第十章 身份认证协议

这一章阐述身份认证(或称身份鉴别)协议，10.1 和 10.2 节讨论这类协议的普遍特点和目前被实际应用的典型实例，这些协议实例都属于公钥密码类型；10.3 节讨论身份认证协议与曾经阐述过的数字签名方案(回顾 8.2 节)之间的一个有趣的关系，即著名的 *Fiat-Shamir* 变换；第 10.4 节阐述一类基于对称密码方案的身份认证协议，即广泛应用的 *Kerberos* 协议。

10.1 概 念

身份认证协议用于在网络环境中主体之间彼此证实对方的身份，协议双方可以是用户、机器、进程等实体。直观地说，一方面，身份认证协议保证使合法的协议双方[1]彼此确信对方确实是其所声称的确那个实体，另一方面，任何非法的主体(即攻击者，他们缺少合法协议方所持有的某些秘密信息)不能成功地冒充任何合法的主体而又不被协议的合法参与方所发现。因此，身份认证协议安全性涵义的实质是抗身份欺诈。

现在更精确地描述基本概念。实际应用最广泛的身份认证协议多数都是所谓 *Fiat-Shamir* 型身份认证协议，这类协议 SI=(KG, P, V) 用 P.P.T.算法 K、P 和确定性算法 V 以及复杂性参数 k 定义。KG 是密钥生成算法，以 k 为输入并输出公钥/私钥偶(pk, sk)；P 是身份证实算法，V 是身份验证算法，P 以私钥/公钥偶为输入参数、V 则仅以公钥 pk 为输入参数。P、V 都是有状态的过程，状态分别是 P、V 的私有信息，各自的初始状态分别为 P、V 的输入参数。P、V 每接收一条消息 M_{in} 后进行计算并发送一条新消息 M_{out}，这一行为分别表达为(σ_P, M_{out})←P(σ_P, M_{in})和(σ_V, M_{out})←V(σ_V, M_{in})。不失一般性，总假设由 P 发起协议的第一条消息[2]，V 在最后的步骤中不发送消息而是输出一个判定元素 d，d=1 表示接受 P 的身份(协议成功)、d=0 表示拒绝 P 的身份(协议失败)。所有算法还满足一致性关系。

实际应用中的 *Fiat-Shamir* 型身份认证协议都具有三消息形式，这时以上描述的协议过程可以用下面的图 10.1 来表达。

[1] 实际应用中也存在多方身份认证协议，但这类协议比较复杂，本教程不予讨论。
[2] 若 V 发起第一条消息 M_1，则可以将协议的“第一条消息”看做空消息 P→V: ε, V→P: M_1←V(St_V,ε)。

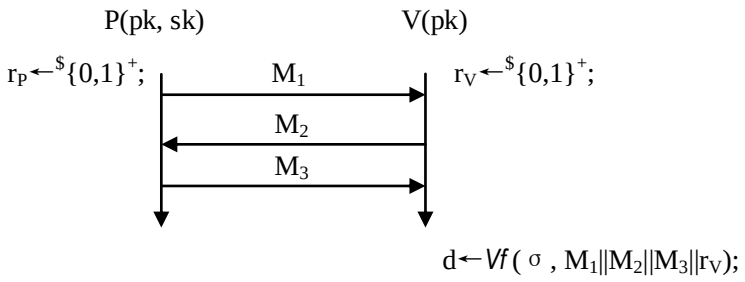


图 10-1 Fiat-Shamir 型身份证认证协议的普遍结构

针对身份认证协议存在三类攻击：被动攻击、主动攻击和并发攻击。这些攻击的能力依次增强，攻击者的目标都是要能成功地欺骗诚实的身份验证方 V，即实现身份欺诈。*被动攻击者*仅仅收集诚实的身份证实方 P 与诚实的身份验证方 V 之间的协议会话记录，由此推断有助于身份欺诈的信息。*主动攻击者*首先伪装成合法的身份验证方 V，与诚实的身份证实方 P 交换协议消息，套取需要的信息，然后再假冒某个合法的身份证实方与诚实的身份验证方进行协议会话，以欺骗该验证方。主动攻击者在实施身份欺诈之前可以与许多合法的证实方进行协议会话，但*每次只能与一个证实方会话*，这些会话在时间上*不能交错*，因此会话可能不完整。*并发攻击者*的策略与主动攻击者类似，即分为信息套取和身份欺诈两个阶段，所不同的是，在实施身份欺诈之前并发攻击者可以*同时与多个合法的证实方进行协议会话*(即并发攻击者在信息套取阶段的会话是并行的，可以同时保持多个会话)。图 10-2 是对这些直观概念的图形表达(实际上还有第四类、也是能力最强的攻击，即状态重置攻击，但我们不对此进行描述)。

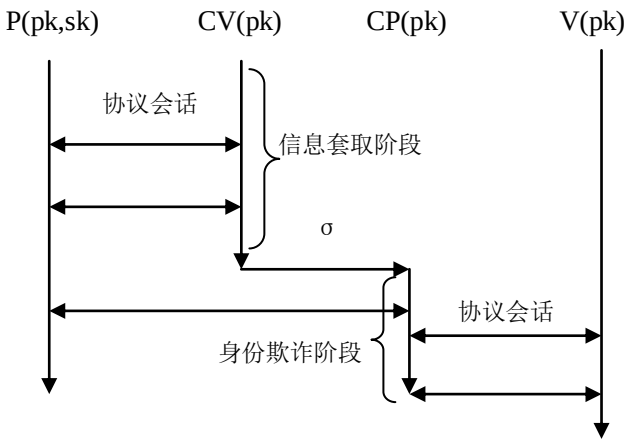


图 10-2 对 Fiat-Shamir 型身份认证协议的攻击模型

10.2 协议实例

目前已开发了许多身份认证协议，例如第 9 章讨论的 *Needham-Schroeder* 协议和 *Woo-Lam* 协议等都属于这类协议(当然它们都是设计不当的例子)。这一节阐述两个被广泛应用的身份认证协议，即著名的 *Schnorr* 协议和 *Feige-Fiat-Shamir* 协议。这两个协议发表于 80 年代末，不仅都被严格证明具有上一节意义上的抗身份欺诈性质，而且计算效率都很高，例如 *Schnorr* 协议因此而广泛应用于智能卡领域。

10.2.1 Schnorr 协议

q 是 *k* 位素数，G 是 q 阶循环群，g 是 G 的生成子(因此 G 的元素是 $g^0=e$、g、g^2、g^3、…、g^{q-1})，并且所有这些对象都公开。在 $Z_q=\{0,1,2,\ldots,q-1\}$[3]中随机选取一个数 x，然后计算 $y\leftarrow^{\$}g^{-x}$，以 y 和 x 分别作为合法的主体 P 的公钥和私钥。协议的会话过程如图 10-3。

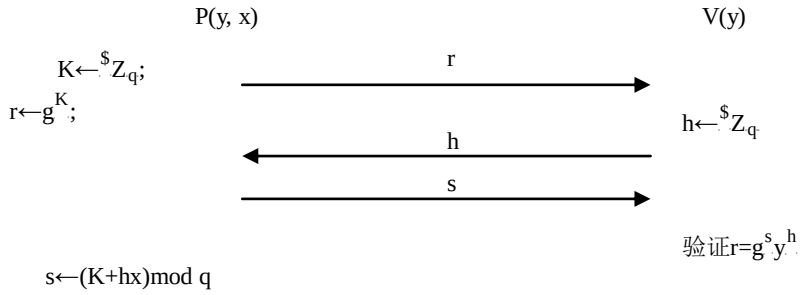


图 10-3 *Schnorr* 身份认证协议

当P需要向V证实自身的身份，即向V证明自己确实持有公钥y所对应的私钥x，P生成随机数K、计算$r\leftarrow gP^{KP}$并向V发送r；V生成并向P发送随机数h；P在接收到h后计算出整数$s\leftarrow(K+hx)\bmod q$并向V发送s；V在接收到s后验证$r=g^sy^h$是否成立，若成立则接受对方确实是P，否则判定对方是冒充者。

不难验证，若 P 确实持有公钥 y 所对应的私钥 x，则 $gP^{sP}yP^{hP}=gP^{sP}(g^{-x})P^{hP}=gP^{sP-x}P^{hP}=gP^{KP-qnP}=gP^{KP}=r$，其中n是某个整数，$gP^{qP}=1$，这意味着诚实的主体P总能够向V证实自己的身份。另一方面，对冒充的主体，已经证明若群G上的离散对数问题难解(离散对数问题见上一章例 8-1 中的说明)，则*Schnorr*协议抗主动身份欺诈。最

[3] 有些读者可能会认为应该是集合 $Z_q^*=\{1,2,\ldots,q-1\}$，理由是 x=0 不适合于用做私钥。但实际上在 Z_q 中随机取到 0 的概率只有 1/q，例如对 512 位素数 q 这概率只有 2^{-512}，因此用 Z_q 也无妨。

后注意*Schnorr*协议所涉及的计算无非是剩余运算(各种*Euclid*算法)和幂运算(一个通用快速算法见习题 8-25)，效率是很高的。

10.2.2 Feige-Fiat-Shamir 协议

p、q 是 *k* 位秘密素数，N=pq，N 公开但 p、q 保密。在{1,2,…,N-1}上随机生成一个数 $x\leftarrow^{\$}\{1,2,\ldots,N-1\}$，计算 $y\leftarrow^{\$}x^2 \bmod N$，将 y 和 x 分别作为合法的主体 P 的公钥和私钥。协议的会话过程如图 10-4。

