常巧妙的攻击技术，它利用某些消息域的类型不确定性有意混淆某些消息项的值，使接收方错误地解释该消息域，达到攻击的目的。这类攻击目前已经发现不少例子，包括一直被认为是比较安全的某些协议。下面是最近在苏黎士联邦工业大学的 OFMC 系统上发现的关于 Yahalom 协议的一个类型攻击。

Yahalom 协议是一个带身份鉴别的会话密钥生成协议，参与方 A、B 通过双方都信任的服务器生成会话密钥 K：

[3] 这一要求就是非可塑性(non-malleability)，更一般地，是指加密运算$\{\}_K$保证在未知 K 的条件下不能从$\{m\}_K$有效地推断出 $\{f(m)\}_K$, 这里 f 是任何一个有效算法。这是一个很强的计算密码学概念，我们今后始终假设它成立。

$A \rightarrow B$：A, N_a

$B \rightarrow S$：$B, \{A, N_a, N_b\}_{Kbs}$

$S \rightarrow A$：$\{B,K, N_a, N_b\}_{Kas}, \{A, K\}_{Kbs}$

$A \rightarrow B$：$\{A, K\}_{Kbs}, \{N_b\}_K$

K_{as}、K_{bs} 分别是 A、B 与 S 之间的共享对称密钥。根据协议，B 在得到第四条消息$\{A, K\}_{Kbs}$，$\{N_b\}_K$ 后应确认会话钥 K 来自服务器 S，但实际上，一个攻击者 P 却可以不通过服务器 S 而自己生成一个会话密钥，并使 B 相信该密钥是来自服务器 S。这里假定 P 持有一个已被破译出的合法参与者 A 与服务器 S 的共享密钥 K_{as}，或 P 就是一个合法但不诚实的参与者 A，看下面的攻击过程：

$A/P \rightarrow B$：A, N_a

$B \rightarrow S$：$B, \{A, N_a, N_b\}_{Kbs}$ P 截获该消息

$S \rightarrow A/P$：$\{B, K, N_a, N_b\}_{Kas}, \{A, K\}_{Kbs}$

$A/P \rightarrow B$：$\{A, \underline{N_a, N_b}\}_{Kbs}, \{N_b\}_{\underline{Na,Nb}}$

注意下划线部分，P 在第四条消息中回放它截获的第二条消息的项$\{A, N_a, N_b\}_{Kbs}$，因为 P 已知 K_{as} 故能解密出 N_b，然后以字联结 N_aN_b 为密钥加密 N_b；B 在接收到第四条消息时从 K-域提取出的是 N_aN_b 并且对$\{N_b\}_{\underline{Na,Nb}}$的解密“证实”了 N_aN_b 正是“来自服务器的会话密钥”。

这类例子提示协议设计者，在协议消息中不加类型限定地使用消息域，例如，滥用可变字长的域，将是非常危险的[4]。

9.2.5 代数攻击

如果协议本质上依赖于某些数学特征，例如 Diffie-Hellman 协议，攻击者毫无疑问有更多的途径可以利用，这些“更多的途径”来自于将消息域看做可运算的数，从而有可能利用代数性质实现攻击。Bull 协议就是一个十分简单但颇耐人寻味的例子，这个协议为三方 A、B、C 的通讯生成会话密钥，A、C 彼此之间不应该知道它们与 B 之间的会话密钥，协议过程如下：

$A \rightarrow B$：$M_a=A,B,N_a,h(A,B,N_a)$

$B \rightarrow C$：$M_b=B,C,N_b,M_a, h(B,C,N_b,M_a)$

[4] Yahalom 协议经受住了长期的分析，为什么一直没有发现这一攻击，与分析方法有关，事实上大多数分析系统带类型限定或要求密钥为原子项，因此未能发现这一类型攻击就不奇怪了，例如 Paulson 在 1999 年的论文中证明了 Yahalom 协议满足严格的身份鉴别性质，但他的系统是有类型的，因此更准确地说是证明了在有类型限定的条件下 Yahalom 协议是安全的。在这里读者已经看到，如果去掉类型限定这一条件，Yahalom 协议是不安全的。

C→S：M_c=C,S,N_c,M_b, h(C,S,N_c,M_b)

S→C：$K_{ab}\oplus h(K_{as}, N_a)$, $K_{ab}\oplus h(K_{bs}, N_b)$, $K_{bc}\oplus h(K_{bs}, N_b)$, $K_{bc}\oplus h(K_{cs}, N_c)$

