密钥分配、密钥分割

裴士辉
QQ:1628159305

内容

- 口令
- 密钥生成
- 密钥分配
  - 公开密钥的分配
  - 共享密钥的分配
  - 会话密钥的分配
    - 通过公钥获得会话密钥
- 密钥分割

口令

口令的使用

- 现在有很多地方的安全性依赖于口令:
  - 银行**ATM**的口令；服务器注册口令；**Web**服务口令等
- 我们很少拥有密码学上的密钥，但是我们有很多口令，多到几乎记不住的程度
- 口令使用方便，而且历史悠久
- 防止字典攻击是取口令应该注意的地方

对口令的字典攻击

- 攻击者并不按照数字顺序去试所有可能的密钥，首先尝试可能的密钥，例如英文单词、名字等。（**Daniel Klein**使用此法可破译**40%**的计算机口令），

方法如下：

（**1**）用户的姓名、首字母、帐户名等个人信息

（**2**）从各种数据库得到的单词

（**3**）数据库单词的置换

（**4**）数据库单词的大写置换

（**5**）对外国人从外国文字试起

（**6**）尝试词组

口令的选择与密钥空间

	4字节	5字节	6字节	7字节	8字节
小写字母（26）	$4.6*10^5$	$1.2*10^7$	$3.1*10^8$	$8.0*10^9$	$2.1*10^{11}$
小写字母+数字（36）	$1.7*10^6$	$6.0*10^7$	$2.2*10^9$	$7.8*10^{10}$	$2.8*10^{12}$
字母数字字符（62）	$1.5*10^7$	$9.2*10^8$	$5.7*10^{10}$	$3.5*10^{12}$	$2.2*10^{12}$
印刷字符（95）	$8.1*10^7$	$7.7*10^9$	$7.4*10^{11}$	$7.0*10^{13}$	$6.6*10^{15}$
ASCII字符（128）	$2.7*10^8$	$3.4*10^{10}$	$4.4*10^{12}$	$5.6*10^{14}$	$7.2*10^{16}$
8位ASCII字符（256）	$4.3*10^9$	$1.1*10^{12}$	$2.8*10^{14}$	$7.2*10^{16}$	$1.8*10^{19}$

口令的选取方法

- 想出一个可记忆的短语，如：

  "over the river and through the woods, to grandmother's house we go."

- 然后只引用第一个字母而把它变成简写（包括标点）。

  otrattw,tghwg.

- 把简写中的字母替换成数字和符号来增加其复杂性。例如，用 **7** 来替换 **t**，用 **@** 来替换 **a**：

  o7r@77w,7ghwg.

- 至少把一个字母变成大写来增加其复杂性，如 **H**。

  o7r@77w,7gHwg.

密钥的生成

ANSI X9.17 PRG

令$E:\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$表示一个分组密码，定义算法：

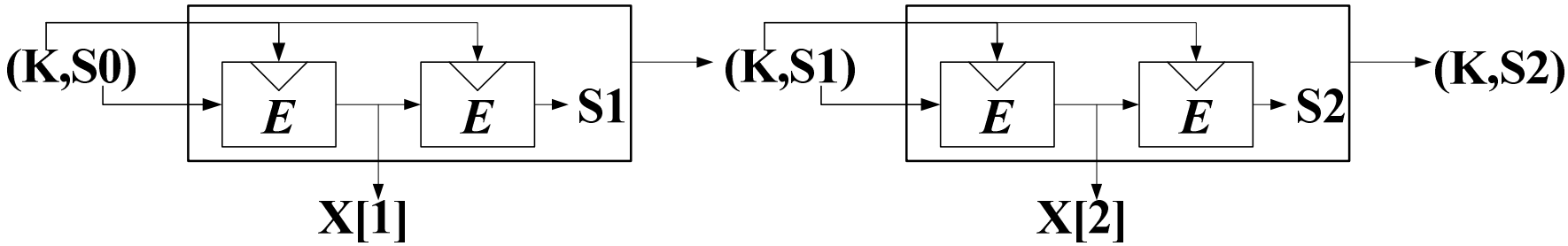
algorithm $G(St)$

$(K,S) \leftarrow St$

$X \leftarrow E_K(S)\ ;\ S \leftarrow E_K(X)$

return $(X,(K,S))$

- 状态为(K,S)，K是加密的密钥，$S \in \{0,1\}^n$
- 初始状态为(K,S)，其中$K \xleftarrow{\$} \{0,1\}^k, S \xleftarrow{\$} \{0,1\}^n$



其他标准

- **FIPS-186**定义了基于**DES**和**SHA-1**的**PRGs**
- **NIST SP 8000-90**定义了基于**hash**、**HMAC**、**CTR**、**ECC**的生成器
- **ANSI X9.31**和**ANSI X9.62**
- **OpenSSI**定义了基于**SHA-1**的**PRG**