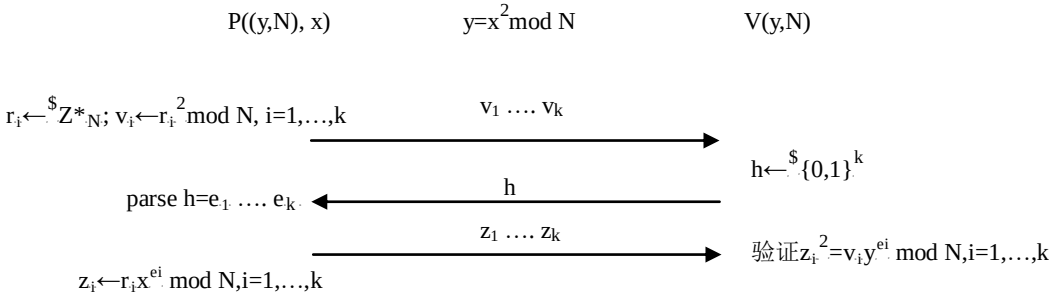


图 10-4 *Feige-Fiat-Shamir* 正则身份认证协议

当P需要向V证实自身的身份，即向V证明自己确实持有公钥y所对应的私钥x，P生成一组k个随机数$r_i\leftarrow^{\$}Z_N$, K、计算$v_i\leftarrow^{\$} r_i^2 \bmod N$并向V发送$v_i$, i=1,…,k；V生成并向P发送k位随机数h；P在接收到h后计算出一组k个整数$z_i\leftarrow r_i x^{e_i} \bmod N$，其中$e_i$是h的第i位(因此实际上$z_i=r_i$或$xr_i \bmod N$), i=1,…,k 并向V发送$z_1\ldots. z_k$；V在接收到$z_1\ldots. z_k$后验证 $z_i^2 = v_i y^{e_i} \bmod N$ 是否对每个i=1,…,k 都成立，若全部成立则接受对方确实是P，否则表明对方是冒充者。

不难验证，若P确实持有公钥y所对应的私钥x，则对每个i=1,…,k 都有 $v_i y^{e_i} = r_i^2 x^{2e_i} = z_i^2 \bmod N$，这意味着诚实的主体P总能够向V证实自己的身份。另一方面，对冒充的主体，已经证明若N=pq型的素因子分解问题难解则*Feige-Fiat-Shamir*协议抗主动身份欺诈。最后注意这一协议所涉及的计算仅是剩余运算(各种*Euclid*算法)和平方运算，效率很高。

10.3* 与数字签名方案的关系：*Fiat-Shamir* 变换

身份认证协议和数字签名方案的根本目的都是抗伪造，前者抗伪造的具体涵义是使攻击者不能有效实施身份欺诈，后者抗伪造的具体涵义是使攻击者不能有效生成一个合法主体

对新消息的数字签名。两者之间存在一个精确的联系，使得从身份认证协议可以导出一个数字签名方案，并且若前者抗被动身份欺诈则后者一定具有数字签名意义上的抗伪造性质(参见 8.2.2 节)，这就是著名的 *Fiat-Shamir* 变换。反之亦然，给定一个数字签名方案，可以导出一个对应的身份认证协议，若前者具有数字签名意义上的抗伪造性质则后者一定抗被动身份欺诈。

为精确表达 *Fiat-Shamir* 变换，在此先将 *Schnorr* 协议和 *Feige-Fiat-Shamir* 协议这类身份认证协议统一为一大类普遍形式，这就是所谓正则身份认证协议。正则身份鉴别协议 ID=(KG, P, V, C)是一类高效的三消息身份鉴别协议，包括一组 P.P.T.算法 KG 和 P、确定性算法 V 以及复杂性参数 k 的函数 C，KG 是密钥生成算法，以 k 为输入并输出公钥/私钥偶(pk, sk)；P 是身份证实算法，V 是身份验证算法；P 以私钥/公钥偶为初始输入、V 以公钥 pk 为初始输入； C 表示协议第 2 步所生成的随机串的长度。协议的会话过程如图 10-5，所有算法还满足一致性关系：对任意的 k 恒有
$P[(pk,sk)\leftarrow KG(k); (Cmt,St)\leftarrow P(pk,sk); Ch\leftarrow^{\$}\{0,1\}^{C(k)}; Rsp\leftarrow P(Ch,St): V(pk, Cmt||Ch||Rsp)=1]=1$。

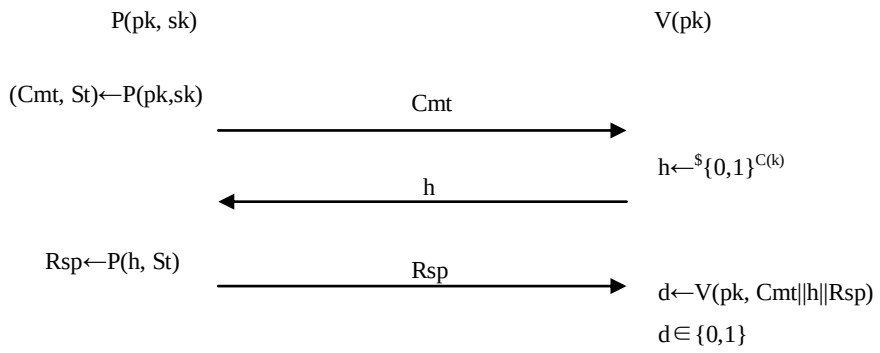


图 10-5 正则身份认证协议

设 s(k)是值为非负整数的 k 的函数，$H:\{0,1\}^{+}\rightarrow\{0,1\}^{C(k)}$ 是一个随机散列函数。正则身份认证协议 ID 的与 s(k)和 H 相关的 *Fiat-Shanir* 变换是一个数字签名方案 Ξ=(KG, Sig, Vf)，其中公钥/私钥偶的生成算法 KG 与 ID 的对应算法 KG 完全相同，签名算法和验证算法分别定义如下：

签名算法 $Sig^{H}(sk,M)$:

$R\leftarrow\{0,1\}^{s(k)}$; (Cmt, St)←P(pk,sk); h←H(R||Cmt||M); Rsp←P(h,St);

return(R||Cmt||Rsp);

验证算法 $Vf^{H}(pk,M,\sigma)$:

parse σ as R||Cmt||M; h←H(R||Cmt||M);

return(V(pk, Cmt||h||Rsp));

读者不难验证从协议 ID 的一致性关系能直接导出数字签名方案 Ξ 的一致性关系。

下面具体以上一节的身份认证协议为例，具体导出相应的数字签名方案。

例 10-1 $\mathcal{G}$ 是 q 阶循环群，q 是 k 位素数，g 是 $\mathcal{G}$ 的生成子；H:$\{0,1\}^+ \to Z_q$ 是随机散列函数，q、g、$\mathcal{G}$ 和 H 公开。*Schnorr* 协议(图 10-3)的 *Fiat-Shamir* 变换所导出的数字签名方案的组成算法如下：

公钥/私钥生成算法 KG(k, $\mathcal{G}$, g, q):

$x \leftarrow^{\$} Z_q$; $y \leftarrow^{\$} g^{-x}$; vk←y; sk←x; return(vk,sk);

签名算法 Sig^H(sk, M), 其中 sk=x:

$K \leftarrow^{\$} Z_q$; $r \leftarrow g^K$; h←H(M||r); s←(K+xh)mod q; return(r,h,s);

验证算法 Vf^H(vk, M, (r,h,s)), 其中 vk=y:

return(h=H(M,r)∧$r=g^s y^h$)

实际上，这就是在上一章例 8-1 中详细描述过的 *Schnorr* 数字签名方案。

例 10-2 p、q 是 k 位秘密素数，N=pq，H:$\{0,1\}^+ \to \{0,1\}^k$ 是随机散列函数，N、H 公开。*Feige-Fiat-Shamir* 协议(图 10-4)的 *Fiat-Shamir* 变换所导出的数字签名方案的组成算法如下：

公钥/私钥生成算法 KG(k, $\mathcal{G}$, g, q):

$x \leftarrow^{\$} \{1,2,\ldots,N-1\}$; $y \leftarrow^{\$} x^2 \bmod N$; vk←y; sk←x; return(vk,sk);

Sig^H(sk, M), 其中 sk=x:

$r_i \leftarrow^{\$} Z_N$, $v_i \leftarrow^{\$} r_i^2 \bmod N$, i=1,…,k;

h←H(M||v_1||…||v_k);

$z_i \leftarrow r_i x^{e_i} \bmod N$, e_i 是 h 的第 i 位, i=1,…,k;

$\sigma_1 \leftarrow v_1 \ldots v_k$;

$\sigma_2 \leftarrow z_1 \ldots z_k$;

return(σ_1,h,σ_2);

Vf^H(vk, M, (σ_1,h,σ_2)), 其中 vk=y:

parse σ_1 as $v_1 \ldots v_k$;

parse σ_2 as $z_1 \ldots z_k$;

parse h as $e_1 \ldots e_k$;

return(h=H(M||σ_1)$\quad \wedge_{i=1,\ldots,k} \quad z_i^2 = v_i y^{e_i} \bmod N$);