C→B：$K_{ab}\oplus h(K_{as}, N_a)$, $K_{ab}\oplus h(K_{bs}, N_b)$, $K_{bc}\oplus h(K_{bs}, N_b)$

B→A：$K_{ab}\oplus h(K_{as}, N_a)$

K_{as}、K_{bs}、K_{cs} 分别是 A、B、C 与共同信任的服务器 S 之间的共享对称密钥，N_a、N_b、N_c) 是 A、B、C 生成的随机数，K_{ab}、K_{bc} 是 S 生成的 A 和 B 之间、B 和 C 之间的会话密钥匙，h 是单向散列函数，⊕是位异或运算。

显然，A 不应该知道 K_{bc}，C 也不应该知道 K_{ab}。这一协议的安全性质也曾经得到过“证明”，但是读者仔细观察 C 获得的消息 $K_{ab}\oplus h(K_{as}, N_a)$, $K_{ab}\oplus h(K_{bs}, N_b)$, $K_{bc}\oplus h(K_{bs}, N_b)$, $K_{bc}\oplus h(K_{cs}, N_c)$就会发现以下等式：

$K_{bc} = h(K_{cs}, N_c)\oplus (K_{bc}\oplus h(K_{cs}, N_c))$

$K_{ab}=(K_{ab}\oplus h(K_{bs}, N_b))\oplus (K_{bc}\oplus h(K_{bs}, N_b))\oplus K_{bc}$

于是，C 总能知道 K_{ab}！

事实上，与前面的脚注 4 相似，对 Bull 协议的安全性证明是有条件的，这个条件就是不考虑运算⊕的代数性质。这个例子恰恰说明这样一种前提很不充分。许多现代密码协议对代数性质的依赖是本质性的，包括 RSA 和 Diffie-Hellman 协议，问题在于，当考虑代数性质时，协议分析问题可能成为不可解问题，即使在可解的情形，其分析技术也比不考虑代数性质的所谓自由代数要复杂得多。这是协议分析领域目前极富挑战性的方向之一。

9.3 更复杂的协议和攻击

上一节讨论的例子几乎都是用于密钥协商与身份鉴别的协议，所追求的安全目标主要是保密性和身份证实，但密码协议的分析与验证技术并不局限于这类协议，同样也能针对更高级的协议和更复杂的安全性质来应用。尽管如此，密钥协商与身份鉴别协议仍然是我们最经常使用的例子，用来检验我们的理论和技术，这是因为一方面这类协议应用最广泛，另一方面这类协议从结构到安全性质相对而言最简单，适合于作为对一种复杂理论的最初的检验。当我们通过对这类协议的分析与验证积累了足够的规律性认识之后，就可以将这些方法向更复杂的协议推广。

在具体攻击策略上，除了前面这些针对单一协议进行的攻击外，更复杂的攻击是利用多种不同的协议（而不是单一协议的不同的运行实例）之间可能的相互干扰，例如同时运行两

个不同的协议 P_1 和 P_2，使 P_2 错误地将 P_1 的消息解释为自己的消息，凡此种种。这就要求对多个协议进行联合验证，目前这方面的分析验证工作还很少，但多种协议并发运行时的干扰问题应该是任何安全系统设计者应该顾及的问题之一。一个具体的例子见习题 9.5。

9.4 小结与进一步学习的指南

本章不借助于任何专门技术，完全以直观的方式分析了一些著名的网络安全协议实际上并非安全，并在许多情况下具体构造出了攻击途径。在近二十年来的研究实践中，发现这些事后看来十分简单的攻击途径却往往不是件容易的事情，甚至常常是改正的协议在几年后又被人发现有新的安全漏洞，反复改进。读者要特别注意的是，靠发现特定的攻击途径来检验协议是否安全这种方法在实际应用中既不容易、也不充分（请思考为什么？）。读者从上面的各种例子中能体会到需要严格而统一的理论方法与技术来建立协议的精确模型、定义协议的安全性质、论证协议的安全条件（充分条件和必要条件），最终以此为基础证明一个协议是安全的或不安全的。赶兴趣的读者可以参考作者最近的专著：

田园 著 计算机密码学：通用方案构造及安全性证明 北京：电子工业出版社,2008

尽管直观的论证与分析既不精确也不充分，但从上面的讨论中还是能归纳出一些较普遍的规律，作为今后设计密码协议的经验性指导原则，以避免重复前人的错误，在这方面它们是颇有价值的。对这些经验性原则的详细解释至今值得学习，读者可以参考 Abadi 和 Needham 的著名论文 *Prudent Engineering Practice for Cryptographic Protocols*, Research Report 125, Digital Equipment Corp. Systems Research Center,1995。

