互联网是如何工作的

一. 正常访问网站时发生了什么?

访问一个网站就像网购，你给这家店寄信，告诉他们要买 A 物品，商家收到你的信之后把货物发给你。每个网站都有一个域名，可以看成这家店铺的名字。但是只知道店铺的名字是不够的，还需要知道这家店铺的地址。

1. 通过域名查询 IP 地址

所以当在浏览器输入域名之后，首先系统会根据输入的域名，将其在本机中的 Hosts 文件里查找网站服务器的 IP 地址，这个 Hosts 文件就像家里的通讯录。

如果自己的通讯录里面没有这个地址，电脑就会把这个域名发送给宽带运营商的 DNS 服务器，在这个服务器里面进行查找，DNS 服务器就像查询台，里面有各种域名的地址。如果能查到的话，DNS 服务器就会把 IP 地址发回来。

如果还是查不到，DNS 服务器就会去问根域名服务器，根服务器记录了哪些是负责查.com 的，哪些是负责查.cn 的。根服务器根据域名的根，去问相应的服务器，问到了之后，把 IP 地址发给电脑。

2. 访问网站

电脑将出发地（源 IP 地址），目的地（目标 IP 地址）等信息写在信封上。然后寄给网站服务器。网站服务器收到之后，把数据流再寄回来。这样一来我们要的内容就显示在电脑上了。

3. 内容分发网络

离发货地越远，收货时间就越长，还可能会丢失数据。于是网站除了主服务器以外，又在很多地方建立了服务器，这些服务器与主服务器之间定期同步数据。这也叫 CDN，即内容分发网络。

二. GFW 做了什么?

防火长城（英文名称 Great Firewall of China，简写为 Great Firewall，缩写 GFW），也称中国防火墙或中国国家防火墙，指中华人民共和国政府在其管辖因特网内部建立的多套网络审查系统的总称，包括相关行政审查系统。首要设计者为北京邮电大学原校长方滨兴，被称为“国家防火墙之父”。

GFW 中的主要技术主要包括 DNS 劫持、主干路由器的关键字过滤阻断、HTTPS 证书过滤、IP 地址封锁等。

1. DNS 劫持

通过攻击域名解析服务器（DNS），或伪造域名解析服务器（DNS）的方法，如果搜索域名里含有关键词，则将错误的 IP 地址发送回电脑，或者不返回 IP 地址，导致无法访问，这就叫 DNS 劫持。但是，我们可以通过把将网站的 IP 地址写在 hosts 文件中，也就是自己的通讯录上，或者用其他没有被干扰的 DNS 服务器，绕开 DNS 劫持。

2. 主干路由器的关键字过滤阻断

在数据传输过程中，早期是没有加密的数据。而数据的传输需要经过一些中间结点，如路由器、服务器等。在主干路由器中 GFW 可以轻易的筛选数据内容。一旦检测到关键词，GFW 就会立马把数据拦下。但我们也可以通过数据加密，来阻止被过滤，例如现在的 https 而不是 http。

3. HTTPS 证书过滤

已经存在着很多的加密代理，比如 HTTPS 代理或者 Socks5 over TLS，但是这些加密代理协议都有一些问题，那就是 TLS 协议在握手阶段并不是加密的，服务器在传回自己的证书时是明文的，这就很容易被 GFW 探测到。往往只会打断与特定的 IP 地址之间的 TLS（HTTPS，443 端口）握手。

4. IP 地址封锁

加密之后虽然信件内容不可见，但是信封上的 IP 地址还是暴露在外面的。GFW 会根据需要屏蔽相应的 IP 地址，并把这封信全部丢掉。即通过使发往特定 IP 地址上特定端口的数据包全部被丢弃而达到封锁目的。