实际上，这个例子所导出的就是著名的 *Feige-Fiat-Shamir* 数字签名方案。

10.4 Kerberos V5 协议

前面所讨论的关于认证用户身份的技术都属于公钥密码类型，实际上同样存在对称密码类型的身份认证协议，其中 *Kerberos* 协议是目前比较完美的一种。本节对 *Kerberos* V5 进行详细的描述和分析，并给出了一个应用实例，最后列举它的一些不足之处。为了能够突出要点，本节略去了 *Kerberos* V5 [4]的一些细节，比如它对跨域 (Inter-realm)认证的支持方法和对代理认证的支持。

10.4.1 术 语

在希腊神话中，*Kerberos* 是指守护地域之门的三头狗。*Kerberos* 协议源于美国麻省理工学院 Athena 计划的一部分。总的来说，*Kerberos* 协议 是一种基于对称加密算法、并且需要可信任的第三方认证服务器的身份认证系统，使网络上通讯的实体互相验证彼此的身份，同时还能够提供对通讯数据的保密性和完整性保护。

这一节的目的是解释 *Kerberos* 协议特有的一些术语。因为目前对这些术语并没有形成普遍而一致的规范的汉语表达，因此为方便读者阅读原始文献，这里对大多数术语直接给出了其英文名称。

Realm：一个 *Kerberos* 协议的认证数据库所负责管辖的网络范围称作一个 Realm。这个认证数据库中存放有该网络范围内的所有 Principal(见下面)和它们的对称密钥，数据库的内容被 *Kerberos* 的两类认证服务器 AS 和 TGS 所使用(具体在后面还要进一步解释)。Realm 的名字通常用大写的字符串表示，并且在大多数 *Kerberos* 系统的配置中，Realm 的范围和该网络环境的 DNS 域是一致的，但这只应该看做一种为方便系统管理而采取的策略，并不意味着 Realm 和 DNS 域有任何概念上的必然联系。

Principal：在 *Kerberos* 中，Principal 指参与身份认证的主体，而且一般来说有两种，一种用来代表网络服务使用者的身份，另一种用来代表某一特定主机上的某种网络服务，也就是说 Principal 是用来表示客户端和服务端身份的实体。为了使用方便，用户所见到的 Principal 的名字用一个字符串来表示，而在网络实际传输中，Principal 的格式采用 ASN.1 标准(即 Abstract Syntax Notation 1)来准确定义。用户所见的表达 Principal 名字的字符串分为三个部分：

①Primary：第一部分。当代表客户方时，它是一个用户名；当代表服务方时，它是一

[4] Kerberos 协议的早期版本存在安全漏洞, 后来被发现和改进, 目前广泛应用的是 Kerberos V5 版本, 它也是因特网标准的基础。此外, Kerberos 是开放源代码的软件, 它的源代码和相关文档可以从 http://web.mit.edu/kerberos 下载。这里请读者注意: 根据美国法律, 用于出口的加密产品(无论软件还是硬件)其保密强度必须低于用于美国国内的同类产品(但对数字签名产品则无此限制, 读者能理解这是为什么吗?), 因此如果付诸实用, 这类开放源代码软件中的加密部分必须自己重新实现, 不应直接应用原始程序。

种网络服务的名字。

②Instance：第二部分，用以对 Primary 做进一步描述，例如 Primary 所在的主机名或 Primary 的类型等。这一部分可以被省略。它与第一部分之间用‘/’分隔开。

③Realm：第三部分，表示 Principal 所在的 Realm，与第二部分之间用‘@’分隔开，缺省值为本地 Realm 的名字。

例如，Principal 的名字“bernie/ssdut.dlut.edu.cn @DLUT.EDU.CN”表示 Realm “DLUT.EDU.CN”中主机 ssdut.dlut.edu.cn 上的用户 bernie，而 Principal 名字“host/ssdut.dlut.edu.cn @DLUT.EDU.CN”表示 Realm“DLUT.EDU.CN”中主机 ssdut.dlut.edu.cn 上的 rlogin 服务。

Credential：一个 Ticket(见下面的解释)和与之相联系的会话密钥合在一起统称为一个 Credential，它们是客户端在向服务器证明自己的身份时所必需提供的两样东西。在一个 Ticket 的生存期内客户端会将这两样东西以 Credential 为单位保存在一个缓存文件中。

Ticket：一个 Ticket 是一个用于安全地传递用户身份所需要的信息的集合，它不仅包含该用户的身份标识，而且包含其它一些相关的信息。一般来说，它主要包括客户方 Principal 的名字、目标服务器的 Principal 的名字、客户方 IP 地址、时间戳(分发该 Ticket 时的时间)、该 Ticket 的生存期以及会话密钥等内容。它的格式亦用 ASN.1 来准确定义。

Authenticator：在客户端向服务器进行认证时伴随 Ticket 一起发送的另外一个部分，它的作用是证明当前发送 Ticket 的用户确实是拥有该 Ticket 的用户，即防止重放攻击。它的主要内容是一个时间戳，即客户端发送该 Ticket 时的时间。

AS 服务器(Authentication Server)：为用户分发 TGT (Ticket Granting Ticket)的服务器。

TGT 服务器(Ticket Granting Ticket)：使用户能够向 TGS 服务器(Ticket Granting Server)证明自己身份的 Ticket。

TGS 服务器(Ticket Granting Server)：为用户分发到最终目标 Ticket 的服务器，用户使用这个 Ticket 向自己要求提供服务的那个服务器证明自己的身份。对 AS 服务器、TGT 服务器、TGS 服务器这三个概念在这里只能暂时给出简单的叙述，它们更确切的含义在下面一节还可以见到。在软件实现时，AS 服务器和 TGS 服务器实际上常由同一进程实现的，因为它们的实现机制并没有太大的差别，只是在加密所发送的 Ticket 时所使用的密钥不同，即 AS 服务器使用用户的密钥，而 TGS 服务器使用会话密钥。

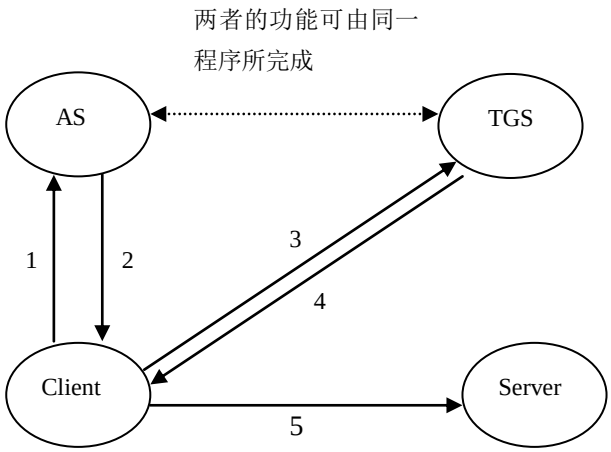
KDC (Key Distribution Center)：这个概念系历史遗留，比较模糊，通常将 AS 服务器和 TGS 服务器统称为 KDC，有时也把 AS 服务器单独称为 KDC。

10.4.2 协议的会话过程

Kerberos 协议的会话过程比较复杂，为下面叙述方便，首先约定一些记号：

c	客户端 Principal
s	服务器的 Principal
addr	客户端的 IP 地址
life	该 Ticket 的生存期
n	nonce，一个随机串
TGS	Ticket Granting Server
AS	Authentication Server
K_x	x 的密钥
$K_{x,y}$	x 和 y 的会话密钥
$T_{x,y}$	x 向 y 申请服务所用的 Ticket
$\{M\}_{K_x}$	用 x 的密钥加密明文 M 后的密文
A_x	x 发出的 authenticator

Kerberos V5 认证协议主要由 5 步构成(图 10-6)，下面详细解释。



1. Client → AS : c, tgs, n
2. AS → Client: $\{K_{c,tgs},n\}_{Kc},\{T_{c,tgs}\}_{Kc,tgs}$
3. Client → TGS: $\{A_c\}_{Kc,tgs},\{T_{c,tgs}\}_{K,tgs}$, s, n
4. TGS → Client: $\{K_{c,s},n\}_{Kc,tgs},\{T_{c,s}\}_{Ks}$
5. Client→Server: $\{A_c\}_{Kc,s},\{T_{c,s}\}_{Ks}$

图 10-6 Kerberos V5 认证协议

第一步：Client→AS: c, tgs, n 即 Client 向 AS 申请 TGT(即图中的 $T_{c,tgs}$)。

客户向 AS 发送的消息主要有：代表自己的 Principal 的字符串 c，代表所申请服务的 Principal 的 tgs 以及一个随机串 n，n 的作用在下面可以见到。

第二步：AS→Client: $\{K_{c,tgs},n\}_{Kc},\{\ T_{c,tgs}\ \}_{Ktgs}$，即 AS 发送 TGT($T_{c,tgs}$)和会话密钥 $K_{c,tgs}$ 给 Client。

AS 收到 Client 第一步发来的消息后，依据 Principal 的名字 tgs 知道 Client 拟申请 TGT。在验证自身数据库中存在 c 所表示的 Principal 后，AS 首先生成能够证明 Client 身份的 Ticket $T_{c,tgs}$ 和一个随机会话密钥 $K_{c,tgs}$。$T_{c,tgs}$ 的结构如下：

$$T_{c,tgs}=\{\ c\ ,\ tgs,\ addr,\ realm,\ timestamp,\ life,\ K_{c,tgs}\}$$

然后 AS 向 Client 发送下列消息：用 c 的密钥 K_c 加密的 $K_{c,tgs}$ 和 n 以及用 TGS 的密钥 $K_{,tgs}$ 加密的 $T_{c,tgs}$。将 Client 发送过来的 n 加密后再发回的作用是向 client 证明自己确实就是 AS，以防止第三方冒充 AS(因为 K_c 只有 Client 和 AS 两者共享，故$\{K_{c,tgs},n\}_{Kc}$ 只能由 AS 正确生成)。