完整的过程

- 在实际中,随机数生成器**(RGN: random numbergeneration)**包括两个部分：
  - 种子
  - **PRG**
- **RGN**失败的例子很普遍：**Netscape, Debian Linux,** 等等。失败既包括种子，也包括**PRG.**
- 原则上，可以设计出好的**PRG.**
- 但是种子的选取是一个问题。
- 典型的方法：
  - 维护一个熵池（**entropy pool**）
- 可选的方法：基于硬件的**RNG**

公开密钥的分配

公开密钥

Alice		$Bob^{pk[A]}$
$x \leftarrow D_{sk[A]}(y)$	$\xleftarrow{y}$	$y \leftarrow E_{pk[A]}(x)$
$\sigma \leftarrow Sig_{sk[A]}(m)$	$\xrightarrow{m,\sigma}$	$Ver_{pk[A]}(m,\sigma)$

只要获得 **Alice** 的公开密钥 $pk[A]$

Bob 可以：

- 向 **Alice** 发送密文
- 验证 **Alice** 的签名

那么：

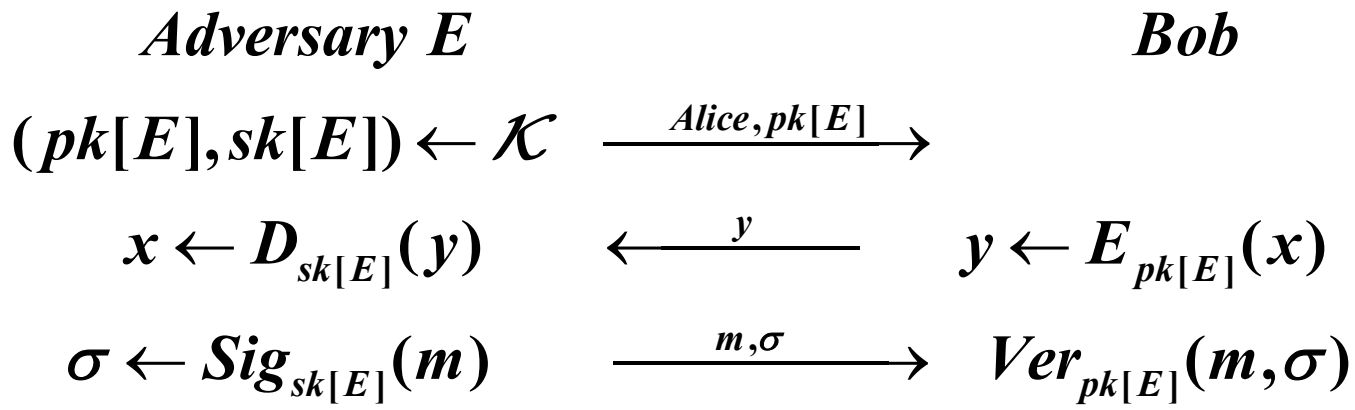
如何获得 **Alice** 的公开密钥 $pk[A]$?

公开密钥的分配

这样如何：

Alice		*Bob*
$(pk[A], sk[A]) \leftarrow \mathcal{K}$	$\xrightarrow{Alice, pk[A]}$	
$x \leftarrow D_{sk[A]}(y)$	$\xleftarrow{y}$	$y \leftarrow E_{pk[A]}(x)$
$\sigma \leftarrow Sig_{sk[A]}(m)$	$\xrightarrow{m, \sigma}$	$Ver_{pk[A]}(m, \sigma)$

中间人攻击



Bob 传给 **Alice** 的密文可以被攻击者 **E** 得到并解密
攻击者 **E** 同时可以伪造 **Alice** 的签名

PKI中的权威性

- **Bob** 需要 **Alice** 的公钥是来自权威的拷贝
- **PKI（Public Key Infrastructure）**负责保证公钥的权威性
- 通常是通过数字证书来实现

数字证书的过程

- **Alice** 生成公钥 ***pk***，然后传递给 **CA**
- **CA** 检查并核实 **Alice** 的身份的真实性
- **Alice** 向 **CA** 证实其知道私钥 ***sk***
- **CA** 向 **Alice** 颁发数字证书
- **Alice** 将数字证书传递给 **Bob**
- **Bob** 验证数字证书，并获得 **Alice** 的公钥 ***pk***