我们可以找到一个海外代理(proxy)，将然后通过代理就可以浏览自己平时看不到的资讯了。但网络封锁部门也就开始把人们常用的海外代理加入了 IP 封锁列表。

三. 如何绕过 GFW

1. 代理服务器（proxy）

电脑发送访问请求给一台海外 proxy，让其帮忙访问目标服务器，proxy 在海外解析域名并访问目标服务器，目标服务器把数据传给 proxy，再传回给电脑。但是如果在 proxy 传输给电脑的过程中，被 GFW 检测到了违规，则可能将这台 proxy 的 IP 地址加入封锁列表。后来出现了用 TLS 协议加密的 proxy 可以保护电脑和 proxy 之间传输的信息基本不被 GFW 检测到。

2. 虚拟专用网络（VPN）

VPN 称为虚拟专用网络（Virtual Private Network），其实质上就是利用加密技术在公网上封装出一个数据通讯隧道，利用公用网络架设专用网络。

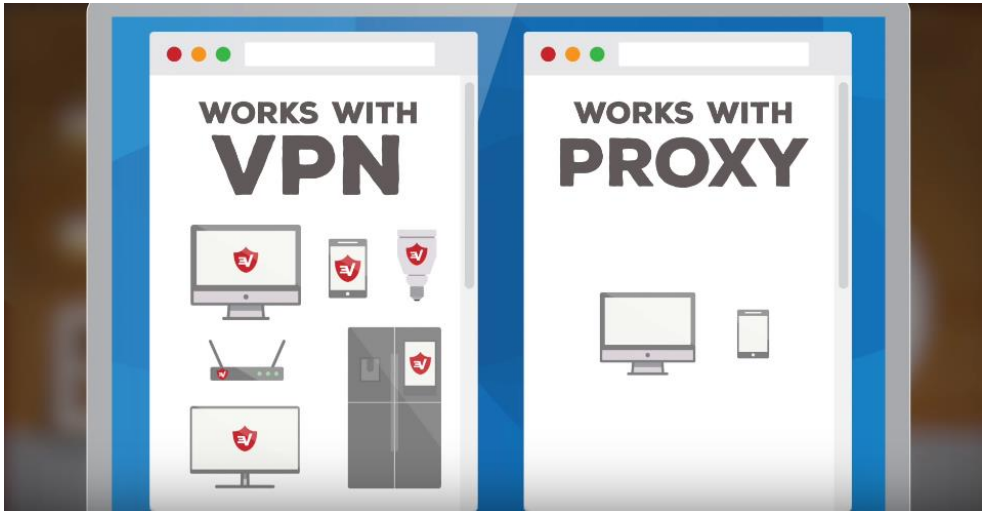
打开 VPN 后，VPN 会在电脑上虚拟出一个 IP 地址，并且对电脑和 VPN 服务器的通讯进行加密。我们向 VPN 服务器传输数据包时，自己的地址会用这个虚拟的 IP 地址，收件人的地址是 VPN 的服务器地址。VPN 服务器收到数据包之后进行解密，解析域名并将数据包传给网站服务器。网站服务器将数据包传给 VPN 服务器，VPN 服务器对其加密，再传回给电脑。

3. proxy 和 VPN 的区别

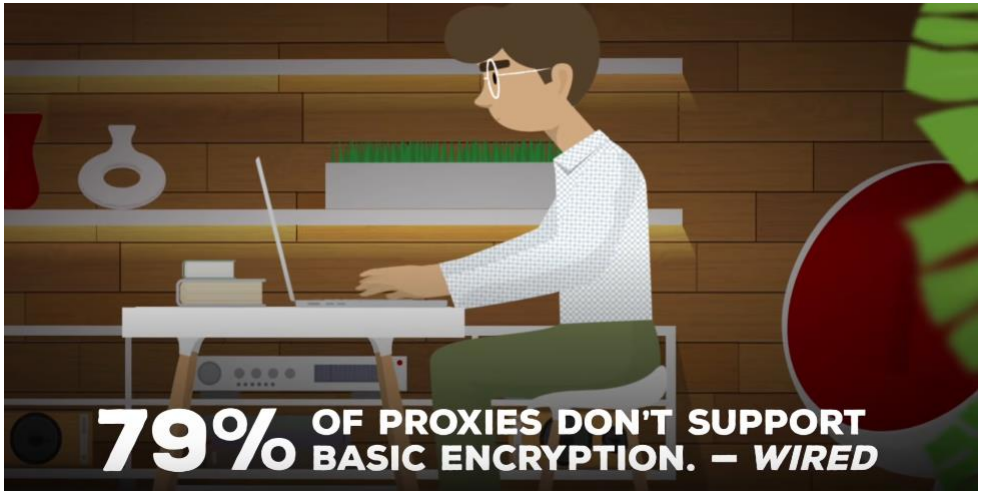
VPN 和代理很像，它们都电脑从另一个位置连接到目标服务器，目标服务器无法识别到我们的 IP 地址，而且可以绕开 GFW。 然而，它们本质上并不同。