第三步：Client→TGS: $\{A_c\}_{Kc,tgs},\{T_{c,tgs}\}_{Ktgs}$, s, n 即 Client 向 TGS 申请用以访问 Server 的 Ticket（图中的 $T_{c,s}$）。

Client 在收到第二步 AS 发回的消息后，首先用自己的密钥 K_c 解密$\{K_{c,tgs},n\}_{Kc}$ 得到 $K_{c,tgs}$ 和 n，然后将这个 n 与第一步发出的 n 相比较，如不同则说明有异常；如果相同则把 $K_{c,tgs}$ 和$\{T_{c,tgs}\}_{Ktgs}$ 保存在 Credential 缓存文件中。接下来，Client 生成 Authenticator A_c 并把它用 $K_{c,tgs}$ 加密。A_c 的结构包括 c , s , realm , timestamp 四个分量。

最后 Client 向 TGS 发送消息，消息的内容有：原封不动地照搬第二步收到的$\{T_{c,tgs}\}_{Ktgs}$、用 $K_{c,tgs}$ 加密的 A_c、代表要申请的服务的 Principal 的名字 s 和一个新的随机串 n（作用与第一步中的 n 相同）。

第四步：TGS→Client: $\{K_{c,s},n\}_{Kc,tgs},\{T_{c,s}\}_{Ks}$，即 TGS 发送$\{T_{c,s}\}_{Ks}$和会话密钥$K_{c,s}$给 Client。

TGS 收到 Client 第三步发来的消息后，首先用自己的密钥 K_{tgs}（只有 TGS 和 AS 事先共享 $K_{,tgs}$）解密$\{T_{c,tgs}\}_{Ktgs}$ 得到 $T_{c,tgs}$，进一步可以从 $T_{c,tgs}$ 中得到 $K_{c,tgs}$，然后便可以解密出 A_c。接下来，它将 A_c 中的 c 和 realm 与 $T_{c,tgs}$ 中的 c 和 realm 相比较，看它们是否一致。如果不一致，说明有错。如果一致，说明 A_c 是用 $K_{c,tgs}$ 加密的。如果 A_c 中的时间戳不是最近的（例如 5 分钟以内，具体数值由系统配置），说明有可能是重放攻击。如果 A_c 中的三元组(c,s,timestamp)是最近曾经接收过的，也表明有可能是重放攻击。如果不是最近接收过的，TGS 将把这个三元组记录下来，作为以后比较之用。除此之外，TGS 还会检查 $T_{c,tgs}$ 是否已超过它的生存期。如果这些检查都获得通过，TGS 便判定 Client 确实知道 $K_{c,tgs}$，又因为 $T_{c,tgs}$ 在客户端上只能用 K_c 解密得到，也即 Client 知道 K_c，那么 Client 的身份就是 c。

在确认 Client 的身份后，TGS 便产生随机的会话密钥 $K_{c,\ s}$ 和票据 $T_{c,\ s}$，然后从它的数据库中取出 s 的密钥 K_s，进行一番加密过程之后把$\{K_{c,\ s},\ n\}_{Kc,tgs}$ 和$\{T_{c,\ s}\}K_s$ 发送给 Client。

第五步：Client→Server: $\{A_c\}_{Kc,\ s},\{T_{c,\ s}\ \}_{Ks}$，即 Client 将 $T_{c,\ s}$ 提交给 Server。

Client 收到 TGS 第四步送到的消息后的处理与第三步中的描述基本相同，只不过将解密钥由 K_c 换成了 $K_{c,tgs}$，加密钥由 $K_{c,tgs}$ 换成了 $K_{c,\ s}$。最后，它把$\{A_c\}_{Kc,\ s}$ 和$\{T_{c,\ s}\ \}_{Ks}$ 发给 Server。

Server 认证 Client 身份的方法与 TGS 认证 Client 身份的方法相同，请参见第四步描述。如果 Client 反过来需要 Server 向它证明身份，那么 Server 便会将 A_c 中的 timestamp 用 $K_{c,s}$ 加密并返回给 Client，以证明自己就是 Server。在以后的通讯中，双方就可以直接用 $K_{c,s}$ 或用通过 $K_{c,s}$ 协商的新的会话子密钥来加密通讯数据，或用它们来计算通讯数据的验证码以保证数据完整性。

这里特别需要指出的是，*Kerberos* 协议要求用户经过 AS 和 TGS 两重认证的好处主要有两个：一是可以显著减少用户密钥的密文的暴露次数，即可以减少攻击者对有关用户密钥的密文的积累；二是使 *Kerberos* 认证过程具有一次性认证的优点，也就是说，只要用户得到了 TGT 并且该 TGT 没有过期，那么用户就可以使用该 TGT 通过 TGS 向任何一个服务器证实自己的身份，而不必重每次针对不同的服务器新输入口令。

为了更清楚的说明 *Kerberos* V5 认证协议，这里给出一个实际应用中的例子：假设某一局域网 LAN#A 的 DNS 域为 DLUT.EDU.CN；Realm 名字为 DLUT.EDU.CN；*Kerberos* 的用户数据库、AS 以及 TGS 服务器都在主机 kerberos.dlut.edu.cn 上。LAN#A 中主机 ssdut.dlut.edu.cn 上的用户 bernie 对应的 Principal 名字为 bernie/ssdut.dlut.edu.cn @DLUT.EDU.CN；LAN#A 中的另一台主机 server.dlut.edu.cn 上的 rlogin 服务器对应的 Principal 名字为 host/server.dlut.edu.cn @DLUT.EDU.CN。假设 Rlogin 的客户端程序 krlogin 和服务端程序 krlogind 都是 *Kerberos* 化的，即都支持 Kerberos 认证协议。假设 bernie 想通过 rlogin 远程登录到 LAN#A 中的另一台主机 server.dlut.edu.cn 上，这时 bernie 所要完成的步骤如下(以下所有程序名、命令和输出都是真实的情况)：

运行 kinit 程序。kinit 程序的作用是向 AS 申请 TGT，并将获得的 TGT 和会话密钥放在保存 Credential 的文件中。为此，bernie 在命令行上键入：

bernie ssdut$ kinit -p bernie/ssdut.dlut.edu.cn

“-p”后面跟的 Principal 名字表示为谁申请 TGT。kinit 程序在收到 AS 发回的消息后，就提示 bernie 输入口令：

bernie ssdut$ kinit -p bernie/ssdut.dlut.edu.cn

Password for bernie/ssdut.dlut.edu.cn @DLUT.EDU.CN:

在 bernie 正确输入口令后，kinit 程序将这个口令用约定的算法转化为 bernie 的密钥，并用这个密钥解密从 TGS 发回的消息，从而获得下一步使用的 Credential。到此，kinit 程序结束。注意 bernie 的口令并没有在网上传输，在网上传输的只是由 bernie 的密钥加密后生成的密文。

运行 rlogin 的客户端程序 krlogin，为此 bernie 在命令行上键入：

bernie ssdut$ krlogin server.dlut.edu.cn

krlogin 程序首先在保存 Credential 的文件中寻找未过期的 TGT 并把它交给 TGS，从而获得访问 server.dlut.edu.cn 上 rlogin 服务的 Ticket。接下来，它把这个 Ticket 和 Authenticator

一起发送给 server.dlut.edu.cn 的服务器程序 krlogind。Krlogind 在验证了 bernie 的身份后，搜索本地 bernie 主目录下的文件“.k5login”，在“.k5login”文件中放有允许用 rlogin 远程登录到 bernie 帐户下的 Principal 的名字列表。在找到了名字“bernie/ssdut.dlut.edu.cn @DLUT.EDU.CN”之后，krlogind 申请一个虚拟终端并生成一个命令解释程序 shell，这时在机器 ssdut.dlut.edu.cn 的终端上就会显示：

bernie ssdut$ krlogin server.dlut.edu.cn

Sun Microsystems Inc. SunOS 5.6 Generic August 1997

bernie server$

到此为止用户的登录—即向 LAN#A 证实自己的身份—就完全成功了。

10.4.3 协议的不足之处

目前 *Kerberos*V5 仍有一些不足之处。首先是不能很好的防止口令猜测攻击，也就是所谓字典攻击。为了使用方便，协议中用到的密钥都是由一个口令按某种算法转化而成。由于在网上传输的是$\{K_{C,tgs},n\}_{Kc}$，其中的 n 是可以窃听到的(见协议的第一步)，这样就给猜测 K_C 提供了可能。其实，大括号中的明文不止 $K_{C,tgs}$,和 n，根据具体实现还有其它一些具有可读性的明文，例如明文的消息首部等，这使得猜测 K_C 就变得更为容易。由于同样的原因，猜测会话密钥也是可能的。关于口令安全问题下一章还要专门阐述。

Kerberos V5 的设计者也意识到了这一问题，在 *Kerberos* V5 中作了以下几点努力：第一点改进是使 Ticket 具有生存期，在生存期内使用者不必再次申请 Ticket，这样就大量减 少了攻击者可以积累的被用户密钥加密的密文。但由于 Ticket 和会话密钥被保存起来以备再次使用，这样就使会话密钥被猜中的可能性增大。为了弥补这一不足，第二点改进是在每一次新的连接中，客户端和服务端可以商定一个新的子会话密钥，当连接关闭时，这个子会话密钥自动失效。第三，在请求 Ticket 的时候，可以设置 pre-authentication 选项。这个选项可以用来指明认证需要使用另外一种手持设备，这种手持设备能够在 K_C 的基础上产生不断变化的密钥。这样，就可以几乎完全阻止猜测口令攻击，但缺点是需要使用额外的设备。