生成公钥并传递给CA

密钥的生成：**Alice** 在本地生成她的密钥对

$$(pk, sk) \leftarrow \mathcal{K}$$

发往 **CA**：　**Alice** 传递给 **CA** 如下信息：

$$(Alice, pk)$$

身份的检查与核实

在收到(*Alice*, *pk*)之后，**CA** 需要进行检查，以证实 ***pk*** 确实是 **Alice** 的：

- 和 **Alice** 通电话进行核实
- 检查相关的证件

上述检查是带外（**out-of-band**）进行的

Alice向CA证实其知道私钥

CA 可以通过如下方法知道 Alice 拥有与公钥 ***pk*** 对应的私钥 ***sk*** ：

- 验证 Alice 的签名
- 用其公钥 ***pk*** 加密某些信息，让 Alice 解密

上述方法保证 Alice 不是拷贝他人的密钥

CA向Alice颁发数字证书

一旦 CA 确信公钥 pk 是属于 Alice 的，CA 生成 Alice 的数字证书：

$$CERT_A = (CERTDATA, \sigma)$$

其中 σ 是 CA 对于 $CERTDATA$ 的签名，使用的是 CA 的私钥 $sk[CA]$。

$CERTDATA$：

- pk，$ID(Alice)$
- CA 的名称
- 证书的有效日期
- 使用范围
- 安全级别
- ……

然后把证书 $CERT_A$ 传递给 Alice

数字证书的使用

Alice 把证书$CERT_A$传递给 **Bob, Bob** 接到证书之后：

- $(CERTDATA, \sigma) \leftarrow CERT_A$
- 使用 **CA** 的公钥$pk[CA]$对签名进行验证：

$$Ver_{pk[CA]}(CERTDATA, \sigma) = 1$$

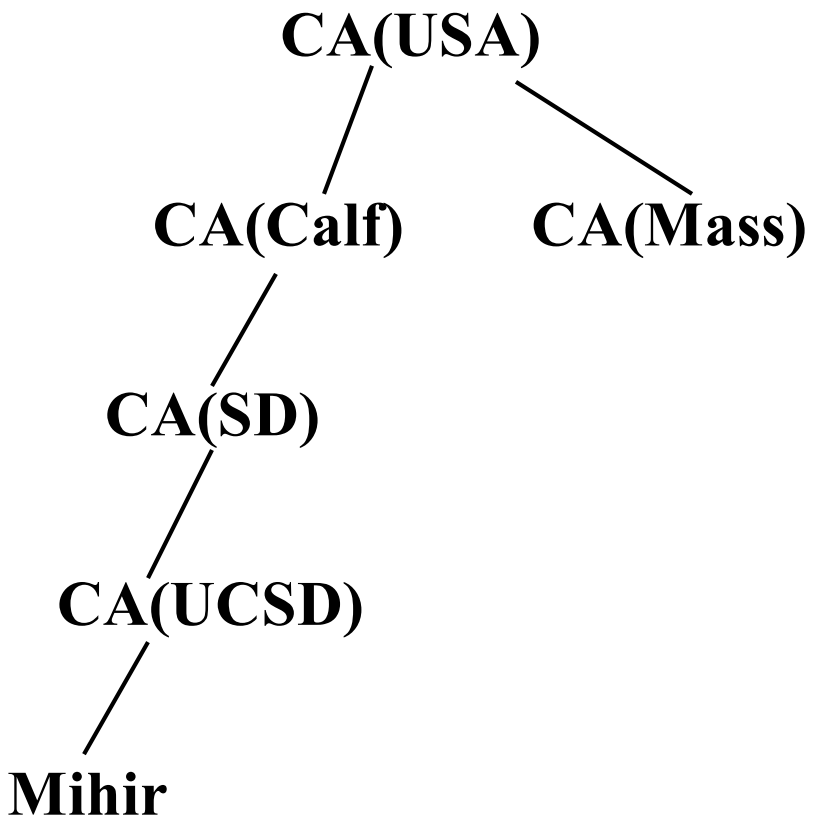
- $(pk, Alice,$ 有效日期，$\cdots) \leftarrow CERDATA$
- 检查证书是否过期
- ……

上诉过程通过，就可以使用证书了。

Bob如何得到pk[CA]

CA的公钥一般嵌入到软件之中，例如浏览器

证书链



$\mathbf{CERT}_{Mihir}$

$\mathbf{CERT[CA(USA):CA(Calf)]}$

$\mathbf{CERT[CA(Calf):CA(SD)]}$

$\mathbf{CERT[CA(SD):CA(UCSD)]}$

$\mathbf{CERT[UCSDCA):}\ \boldsymbol{Mihir}\ \mathbf{]}$

$$CERT[X:Y]=(pk[Y],Y,\cdots,Sig_{sk[X]}(pk[Y],Y,\cdots))$$

为了验证$CERT_{Mihir}$，Bob 只需要$pk_{CA[USA]}$。

为什么使用证书链

- 对于如下工作：
  检查**Milir**的身份并向**Milir**颁发证书
  CA(UCSD)比**CA(USA)**更适合，因为：
  Milir是**UCSD**的教授，并且**UCSD**掌握了很多**Milir**的相关信息。
- 分布式的身份检查和证书发放可以减少单独**CA**的工作量
- 浏览器只需要嵌入很少的公钥（例如只需要根**CA**即可）

证书的撤销

如果 **Alice** 想要撤销她的证书$CERT_A$，可能因为她的私钥暴露了。

- **Alice** 传递$CERT_A$和撤销申请给 **CA**
- **CA** 确认撤销申请确实是 **Alice** 提出的
- **CA** 标志$CERT_A$为撤消状态