B.Schnerier 的名著 Applied Cryptography(中译本：应用密码学， 吴世忠 等译，北京：机械工业出版社，2000)有相当篇幅讨论各种协议及失败的例子，值得一读。

BAN 逻辑是在严格的形式基础上对密码协议进行分析的第一种系统方法，具体地讲，BAN 逻辑是一个一阶逻辑体系，它将协议所依赖的假设明确表达为一组公理，将协议过程表达为一组谓词公式，将待验证的安全性质（也是一组谓词公式）作为该形式体系中的定理加以证明。BAN 逻辑的原始论文是：

M Burrows, M Abadi and R Needham, *A Logic of Authentication*, DEC SRC Research Report 39. 也可见 ACM Transactions on Computer Systems, 8(1):18-36, February 1990. 该文不仅系统建立了 BAN 逻辑的概念和形式体系，而且有 2/3 的篇幅用 BAN 逻辑具体分析了一系列的重要协议，包括 Otway-Rees 协议、Needham-Schroeder 对称密钥协议、

Kerberos 协议、Wide-mouthed-frog 协议、Secure RPC 协议、Yahalom 协议、Needham-Schroeder 公钥协议、CCITT X.509 协议。该文至今仍然是初学者理解形式分析方法的一个有益的起点。习题 9-6 给出几个例子。

习 题

9-1 请画出 9.2.1 小节 Denning-Sacco 协议及那里所描述的攻击的完整的 Lamport 图。

9-2 如果对任何进程 p，私钥 d_p 与 p 之间、公钥 K_p 与 p 之间都是一一对应的关系（这是一个对实际公钥体制的合理假设），那么以上改正在第三条消息中采用消息项$\{\{B,K,T_a\}_{da}\}_{Kb}$就足够了，不必采用消息项$\{\{A,B,K,T_a\}_{da}\}_{Kb}$，为什么？如果采用消息项 $\{\{A,K,T_a\}_{da}\}_{Kb}$ 能行吗？为什么？

9-3 对 9.2.1 节中改正后的 SSL 协议，只要进程与其私钥是一一对应的,第三条消息也可以改为$\{CA, \{B, K, N_b\}_{da}\}_K$，效果与$\{CA, \{A,B,K,N_b\}_{da}\}_K$相同。但若仅仅改为$\{CA, \{B, N_b\}_{da}\}_K$或$\{CA, \{K, N_b\}_{da}\}_K$，你认为安全吗？

提示：若仅仅改为$\{CA, \{K, N_b\}_{da}\}_K$，9.2.1 节描述的对 SSL 的攻击仍有效。

9-4 以 9.2.2 小节介绍的记号表达 9.2.3 小节中对称 Needham-Schroeder 协议中三个进程 A、B、S 上的消息发送/接收序列。

9-5 以 9.2.2 小节介绍的记号表达 9.2.5 小节中 Bull 协议的三个进程 A、B、C 上的消息发送/接收序列。

9-5 看下面的 Zhou-Gollman 协议：

$A \rightarrow B$ ： L,C

$B \rightarrow A$ ： $sig_{sk(B)}(C)$

$A \rightarrow B$ ： $sig_{sk(A)}(K)$,K

$B \rightarrow A$ ： $sig_{sk(B)}(L,C,K)$

这是一个使消息接收方 B 向消息发送方 A 提供其已接收到消息的不可抵赖性证据的协议，对称加密钥 K 由 A 生成，并计算密文 $C=\{M\}_K$ 和散列 L=h(M)，K 在第三条消息中传递给 B，使 B 能验证 A 的第一条消息是一致的，最终 B 向 A 返回其接受到第一条消息的证据。

利用两轮并发协议进程 α 和 β，对 Zhou-Gollman 协议的一种攻击过程如下，为清楚起见，在协议消息旁标注出了该消息属于哪一轮进程；A/P 表示 P 伪装成 A 的身份进行操作，实际操作者是 P；x 是任意的字。