① 代理服务器的程序级的，连接是在逐个应用程序的基础上配置的，而不是在全计算机范围内配置的。而 VPN 是在操作系统级设置的，VPN 会将每一个在计算机上运行的应用程序的流量加密链接，从 web 浏览器、在线游戏到在后台运行的 Windows 更新，VPN 将其所配置的整个设备连接到网络。

② VPN 可以在很多设备上使用，比如手机、电脑以及所有可以联网的设备上同时使用 VPN 进行网络连接。但是 proxy 只能在其配置的设备上工作。



③ 多数不会加密你的计算机和代理服务器之间的通信，或者取决于应用程序的类型进行不同的加密方式，不适合需要高安全性的场景。而 VPN 数据传输的整个过程都是通过计算机和远程网络之间的高度加密隧道进行的，所以需要下载客户端。这使得 VPN 连接适用于任何涉及隐私或安全的高风险网络使用。



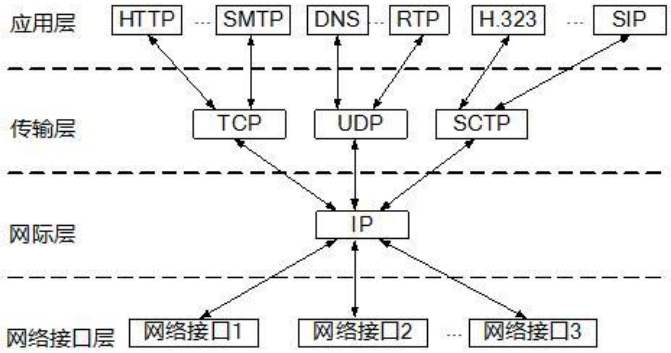
2020.02.12

如何上网（一） ——历史上主要的技术

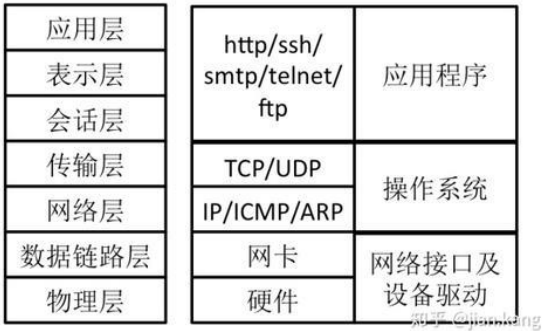
现在常用的软件主要由以下四种：SS、SSR、V2ray、Trojan。每一种软件所用的技术也有所不同。下面先说说历史上主要的两种技术：proxy 和 VPN。

1. proxy 和 VPN

在 OSI 模型上，proxy 根据其工作的层次可分为应用层代理（Nginx）、传输层代理（LVS、Haproxy）和 SOCKS 代理，而 VPN 作用在网际层。



TCP/IP 四层模型



OSI 七层协议模型

2. proxy

应用层代理是工作在 TCP/IP 参考模型的应用层之上，它支持对应用层协议（如 HTTPS、FTP）的代理，主要用于 web 文件的浏览和下载。利用 HTTPS 代理的方式可以为单个网站（域名）解决问题，但是我们日常上网肯定不可能把自己常上的国外网站一个个都配置上反向代理，配置繁琐而且不够实用。且 HTTP 属于 TCP 协议，不支持 UDP 报文的转发。