第二类不足是用时间戳来阻止重放攻击代价偏高。为了使用时间戳来阻止重放攻击，服务器程序必须做以下这些工作：

比较实际消息的客户端 IP 地址和 Ticket 中所含有的客户端 IP 地址，验证两者是否一致；

保存所有有效的三元组(c,s,timestamp)；

每收到一个最近的(fresh)时间戳，都要把它和保存的所有三元组进行比较．如果发现有相同的，则说明是重放攻击．同时删除那些过期的三元组；

由于可能有很多的服务进程同时运行，对三元组的操作必须用某种机制来保持互斥。

由此可见，如果这种服务比较繁忙的话，它运行的速度将会有明显的降低。实际上，在

Kerberos V4 中就已经提出了这种解决方法，但是一直拖到 *Kerberos* V5 才给予实现，MIT 对此的解释正是这种解决方法的代价比较高。

第三类不足是时间同步问题，这也是一个薄弱环节。从以上的描述中可以看到，*Kerberos* V5 在很多地方都要依赖于时间，比如 Ticket 具有生存期、*Authenticator* 中含有时间戳等等。如果分布在各地的主机的时间很难保持一致，那么 Kerberos 认证系统就很难正常的运行。虽然 *Kerberos* 系统允许各个主机的时间有一定的偏差（最大偏差由系统管理员来定义），并在程序设计中把这个偏差考虑了进去，但是由于各台主机的时钟并不能很好的保持走时准确，这个偏差会越来越大。如果各主机的物理时钟实在差别比较大，则有必要采用某种时间同步协议，然而这不仅增加了 Kerberos 认证系统的复杂程度，而且带来了安全方面的新问题（即时钟同步协议带来的安全问题）。另外，如果某台主机的时间被更改，那么这台主机就无法使用 *Kerberos* 认证协议了；如果服务器的时间发生了错误，那么整个 *Kerberos* 认证系统将会瘫痪。

以上列出了 *Kerberos* V5 的一些不足，尽管 *Kerberos* V5 有以上这些不尽人意的地方，但仍然不失为一个在安全与代价的协调方面处理得比较好的协议（即“安全/代价比”合理），这也是它得到了广泛应用的原因之一。目前存在一些用公钥密码体制来加强 *Kerberos* 的方法，但由于公钥体制中的计算速度要比对称密钥体制慢很多，这些方法将增加 *Kerberos* 系统运行的代价。毫不奇怪，更高的安全要求必然要付出更高的代价。

10.5 组播环境中的数据源身份认证

至此所阐述的身份认证协议都属于 2-方协议。随着新一代网络环境(特别是无线/移动自组网络、P2P 网络)的成熟以及因特网上新一代应用的发展，基于组播通讯的应用也越来越多。在组播通讯中，一个发送方发送、多个接收方接收数据，这时的认证发生在一个发送方和多个接收方之间，特别是，数据接收方需要认证所接收到的数据确实来源于所期待的发送方并且在传输过程中没有经过任何篡改。要解决这一问题，一个简单的做法似乎是在发送方和每个接收方之间单独利用 2-方协议进行身份认证。尽管这样可以达到认证的目的，但既没有效率也不灵活。实际上，只要使认证时间稍做延迟就可以构造一个既高效又实用的认证方案，这就是著名的 *TESLA* 协议。

TESLA 协议的设计思想简单、巧妙而富有启发性，值得对此给出一个详细阐述。下面首先阐述原始的 *TESLA* 协议，然后进一步给出一个优化改进的方案。

10.5.1 *TESLA* 协议

相对于其它同类协议，*TESLA* 有很高的计算效率，特别是对接收方的存储量要求不高。*TESLA* 协议高效率的原因之一在于采用基于对称密钥的消息认证码方案对数据源做身份验证，而不是采用计算量较大的数字签名方案进行在线计算，这样既节省计算量又能有效避免对接收方的 DOS 攻击，同时采用一种巧妙而独特的密钥延迟公布机制。发送方将时间划分为等长度的周期区间 I_i，i=1,2,…N，每个区间分配一个密钥 K_i，在区间 I_i 上传输的所有数据 m 都以 K_i 为密钥的消息认证码 $MAC(K_i,m)$作为认证数据源身份的依据。各个密钥之间有数学关系

$$K_i=prf(K_{i+1})$$

其中 prf 是一个拟随机函数，从而使攻击者无法根据 $K_1,\ldots,K_i$ 有效推断出 K_{i+1}。最初的 K_N 由发送方随机生成，在发送数据前按以上公式(沿时间轴逆向)计算出各个 K_i，$\{K_i\}$构成一个密钥的链存储在发送方，钥链的最后一项 K_0 通过某种可信任的安全机制(例如数字签名)发送到组群的所有成员。在开始发送数据时，每个周期 I_i 上用于源身份验证的密钥 K_i 要在几个周期之后才被公布，这就是所谓延迟公布机制。设延迟为 d 个周期，接收方在周期 I_i 上接收到数据后并不是立刻处理，而是缓存 d 个周期直到在周期 I_{i+d} 上收到被公布的密钥 K_i 后，将 K_i 代入计算数据的消息码散列算法来验证缓存数据的来源是否合法。图 10-7 是 *TESLA* 协议的工作机理，延迟参数 d=1。

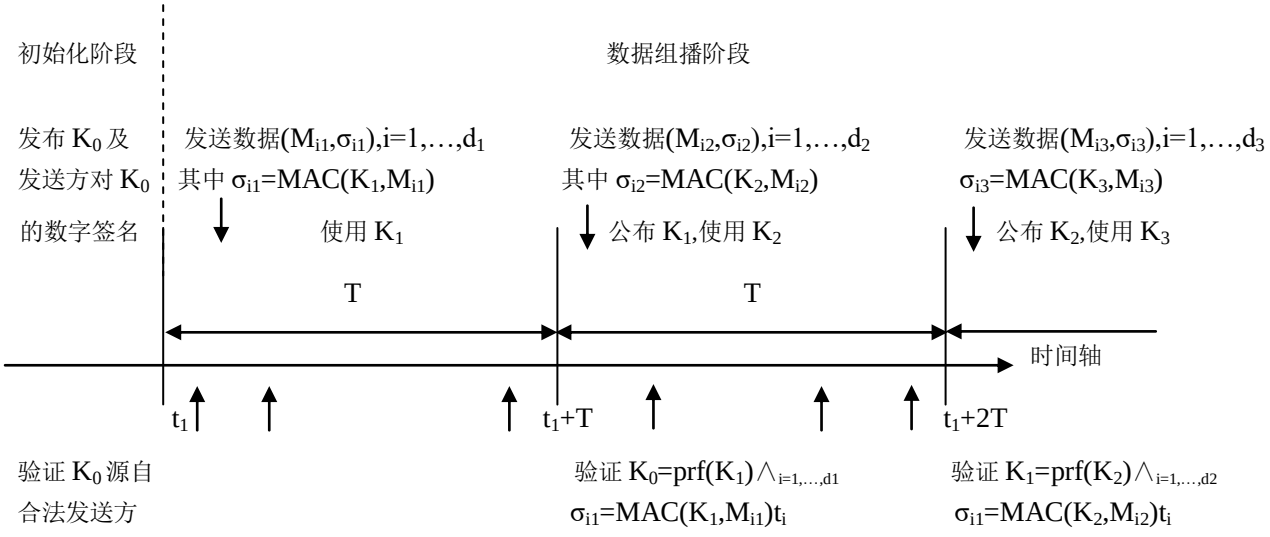


图 10-7 *TESLA* 协议

下面更具体地描述 *TESLA* 协议。在初始化阶段，发送方以其私钥签字并发布协议参数，这些参数包括起始时间 t_1、钥链长度 N、验证周期 T、密钥的公布延迟 d(以 T 为单位，为陈述简洁以下取 d=1)、时钟同步的最大误差Δ和起始钥 K_0。注意各接收方只需要在初始化阶段对这些参数验证一次即可。

在组播通讯阶段，协议的工作机理表示在图 10-7 中，其中发送方的动作表示在时间轴的上方，接收方的动作表示在时间轴的下方。

注意在 *TESLA* 协议中数据接收方需要存储的参数很少，本质上就是发送方的签字公钥和钥链的最后一项 K_0。由于拟随机函数 prf 的单向性质，即使各个接收方被完全入侵，入侵者也无法据此冒充数据发送方。

TESLA 协议需要接受方与发送方保持时钟弱同步，这里“弱”的涵义是指允许两者的本地时钟存在已知的同步误差上限。由于接收方必须确信在其接收到数据时该数据的源验证密钥尚未被公布，否则攻击者将可以观察到密钥从而伪造数据及其来源，因此对周期区间 I_i 内(接收方本地)时刻 t 到达的数据，必须满足 *TESLA* 安全准则 $t+\Delta<t_{i+d}$，其中Δ是时钟同步的最大误差，t_{i+d} 是区间 I_{i+d} 的起始时刻。只有同时满足 *TESLA* 安全准则和消息认码检验的数据才被确认为合法数据。