α.1 A/P→B ： $h(M),\{M\}_K$

α.2 B→A/P ： $sig_{sk(B)}(\{M\}_K)$

β.1 P→A ： x, K

β.2 A→P： $sig_{sk(A)}(K)$

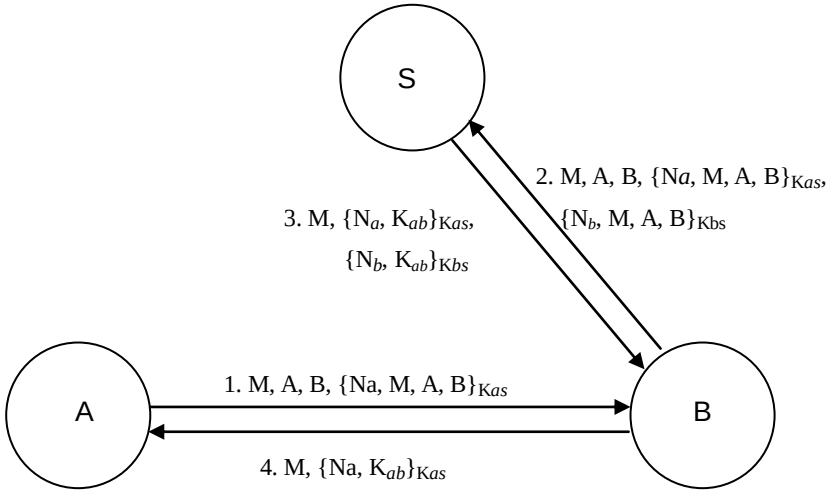
α.3 A/P→B ： $sig_{sk(A)}(K),K$

α.4 B→A/P ： $sig_{sk(B)}(h(M), \{M\}_K, K)$

请读者解释这一攻击的涵义。

9-6 本题给出几个 BAN 逻辑证明协议是否安全的例子，为此读者需要熟悉离散数学课程中学习过的一阶逻辑演算体系。

(1)著名的 Otway-Rees 协议的工作过程如下图所示，其中 A、B、S 分别是协议双方及仲裁服务器，K_{as}、K_{bs} 分别是 A、B 与 S 间的共享密钥，K_{ab} 是 S 生成的 A、B 间会话密钥，N_a、N_b 是随机数。



用逻辑方法分析/验证协议时需要给出一组用谓词公式表达的逻辑前提(assumptions)，相当于一阶逻辑演算体系中的公理；协议的安全目标(conclusion)也被表达为一组谓词公式，协议分析/验证的目的就是运用形式逻辑演算规则从前提推导出结论(安全目标)，如果推导成功则意味着在给定的前提下该协议是安全的。以下谓词 Believe(U,J)、Share(U,V,K)、Jurisdiction(U,P)和 Fresh(N)分别表示 U 信任命题 J、U 和 V 共享保密的对象 K、U 判定事实 P、N 被随机生成。对 Otway-Rees 协议，逻辑前提如下：

Believe(A, Share(A,S,K_{as}))
Believe(B, Share(B,S,K_{bs}))
Believe(S, Share(A,S,K_{as}))

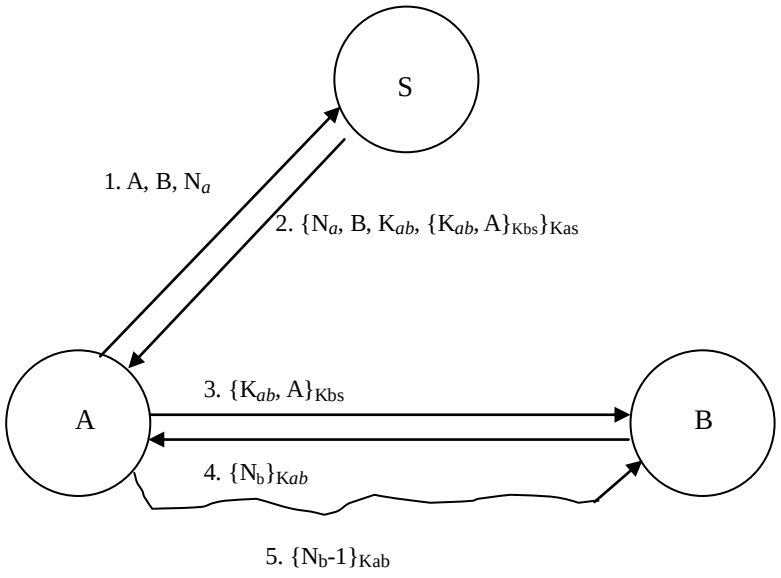
Believe(S, Share(B,S,K_{bs}))
Believe(S, Share(A,B,K_{ab}))
Believe(A, Jurisdiction(S, Share(A,B,K)))
Believe(B, Jurisdiction(S, Share(A,B,K)))
Believe(A, Jurisdiction(S, Sent(B, X)))
Believe(B, Jurisdiction(S, Sent(A, X)))
Believe(A, Fresh(N_a))
Believe(B, Fresh(N_b))
Believe(A, Fresh(N))