证书撤销列表

- **CRLs: Certificate revocation lists**
- **CA 维护了一个 CRL, 其条目具有如下形式：**

  （*CERT*，撤销日期）
- 列表随时发布，任何人可以查询
- 在 **Bob** 接受 **Alice** 的证书之前，他需要检查一下 **CRL**

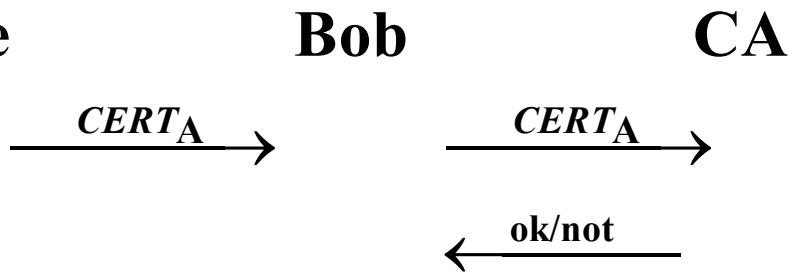
例如

- **11 月 22 日：Alice 的私钥暴露了**
- **11 月 24 日：Alice 的数字证书$CERT_A$撤消了**
- **11 月 25 日：Bob 看到了 CRL**

在 **11 月 22-25** 日期间，证书可能被使用，**Bob** 可能接受了攻击者的签名；另外 **Bob** 发往 **Alice** 的密文可能被攻击者解密。

OCSP

OCSP: On-line Certificate Status Protocol
OCSP 支持在线查询证书是否已经撤消了



但是在线验证违背了公钥密码的初衷。

实际中的证书撤销问题

- **VeriSign** 估计撤销证书的比率为**20%**
- 实际中**CRLs**是巨大的

证书撤销问题是一个大问题，它影响了**PKI**的推广以及公钥密码的使用

共享密钥的分配

共享秘钥

$$\begin{array}{ccc} Alice^K & & Bob^K \\ M \leftarrow D_K(C) & \xleftarrow{C} & C \leftarrow E_K(M) \\ T \leftarrow MAC_K(M) & \xrightarrow{M,T} & Ver_K(M,T) \end{array}$$

Alice 和 **Bob** 使用共享密钥：

- 互相发送加密数据
- 验证彼此的 **MACs**

和公开密钥相比，共享秘钥的优势在于计算速度

但是，如何获得共享秘钥？

Diffie-Hellman密钥协商方案

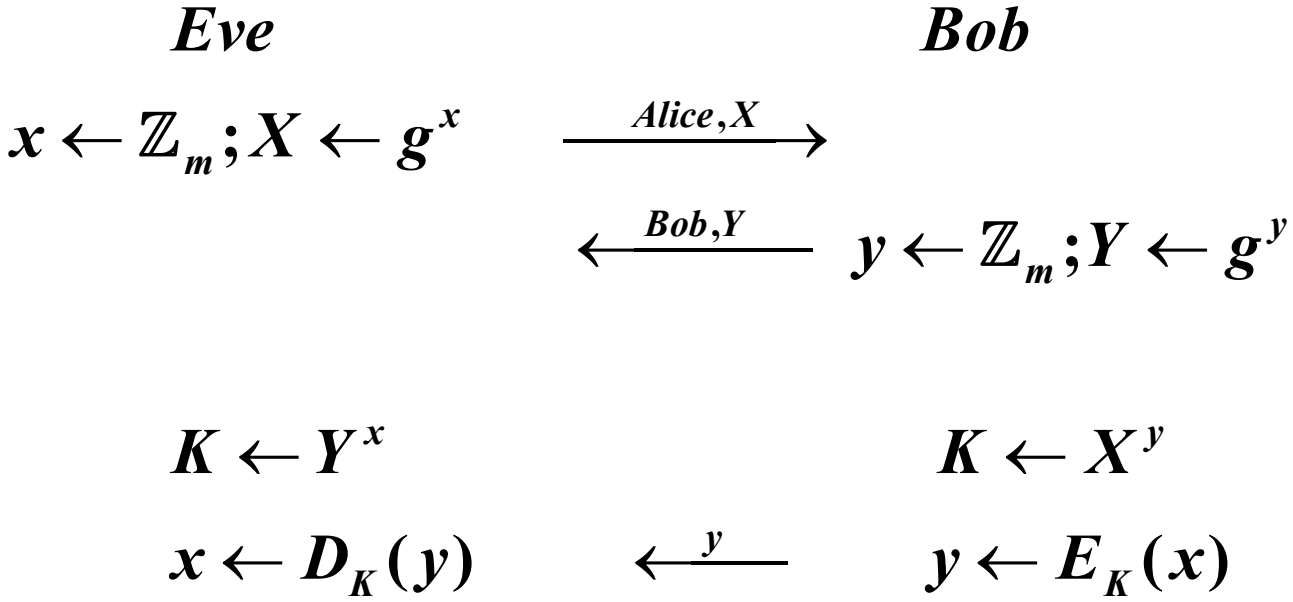
令$G =< g >$表示阶为m的循环群，其中(G, g, m)为公共参数。

Alice		*Bob*
$x \leftarrow \mathbb{Z}_m; X \leftarrow g^x$	$\xrightarrow{Alice, X}$	
	$\xleftarrow{Bob, Y}$	$y \leftarrow \mathbb{Z}_m; Y \leftarrow g^y$
$K \leftarrow Y^x$		$K \leftarrow X^y$

$$Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$$

该方案可以在 **Alice** 和 **Bob** 之间建立共享秘钥。该密钥之后可以用来加密消息，生成 **MAC.**

对DH密钥协商方案的中间人攻击



中间人 **Eve** 可以假冒 **Alice,** 这样：

- **Bob** 认为他与 **Alice** 共享秘钥 ***K***，实际上确是 **Eve** 拥有 ***K***
- **Bob** 传给 **Alice** 的密文，**Eve** 可以解密