TESLA 是一个精心设计的协议，其密钥延迟公布机制本质上是一种秘密承诺机制。章末列举的原始论文对该协议的设计、优化、抗攻击能力及应用做了详细分析。

从理论上讲，TESLA 的初始参数一旦确定下来，只要钥链足够长，则一次初始化后系统就可以正常工作，这样即使初始化机制复杂一些，例如采用上面那样的公钥机制，似乎问题仍然可以合理解决。但问题在于一个很长的钥链在实际应用中往往并不可行：一方面，构造一个足够长的（例如与网络节点生命周期同数量级）的钥链需要发送方预先计算和存储很多密钥，密钥数量与周期 T 成反比、与钥链总的覆盖时间成正比，例如周期为 100 毫秒，覆盖时间 24 小时，每个密钥 8 字节，则要求存储量近 7MB。若验证周期进一步减小、或覆盖时间进一步增大，则存储量要求进一步增大，然而实际应用恰恰期望验证周期较短而覆盖时间较长，这是因为较长的验证周期会延长数据的验证延迟(达 d 个验证周期)，增加缓存要求，特别不利于实时应用，而短覆盖时间的缺点是不言而喻的。另一方面，即使一次性地计算出一个很长的钥链，在实际应用中也可能造成低效率，这是因为发送数据的时刻在实际应用中未必固定，甚至事先未知。考虑一个距离钥链起始时间 t_1 之后很长时间 $\tau=kT$ 才开始发送数据的情形，这时为验证数据源合法，接收节点需要计算 $K_0=prf^{k-d}(K_{k-d})$，当 k 比 d 大很多时这仍然需要很大的计算量，不适于计算能力不强的微型传感器网络。

综合以上分析，构造一种既适合于长覆盖时间、又适合于短验证周期的灵活的参数初始化机制，将成为改进 *TESLA* 协议的关键。

10.5.2* 改进的双钥链 *TESLA* 协议

我们增强 TESLA 协议的基本途径是用两层钥链代替原始协议中的单一钥链：一个是验证周期较长、用于发布源身份可验证的初始化参数的所谓粗链，其验证周期称为粗周期；一个是验证周期较短、用于对测量数据本身进行源验证的所谓细链，验证周期称为细周期。粗周期是细周期的整数倍。在每个粗周期内都发布一组参数，用以决定下一个粗周期上的细链初始化参数。两个链都以原始 TESLA 机制实现数据源验证，粗链的源验证针对初始化参数，而细链则针对测量数据本身。

由此，数据发送方可以较长的周期更新细链 *TESLA* 协议的工作参数，但不必计算和存储过长的钥链；同时，细链上的 *TESLA* 协议则可以选择尽可能短的验证周期，以缩短验证延迟，从而解决了上一节中的矛盾。下面具体地构造这一双链方案。

对粗钥链$\{K_i\}$，粗链长度是 N，周期是 T_1，起始时刻是 t_1。$K_i = prf_1(K_{i+1})$，$i=0,1,\ldots,N-1$，prf_1 是一个拟随机函数。与原始 *TESLA* 协议一样，末端钥 K_N 是发送方选择的随机数，K_i 是用于时间区间 $I_i=[t_i, t_{i+1}]=[t_1+(i-1)T_1, t_1+iT_1]$上的数据源验证钥。粗链的公布延迟取为 1，因为一般粗周期 T_1 比细周期 T_2 长许多，这是一个合理且实用的约定，否则反而降低双链方案的效率。另外，粗链的公布延迟为 1 也能最大程度减少传感器的接收缓存空间（但增强型协议本身很容易适于任何数值的粗链公布延迟）。因此，针对粗链的 TESLA 安全准则是 $\tau_i+\Delta<t_{i+1}$，τ_i 是在 I_i 期间接收到粗链消息的本地时刻，Δ是接收方/发送方时钟同步的最大误差。

在每个粗周期 I_i 上有一条细钥链$\{K_{ij}\}$，$K_{ij}=prf_2(K_{ij+1})$，$j=0,1,\ldots,m-1$，$m=T_1/T_2$，prf_2 是一个拟随机函数(与 prf_1 独立)。细链的公布延迟为 d，根据[6]中讨论的方法选择。对 $j\geq d$, K_{ij} 是细周期 $J_{ij}=[t_i+(j-d)T_2,\ t_i+(j-d+1)T_2]$上的 *TESLA* 数据源验证钥。与 K_N 相似，K_{im} 是由发送方选择的随机数。针对细链的 TESLA 安全准则是$[(\lambda_{ij}-t_1+\Delta)/T_2]+1<j+d+(i-1)m$，$\lambda_{ij}$ 是在 J_{ij} 期间接收到细链 *TESLA* 消息的本地时刻，Δ是接收方/发送方时钟同步的最大误差。

粗钥 K_i 在粗区间 I_{i+1} 上被公布，由下面详述的计算规则验证其合法性。粗周期 I_i 上的 TESLA 消息包含关于 I_{i+1} 上的细链的全部初始化参数，因此一旦粗周期 I_i 上的 TESLA 消息在 I_{i+1} 上被验证合法，传感器即可使用 I_{i+1} 上的 TESLA 细链来对测量数据进行源身份验证。

从理论上讲，不同粗周期 I_i 上的细链$\{K_{ij}\}$可以彼此独立，但这种做法的缺点是协议的容

错能力较原始 *TESLA* 协议差。考虑公布细钥 K_{im} 的细链 *TESLA* 消息被丢失的情形，由于 K_{im} 是 I_i 上的细链的最后一个密钥，这种丢失将导致 K_{im} 无法恢复。为保持 TESLA 协议的容错性，我们设计 $K_{im}=prf_3(K_{i+1})$，prf_3 是一个拟随机函数，与 prf_1 和 prf_2 独立。由此，K_{im} 总可以通过公布 K_{i+1} 的粗-*TESLA* 消息得到恢复(但代价是至少延迟一个粗周期 T_1，因为 K_{i+1} 最快要在 I_{i+2} 上被公布)。

综合以上阐述，现在给出完整的增强协议。

在系统初试化阶段，所有接收方与发送方的时钟同步(具体同步机制不在这里讨论)。发送方生成以下参数：粗钥链长度 N、周期 T_1 及起始时刻 t_1，粗钥链的末端钥 K_0，粗钥链$\{K_i\}$，其中 $K_i=prf_1(K_{i+1})$，i=0,1,…,N−1，prf_1 是一个拟随机函数；细钥链的周期 T_2(T_1 总是 T_2 的某个整数倍：$T_1=mT_2$)，细钥链中的公布延迟 d(以 T_2 为单位)，时钟同步的最大误差Δ。这些参数要求满足约束关系 $dT_2+\Delta<T_1$。

在这些参数中，N、t_1、T_1、T_2、d、Δ和 K_0 都可以事先固定（事先固定 K_0 是可行的，这是因为发送方总需要事先计算出一条完整的粗链$\{K_i\}$，因此 K_0 总是一个已知参数，即使$\{K_i\}$的时间长度 NT_1 不足以覆盖系统的生命周期，发送方也可以事先多计算几条完整的粗链$\{K_i\}$，或通过第 4 节所讨论的进一步改进方法来增加更高层的长周期粗链将这里的 K_0 归结为最高层的初始化参数 K_0，总之不难做到使系统在初始化时完全确定 K_0），因此可以假设系统中的所有接收方都已知这些参数。粗链的公布延迟取为 1。

在每个粗周期 $I_i=[t_1+(i-1)T_1, t_1+iT_1]$开始之前，发送方计算覆盖该周期的一条完整的细钥链$\{K_{ij}\}$，$K_{ij}=prf_2(K_{ij+1})$，j=0,1,…,m-1，$prf_2$ 是一个拟随机函数，对 j≥d, K_{ij} 是细周期 $J_{ij}=[t_1+(i-1)T_1+(j-d)T_2, t_1+(i-1)T_1+(j-d+1)T_2]$上的 *TESLA* 数据源验证钥。这一细链的参数 K_{i0} 在 I_i 开始之前，即在粗周期 I_{i-1} 期间向系统以组播的方式公布，组播消息的形式为：

$$\Omega_{i-1} = K_{i-2}||i-1||K_{i,0}||MAC(K_{i-1}^*, i-1||K_{i,0})$$

(Ω_0 实际上在系统初始化期间发送，这时 K_{-1} 约定为空串)我们采用优化形式：

$$\Omega_{i-1} = K_{i-2}||i-1||K_{i,0}||H(K_{i+1,0})||MAC(K_{i-1}^*, i-1||K_{i,0}||H(K_{i+1,0}))$$

其中||表示字串的联结，i−1 表示当前粗周期的序号，K_{i-2} 是前一个粗周期 I_{i-2} 上的 *TESLA* 粗钥，用以验证消息 Ω_{i-2} 的数据源是否合法；K_{i0} 是要发布的细链参数；H 是一个单向散列函数，在 Ω_{i-1} 中包含 $H(K_{i+1,0})$这一项是为了提高验证效率(详见下面的步骤 4，注意这一优化实际上需要发送方在 I_i 开始前计算出覆盖 I_i 和 I_{i+1} 的两条完整的细钥链$\{K_{ij}\}$和$\{K_{i+1j}\}$，在实际应用中这一般是可行的)；$MAC(K_{i-1}^*, i-1||K_{i,0}||H(K_{i+1,0}))$是对 $i-1||K_{i,0}||H(K_{i+1,0})$的消息验证码[5]，用以防止 $i-1||K_{i,0}||H(K_{i+1,0})$被篡改，密钥 $K_{i-1}^*=f(K_{i-1})$，f 也是一个拟随机函数。