从上面可以看出应用层代理提供的控制最多，但是不灵活，必须要有相应的协议支持。如果协议不支持代理，那就只能在应用层以下代理，也即传输层代理。

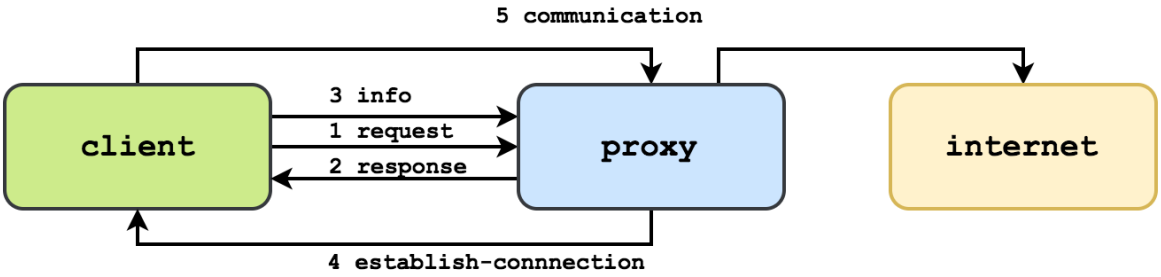
传输层代理直接与 TCP 层交互，更加灵活。要求代理服务器具有部分真正服务器的功能：监听特定 TCP 或 UDP 端口，接收客户端的请求同时向客户端发出相应的响应。现在有时使用 Haproxy 实现对采用 Socks5 代理的 Shadowsocks 进行国内中转加速。

另一种代理需要改变客户端的 IP 栈，即 SOCKS 代理。它是可用的最强大、最灵活的代理标准协议。SOCK V4 允许代理服务器内部的客户端完全地连接到外部的服务器，SOCK V5 增加了对客户端的授权和认证，因此它是一种安全性较高的代理。SOCKS V4 只支持 TCP 连接，而 SOCKS V5 在其基础上增加了安全认证以及对 UDP 协议的支持。

3. SOCKS 协议

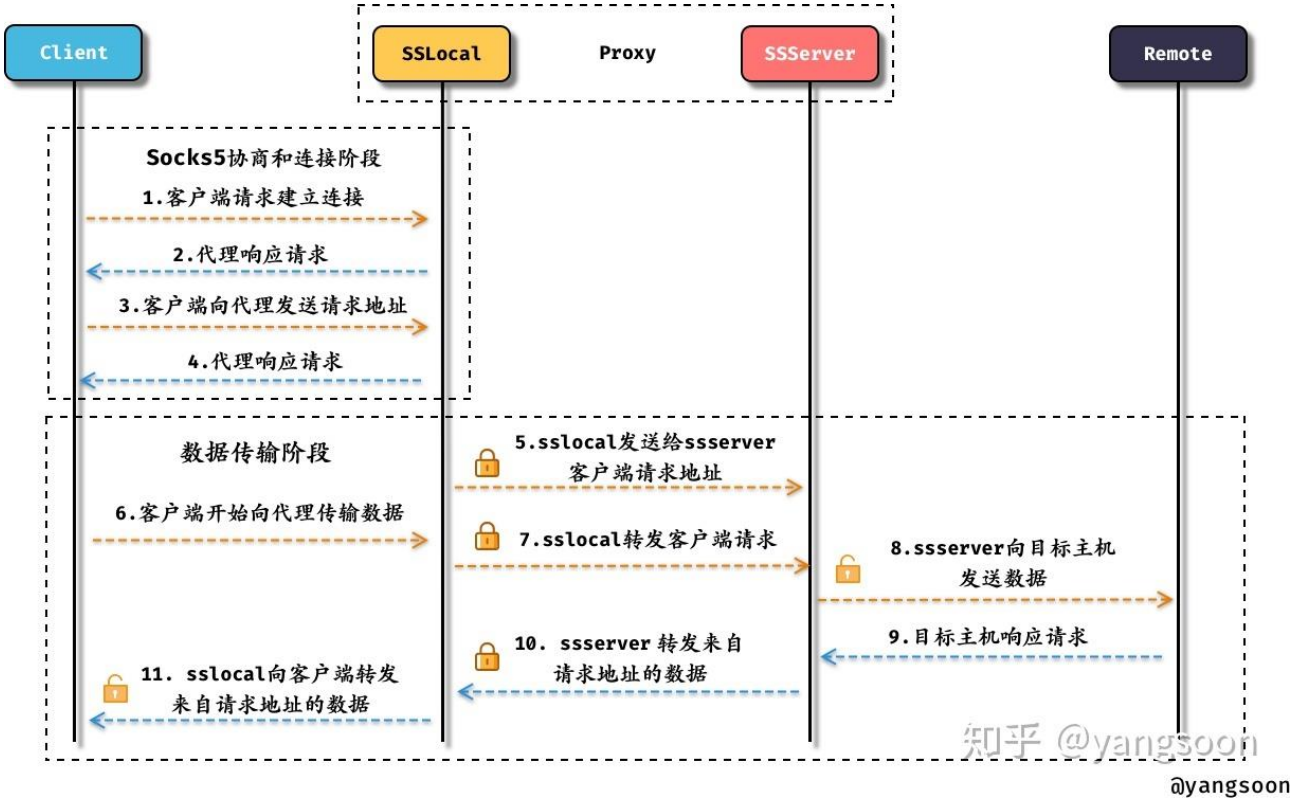
该协议设计之初是为了让有权限的用户可以穿过防火墙的限制，使得高权限用户可以访问外部资源。