结论：**DH** 密钥交换方案在主动攻击下式不安全的。

什么时候使用密钥协商方案？

如果存在中间人攻击，**Alice** 和 **Bob** 不可能做到：

- 从协议开始执行到结束的整个过程
- 通过交换信息得到攻击者不知道的共享秘钥

为什么？

因为 **Alice** 和 **Bob** 没有方法区分是对方还是中间人攻击者。

怎么办？

Alice 和 **Bob** 在执行密钥交换方案之前，应该具有中间人攻击者不知道的信息。即长期密钥。

长期密钥

- 公开密钥：A 有公开密钥 pk_B, B 有公开密钥 pk_A
- 对称密钥：A, B 共享一个密钥 K
- 三方密钥：S 为可信的第三方服务器

  A, S 共享秘钥 K_A

  B, S 共享秘钥 K_B

会话密钥 (session keys)

在实际中，**Alice** 和 **Bob** 之间会发生多次通信“会话”。在每次会话过程中，**Alice** 和 **Bob** 需要做如下工作：

- 首先基于他们的长期密钥，执行会话密钥分配协议，得到一个新的会话密钥
- 在会话过程中，使用会话密钥，进行通信过程中加密和消息认证。

会话密钥分配

- 有非常多的协议
- 数十个安全需求
- 很多个破解的协议
- 会话密钥分配方案容易定义，得到正确的却很难
- 使用的非常多：**SSL, TLS, SSH,** …