记 $t_i=t_1+(i-1)T_1$，接收节点接收到消息 Ω_i (= $K_{i-1}||i||K_{i+1,0}||H(K_{i+2,0})||MAC(K_i*, i||K_{i+1,0}||H(K_{i+2,0}))$) 时做以下处理：

1) 设接收到消息的本地时刻为 τ_i，首先验证 *TESLA* 安全准则 $\tau_i+\Delta<t_{i+1}$，若 $\tau_i+\Delta\geq t_{i+1}$ 则废弃消息 Ω_i；

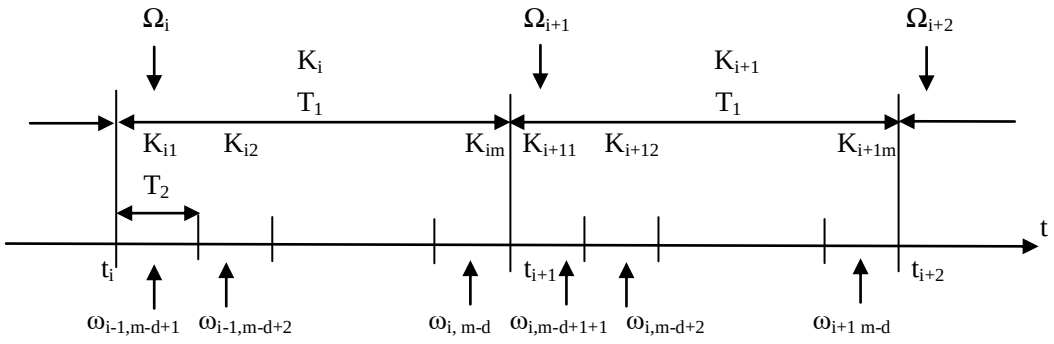
2) 设 $K_j(0\leq j\leq i-2)$ 是该节点在此之前最近的某个粗周期上的合法粗钥[5]，验证 $K_j=prf_1^{i-j-1}(K_{i-1})$，若成立则保存 Ω_i 并以 K_{i-1} 替换 K_j，否则废弃 Ω_i；

3）设 $\Omega_{i-1}(=K_{i-2}||i-1||K_{i,0}||H(K_{i+1,0})||HMAC(K_{i-1}*, i-1||K_{i,0}||H(K_{i+1,0})))$是该节点在前一个粗周期上接收到的粗链消息，验证 $MAC(f(K_{i-1}), i-1||K_{i,0}||H(K_{i+1,0}))=\Omega_{i-1}$ 中的 MAC 项的值，这里等式左面的 K_{i-1} 用 Ω_i 中的项，$i-1||K_{i,0}||H(K_{i+1,0})$则用 Ω_{i-1} 中的对应项，若等式成立则 Ω_{i-1} 的数据源身份合法，否则废弃 Ω_i；

4) 对合法的 Ω_{i-1}，进一步验证 $H(\Omega_i.K_{i+1,0})= \Omega_{i-1}.H(K_{i+1,0})$，这里 $\Omega_i.K_{i+1,0}$ 表示 Ω_i 中的项 $K_{i+1,0}$，$\Omega_{i-1}.H(K_{i+1,0})$表示 Ω_{i-1} 中的项 $H(K_{i+1,0})$，若等式不成立则废弃 Ω_i；

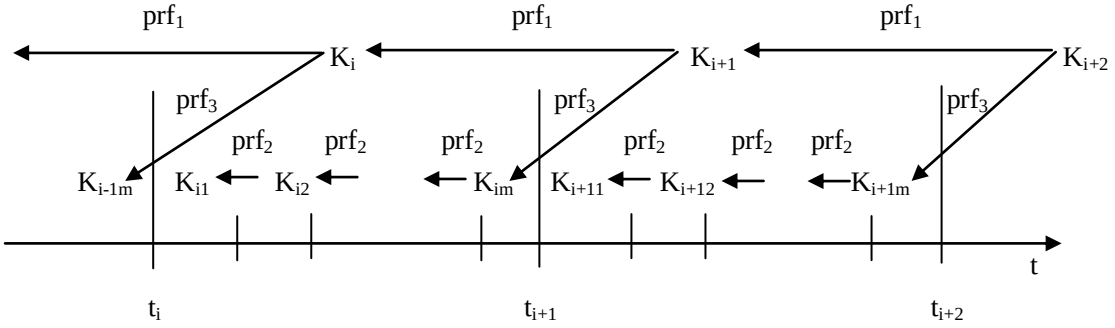
5）在 I_i 中以原始的 *TESLA* 协议处理细链消息。

图 10-8 是对以上构造的直观表达，其中 $\omega_{i\ j}$ 表示细链 *TESLA* 消息，其构造可以完全沿用上一小节的描述。



a) 双链周期及 *TESLA* 消息

[5] K_j 总存在，因为至少 K_0 总可以成为这样一个粗钥。



b) 密钥之间的关系

图 10-8 改进的双钥链 *TESLA* 协议

10.6 小结与进一步学习的指南

在实际应用中，身份认证协议既是一类有独立应用价值的安全机制，同时也经常作为许多更复杂的安全协议的一个组成部分，例如下面章节将要阐述的密钥交换协议。本章阐述了身份认证协议的安全性概念、典型的公钥密码类型的身份认证协议和基于对称密码方案的身份认证协议。和以往一样，这里的讨论仅限于直观阐述，对身份认证协议的进一步参考资料可以阅读第八章所介绍的 W.Mao 的教科书《现代密码学理论与实践》和 Ooschort、Menezes 及 Vanstone 的著作《应用密码学手册》，后者特别从零知识证明的统一观点给予阐述。所谓零知识证明是这样一类协议，它使验证者 V 相信证实方 P 知道关于一个公开的难题 *L* 的解 *w* 但却不向 V 暴露这个解 *w* 本身。关于身份认证协议更精确而系统的理论阐述可以参考下面著作的第 2 章和第 7 章：

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

目前 *Kerberos* V5 协议在各类分布式系统中被广泛实现，包括 Windows 2000 及其之后的平台在其目录服务 ADS 中所应用的身份认证机制就是经过适当改造的 *Kerberos* V5 协议。这一协议业已成为因特网标准，标准文档是 RFC#1510，这是全面理解这一协议的必读资料，其中不仅详细描述的协议消息，还详细定义的各个相关的数据类型与结构。

Kerberos 协议用以使用户登录到一个分布式系统，一次登录即对系统中的所有服务器有效。与此不同，单纯的主机登录协议仅仅使用户可靠地远程登录到特定的服务器本身，并不能够向其它主机保证身份的可信性，在这类协议中，*Telnet* 是被广为熟知的一个，它的本质功能理所当然的是可靠的身份认证，但 *Telnet* 在这方面的不足也同样广为人知，早期的 *Telnet* 甚至不对用户口令加密！为此从 90 年代末开始已经开发了一个重要的替代协议：*SSH*。

目前 *SSH* 已经成为因特网标准，它比 *Telnet* 复杂得多但也安全得多，而且还有成熟的开放源代码系统 *OpenSSH* 可以详细参考，感兴趣的读者可以访问 www.openssh.org。

本章讨论的都是 2-方身份鉴别协议，实际上还存在多方身份鉴别协议，这时需要一个组群中的成员向一个验证方证明自己确实是该组群中的合法成员，但同时不暴露自己究竟是哪一个特定的成员(匿名)；另一方面，成员之间即使合谋也不能伪装成另一个无辜成员的身份欺骗验证方。这类协议有很多应用，例如用于保护成员行踪隐私的安全访问控制系统。这类协议也有 *Fiat-Shamir* 变换，结果就是第 8.5 节提到过的组群签字方案，可以用于实现数字货币。这类协议和方案比 2-方协议和普通数字签名方案复杂得多，目前还没有教科书程度的详细介绍，感兴趣的读者暂时只能阅读研究论文。

10.5 节阐述的 *TESLA* 协议属于多方身份认证协议的特例，发表于 A.Perrig, R.Canetti, D.Song *Efficient and Secure Source Authentication for Multicast.* Proc. Network and Distributed System Security Symposium, 2001。*TESLA* 协议目前已经成为因特网标准。

习 题

10-1 图 10-3 中的 K 如果不是每次会话独立生成的随机数，而是一个重复使用的常数(从而 r 也是常数)，考虑两次协议会话，其中 h 不同且分别为 h_1 和 h_2，对应的消息 s 分别为 s_1 和 s_2，则可以从这两次协议会话的消息及公开的参数计算出 P 的私钥 x，为什么？

10-2 图 10-4 中的各个 r_i 如果不是每次会话独立生成的随机数，而是一组重复使用的常数(从而 v_i 也是常数)，考虑两次协议会话，其中 h 不同且分别为 h_1 和 h_2，对应的消息 z_i 分别为 z_{i1} 和 z_{i2}，则可以从这两次协议会话的消息及公开的参数计算出 P 的私钥 x，为什么？

10-3 这一习题是关于著名的 *Guillio-Quisquater* 数字签名方案和相应的身份认证协议(1988)。

(1) *Guillio-Quisquater* 身份认证协议：设协议双方分别为 P 和 Q，P 有公开的身份标识 *J*(一个整数)和公开的指数 *v*。另一个公开的正整数 N 是两个秘密素数 *p* 和 *q* 的乘积，N 用做模数。P 的私钥是一个整数 *B*，*B*、*J*、*v* 和 N 之间满足关系 $JB^V=1 \bmod N$。