结论是

Believe(A, Share(A, B, Kab))
Believe(B, Share(A, B, Kab))

请应用逻辑演算规则[5]从前提推导出这里的结论，由此证明该协议安全。

(2) 基于对称密钥的 Needham-Schroeder 协议如下图，其中 A、B、S 分别是协议双方及仲裁服务器，K_{as}，K_{bs} 分别是 A、B 与 S 之间的共享密钥，K_{ab} 是 S 生成的用于 A、B 之间的会话密钥，N_a, N_b 是随机数。



前 提：

Believe(A, Share(A,S,K_{as}))
Believe(B, Share(B,S,K_{bs}))
Believe(S, Share(A,S,K_{as}))
Believe(S, Share(B,S,K_{bs}))
Believe(S, Share(A,B,K_{ab}))
Believe(A, Jurisdiction(S, Share(A,B,K)))
Believe(B, Jurisdiction(S, Share(A,B,K)))

[5] 可参考任何一本离散数学教程。

Believe(A, Jurisdiction(S, Fresh(Share(A, B, K)))
Believe(A, Fresh(N_a))
Believe(B, Fresh(N_b))
Believe(S, Fresh(Share(A, B, K_{ab}))

结 论：

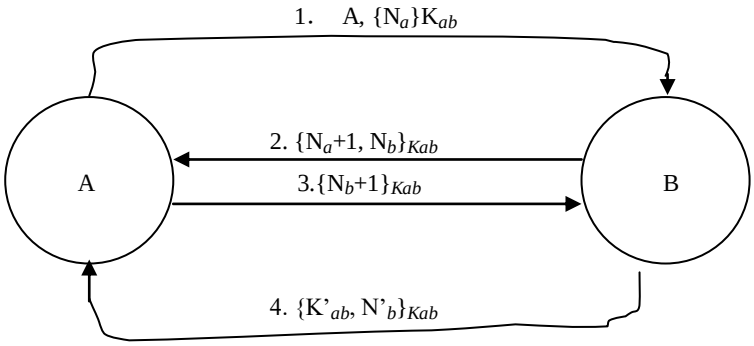
Believe(A, Share(A, B, K_{ab}))
Believe(B, Share(A, B, K_{ab}))
Believe(A, Believe(B, Share(A, B, K_{ab})))
Believe(B, Believe(A, Share(A, B, K_{ab})))

从以上前提导不出这里结论 Believe(B, Share(A, B, K_{ab}))，除非增加一条前提：

Believe(B, Fresh(Share(A, B, K))

请验证现在确实可以导出结论。但注意该前提并非本协议所内蕴，而是外加的。事实上，以上协议的第三条消息可以被攻击者重放以利用已被破译的旧会话密钥 K_{ab}，因此外加的前提 Believe(B, Fresh(Share(A, B, K))可以看做对存在该攻击的显式陈述。

(3) Andrew Secure RPC 协议如下图，其中 A、B 是协议双方，K_{ab} 是 A、B 间的共享密钥，K'_{ab} 是 B 生成的 A、B 之间会话密钥，N_a、N_b 是随机数， N'_b 是会话中的起始序号。



前 提：

Believe(A, Share(A, B, K_{ab}))
Believe(B, Share(B,A,K_{ab}))
Believe(A, Jurisdiction(B, Share(A,B,K))
Believe(B, Share(A,B,K'_{ab}))
Believe(A, Fresh(N_a))
Believe(B, Fresh(N_b))
Believe(B, Fresh(N'_b))

结 论：

Believe(B, Share(A, B, K'_{ab}))

在此逻辑体系中导不出结论 Believe(A, Share(A, B, K'_{ab}))，事实上步骤四确实可以被攻击者重放以利用已被破译的旧会话密钥 K'_{ab}。一种改进是将步骤四的消息改为{K'_{ab}，N_b',

$N_a\}K_{ab}$，由此将导致相应的前提条件 Believe(A, Fresh(N'$_b$))(为什么?)，从而(请验证)能证明结论 Believe(A, Share(A, B, K'$_{ab}$))。