为什么使用会话密钥

- 在效率方面，和长期密钥相比，速度更快
- 从安全方面的考虑，不同的应用使用不同的密钥，在一个应用中不会获得其他应用的信息

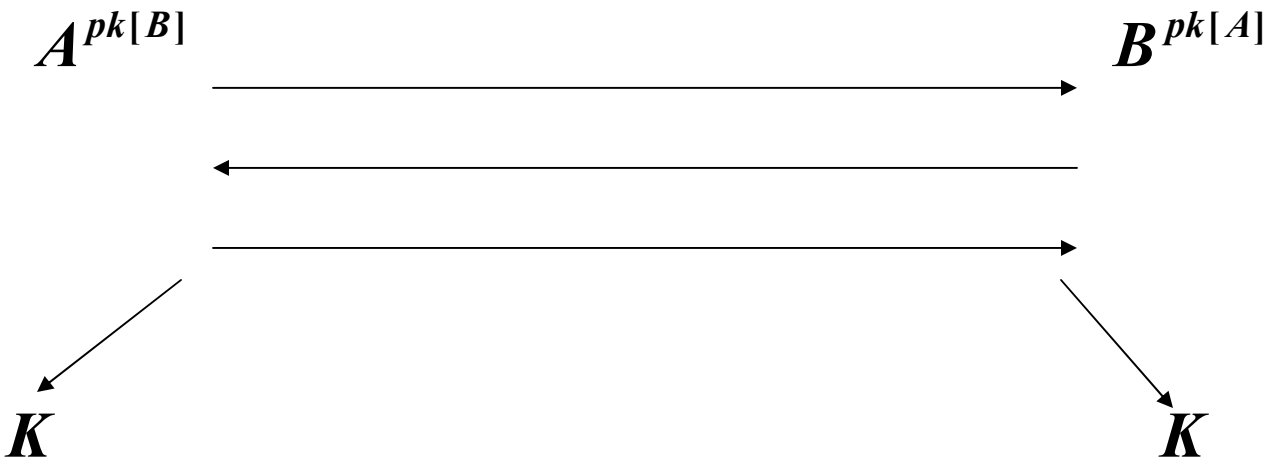
会话密钥基本的安全要求

一方可能同时处在很多不同的通信会话之中

- 一个会话中的会话密钥不会暴露其他会话密钥的信息

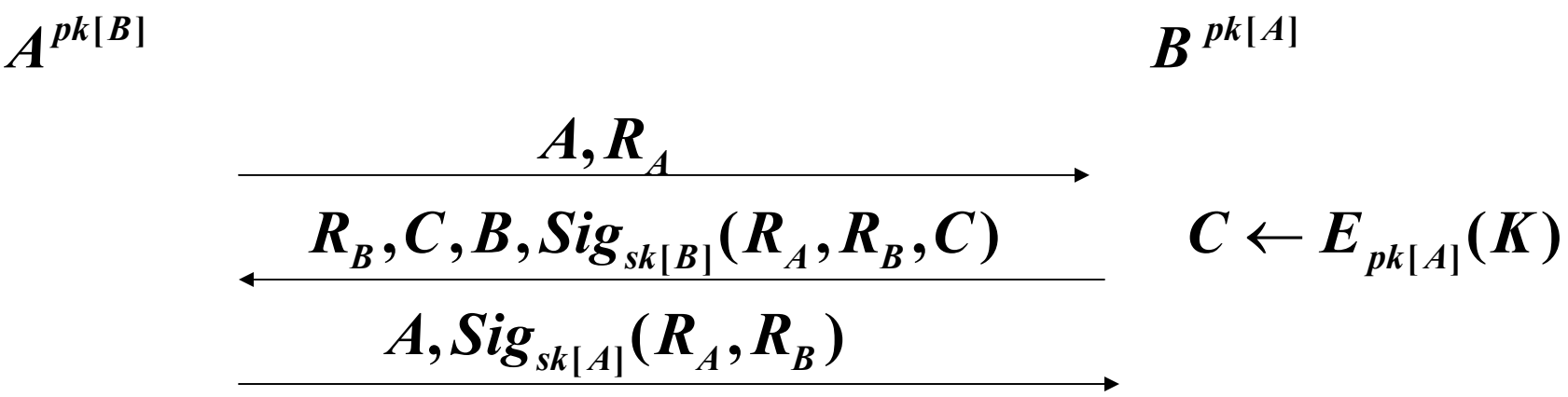
通过公开密钥分配会话密钥

通过公开密钥分配会话密钥



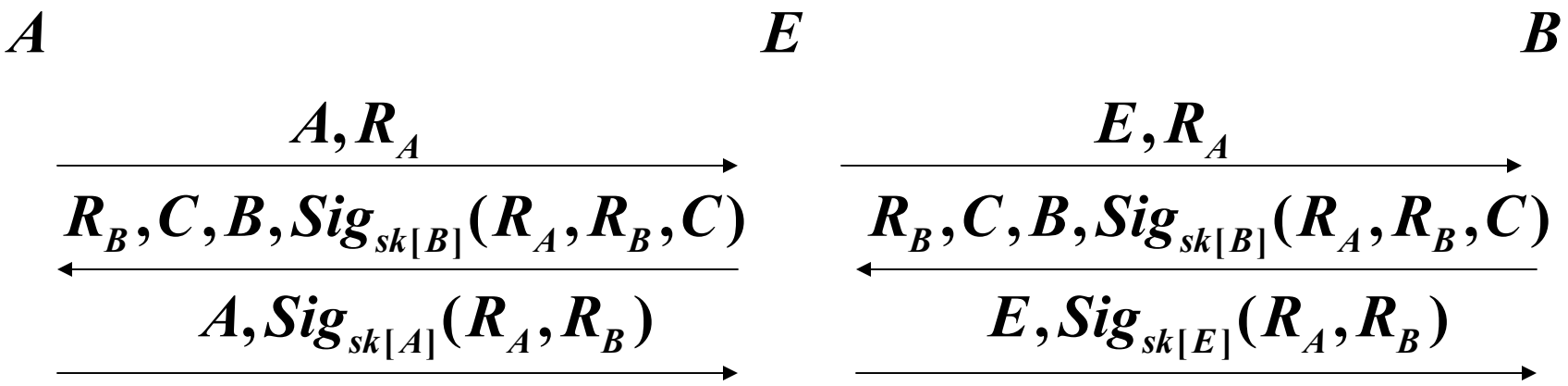
在实际中这是最广泛的会话密钥交换方式，在所有如下的安全通信协议中都使用了这种方式：**SSL，SSH，TLS，IPSEC，802.11**等。