为了证明自己的身份确实是 *J*，P 需要使 Q 相信她确实知道身份标识 *J* 和 *v* 对应的那个 *B*(私钥)。协议的会话过程如下：

P 生成随机整数 *r*，*r* 在 1 到 N−1 之间；P 计算出 $T \leftarrow r^V \bmod N$ 并将 *T* 发送到 Q；

Q 生成随机整数 *d* 并发送至 P，*d* 在 1 到 N-1 之间；

P 计算出 $D \leftarrow rB^d \bmod N$ 并发送至 Q；

Q 验证 $T=D^V J^d \bmod N$ 是否成立，若成立则接受对方的身份确实是 P。

请证明以上验证方程对诚实的 P 总能成立。

(2) *Guillio-Quisquater* 数字签名方案：签字方 P 有公开的身份标识 J、指数 v 和正整数 N；P 的签字私钥是 B 且满足关系 $JB^V=1 \bmod N$；H 是一个单向散列函数，例如 MD5 和 SHA。

对拟被签字的消息 M，签字算法 Sig(B, M)如下：

生成随机整数 r，r 在 1 到 N-1 之间；

$T \leftarrow r^V \bmod N$；$d \leftarrow H(M\|T)$；$D \leftarrow rB^d \bmod N$；

输出 (M, (d, D, J))；

请自行给出验证算法 Vf((N, v), M, (d, D, J))。

(3)验证以上身份认证协议的 *Fiat-Shamir* 变换恰是以上数字签名方案.

(2)的提示：计算出 $T \leftarrow D^V J^d \bmod N$ 然后验证 $d=H(M\|T)$是否成立。

10-4 这一习题处理广义 *Feige-Fiat-Shamir* 协议，其安全性基于因子分解问题的难解性。记 K_{factor} 为多项式复杂度的模数生成算法，即 $K_{factor}(k)$生成两个不同的 k 位奇素数 p、q 和 N=pq，任何 P.P.T 算法仅从 N 和 k 计算出其素因子的概率都不大于 k 的某个速降函数，例如不大于 2^{-k}。这里特别要求所生成的 p 和 q 是所谓 *Blum-Williams* 素数，即满足 $p=q=3 \bmod 4$。对这类素数，总存在整数 a 使 $a^2=-1 \bmod p$(或 q，这是因为 *Legend* 符号$(-1/p)=(-1)^{(p-1)/2}=-1$，参考 8.1.5 节)且任何二次剩余方程 $x^2=a \bmod p$(或模 q)如果有解则总可以被有效计算出来。

记 Z^*_N 是 1 到 N 之间与 N 互素的整数的集合、$QR_N=\{x^2 \bmod N: x \in Z^*_N\}$、$Z^+_N=\{x \in Z^*_N$: *Jaccobi* 符号(x/N)=1}。广义 *Feige-Fiat-Shamir* 协议的钥生成算法 KG 如下，协议过程如下图所示，其中 t 和 m 是 k 函数，$\lim_{k\to\infty}(\log_2 k)/m(k)t(k)=0$；注意若取 $t(k)=k$、$m(k)=1$ 就得到原始的 *Feige-Fiat-Shamir* 协议。

KG(k):

(N, p,q)←K_{factor}(k);

for i=1,…,t(k) do{ x(i)←$^{\$}Z^*_N$; X(i)←$^{\$} \pm x(i)^{-2^{m(k)}} \bmod N$; }

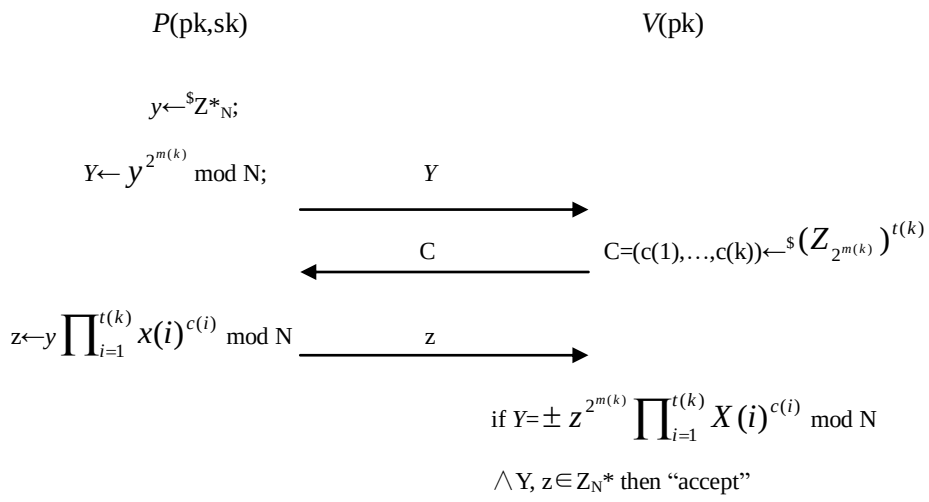
pk←(N,X(1),…,X(t(k));

sk←(N,x(1),…,x(t(k));

return(pk,sk);

(1) 证明：对诚实的 P，验证方程在 V 上总为真。

(2) 应用 *Fiat-Shamir* 变换从这个协议导出对应的数字签名方案。



10-5(*Sakai-Ohgishi-Kasahara*、*Hess* 和 *Cha-Cheon* 协议) 这三个协议都基于椭圆曲线上的双线性配偶计算及相应的难解性假设(关于双线性配偶的概念参考 8.4 节例 8-9 中的说明)，并且有相同的钥生成算法 KG(k)，其中 q 是 *k* 位素数，G_1、G_2 是 *q* 阶循环群，P 是 G_1 的生成子，*e*: $G_1 \times G_1 \to G_2$ 是群偶上的非退化双线性映射：

(q,P,G_1,G_2, *e*)←$K_{pairing}$(k); /* $K_{pairing}$ 是非退化双线性群偶生成算法*/

s←$$Z_q$; S←sP; U←$$G_1$; V←sU;

pk←((q,P,S,G_1,G_2, *e*),U);

sk←V;

return(pk,sk);

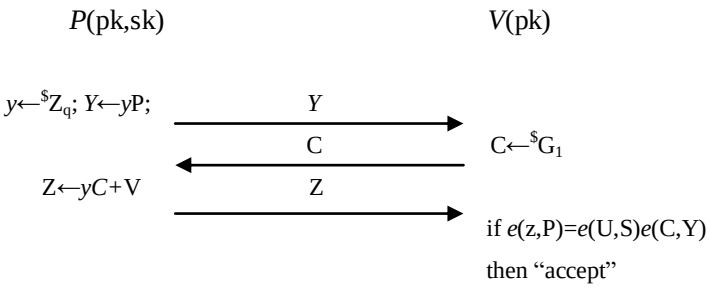
三个协议过程如下图所示。

(1) 对每个协议分别证明：对诚实的 P，验证方程在 V 上总为真。这里用到关于双线性映射 *e* 的性质 $e(aP,bQ)=e(P,Q)^{ab}$(*a* 和 *b* 是任何整数)、$e(P_1+P_2,Q)=e(P_1,Q)e(P_2,Q)$和 $e(P,Q)=e(Q,P)$。

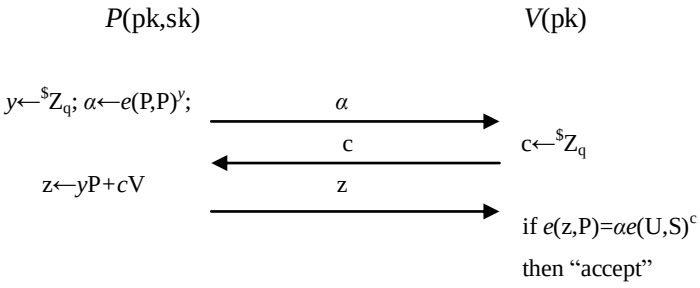
(2) 应用 *Fiat-Shamir* 变换从每个协议导出对应的数字签名方案。

(3) 已经证明(在特定的难解性假设之下)三个协议都抗被动攻击，但 *Sakai-Ohgishi-Kasahara* 协议不具备抗主动攻击能力。验证：如果接受到第一条消息 *Y*=*y*P 时攻击者 A 生成 c←$$Z_q$ 并向 P 响应 C←cP，根据协议，这时证明方 P 的响应 Z 将是什么？由此攻击者能从 Z 计算出证明方的私钥 V，为什么？

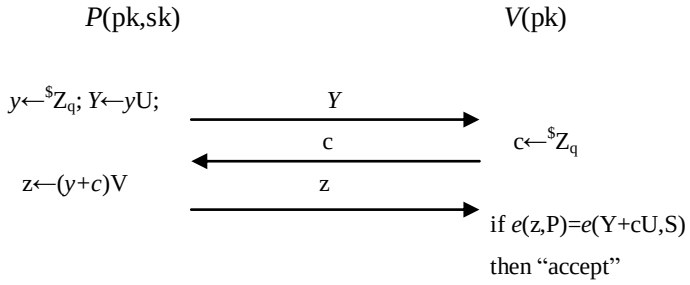
提示：Z=*y*C+V=*y*(cP)+V=c*Y*+V，由此攻击者能从 Z 计算出证明方的私钥 V←Z−c*Y*。



a) *Sakai-Ohgishi-Kasahara* 协议



b) *Hess* 协议



c) *Cha-Chaeon* 协议