KE1方案



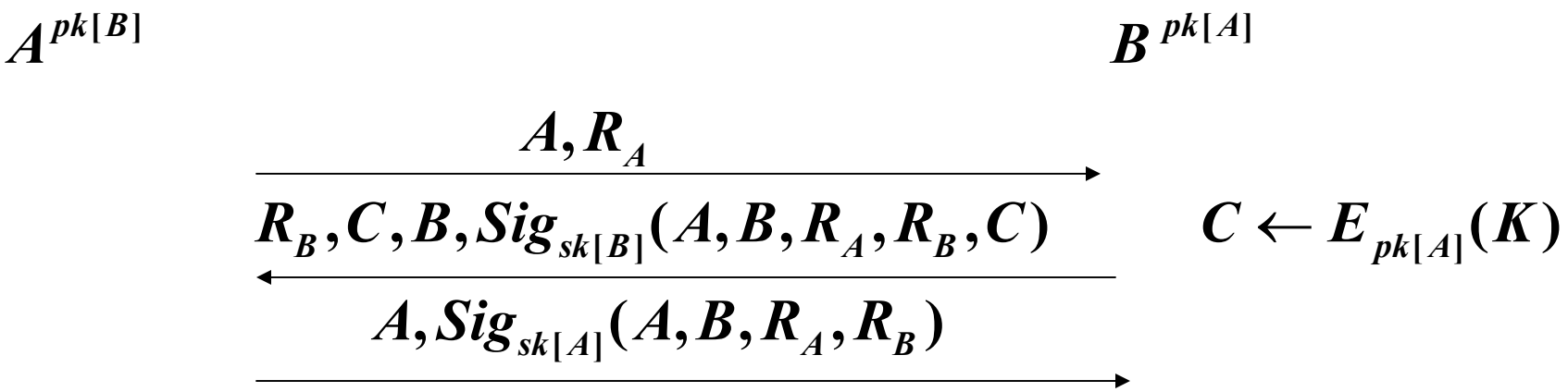
- 会话密钥K是由B生成的
- R_A, R_B是随机的 nonce

绑定(binding)攻击



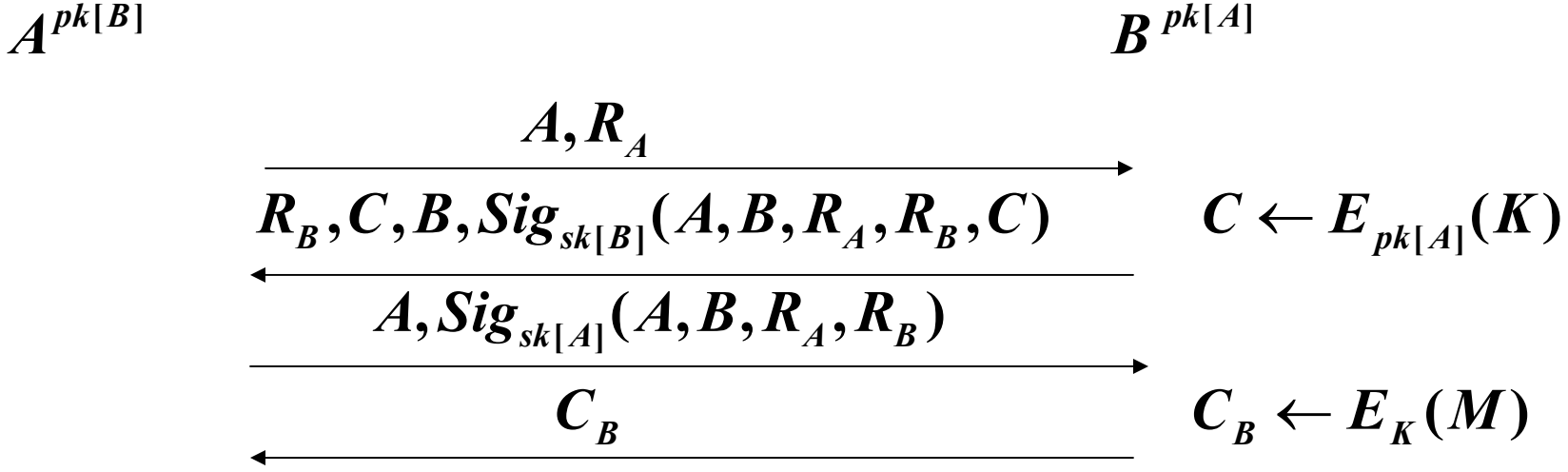
- A 认为他和 B 共享会话密钥K'，B 认为他和 E 共享会话密钥K；虽然 E 不知道会话密钥K'，但通常视之为一个问题
- 虽然目前还没有该问题的危害性的恰当的例子。

KE2方案



- 把身份信息也加入到签名的输入之中，阻止了绑定攻击

前向安全(forward secrecy)

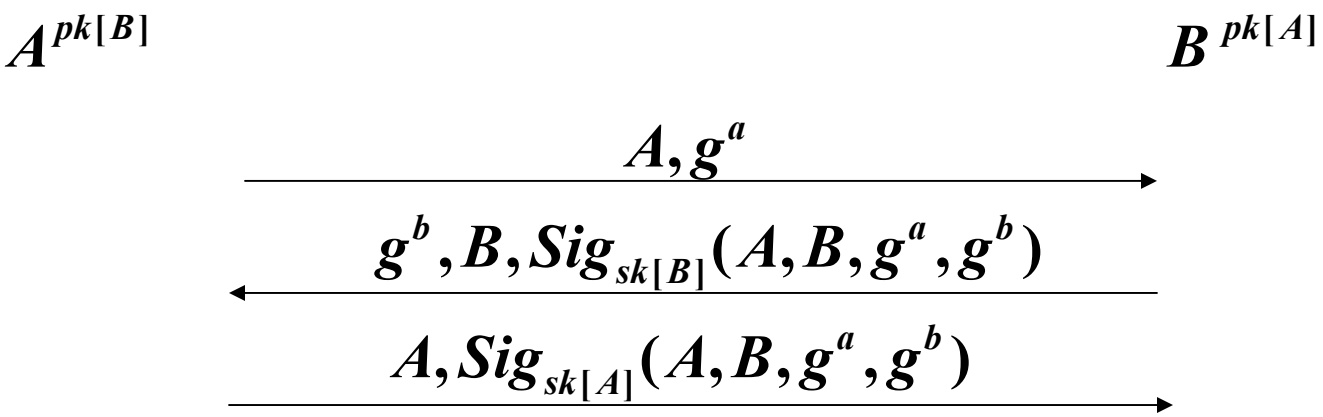


- **10** 月 **10** 日：攻击者 **E** 记录了上述的传递过程
- **11** 月 **18** 日：**A** 的系统被攻破，**A** 的私钥$sk[A]$暴露
- **11** 月 **19** 日：**A** 撤销了他的数字证书，进一步的危害不会发生、但不会阻止攻击者 **E**：

$$K \leftarrow E_{sk[A]}(C);\quad M \leftarrow D_K(C_B)$$

如何获得前向安全性：即使$sk[A]$暴露了，之前的通信安全不受影响？

KE3方案



会话密钥是$K = H(A, B, g^a, g^b, g^{ab})$

即使攻击者获得 A 的私钥$sk[A]$也无助于得到会话密钥，从而保证了前向安全性。

因为这里只有签名过程，没有使用公钥的加密过程。

所有标准中的 DH 方案都具有前向安全性。

秘密共享(秘密分割、密钥分割)

问题的提出

问题的提出

- 假设 **Coca-Cola**公司的董事会想保护可乐的配方. 该公司总裁应该能够在需要时拿到配方，但在紧急的情况下，**12**位董事会成员中的任意**3**位就可以揭开配方。

(k，n)秘密分割门限方案

已知秘密S，将其分割成n个元素$S_1, S_2, \ldots, S_n$

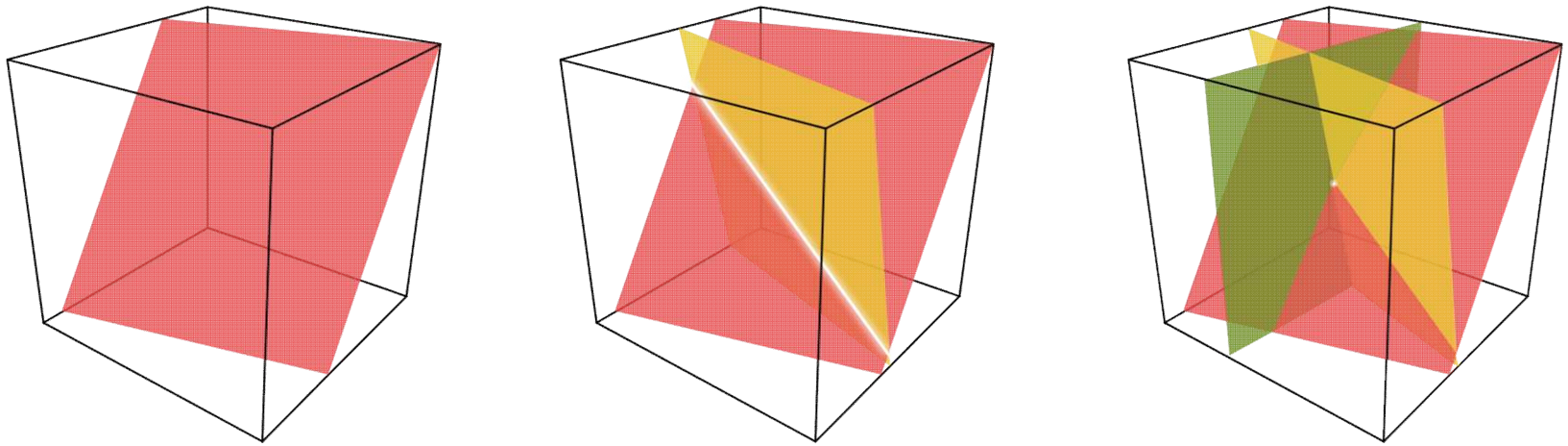
1.拥有大于等于k个不同的S_i，可以重构秘密S；

2.拥有小于k个不同的S_i，不能重构秘密S。

k称为门限值，S_i称为子密钥或影子。

秘密分割方案由 **Adi Shamir** 和 **George Blakley** 在 **1979** 分别提出。

(3，n) Blakley秘密分割门限方案



Blakley's scheme in three dimensions: each share is a plane, and the secret is the point at which three shares intersect. Two shares are insufficient to determine the secret, although they do provide enough information to narrow it down to the line where both planes intersect.

Shamir秘密分割门限方案

依据：2 个点可以构建直线，3 个点可以构建抛物线，4 个点可以构建 3 次曲线……，k个点可以构建$k-1$多项式。

假设要构建(k,n)门限方案，其中共享秘密为S，假设$S \in \mathbb{Z}_p$，

$0 < k \le n < p, S < p$， p为素数。

随机选择$k-1$个正整数$a_1,\cdots,a_{k-1} \in \mathbb{Z}_p$，令$a_0 = S$。由此构建多项式：

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

秘密的分割：

每个参与者分配一个点$(i, f(i))$

问题

- 假设部队中一个班 **10** 个人，其中士兵 **8** 个人，副班长 **1** 人，班长 **1** 人
- 假设一件事情，
  4 个士兵可以决定
  或者 **2** 个士兵加上副班长可以决定
  或者班长可以决定
- 请问如何设计密钥共享方案？