SOCKS 协议是一种网络代理协议，它工作在 OSI 模型中的第五层——会话层，可以对传输的数据进行详细的访问控制。SOCKS 协议像一个代理一样，对客户端到服务器之间的数据联系提供安全上的服务。由于 SOCKS 作用在会话层上，因此它是一个提供会话层到会话层间安全服务的方案，不受高层应用程序变更的影响，也就不必关心是何种应用协议，telnet，http，ftp 都不在话下。而且由于此协议完全不关心应用层的协议，所以应用层通信可以独立加密，保护自己通信的内容不被 proxy 所看到。



socks5 proxy工作图

socks 5 因为简单易用的特性被用来实现代理功能，其实 socks5 并不提供加密功能，只被用来建立连接，所以我们用的 SS/SSR 是 socks5 加上使用一系列加密算法加密数据，以及使用代理服务器做转发，来实现一些网站的访问。下图为进行网站访问的整体流程：



4. VPN

VPN 通过编写一套网卡驱动并注册到操作系统，在操作系统的接口直接虚拟出一张网卡，数据只要经过网卡收发就可以进行拦截处理，后续整个操作系统的网络通讯都将通过这张虚拟的网卡进行收发。这和任何一个代理的实现思路都差不多，应用层并不知道网卡是虚拟的，这样 VPN 虚拟网卡将以中间人的身份对数据进行加工，从而实现各种神奇的效果。

但是，2016 年 11 月 7 日，我国通过了《中华人民共和国网络安全法》，网络监管日益制度化。在 2017 年的 1 月，工信部发布了《关于清理规范互联网网络接入服务市场的通知》（以下简称“《通知》”）。《通知》要求：“未经电信主管部门批准，不得自行建立或租用专线（含虚拟专用网络 VPN）等其他信道开展跨境经营活动。基础电信企业向用户出租的国际专线，应集中建立用户档案，向用户明确使

用用途仅供其内部办公专用，不得用于连接境内外的数据中心或业务平台开展电信业务经营活动”。通过该文我们可以看出，未经批准的 VPN 跨境经营活动被明确禁止。但基础电信企业提供的国际专线服务，则在用户内部办公专用的范围内被允许。

5. SS 与 VPN 的安全性与可识别性

VPN 极容易被禁用，因为 VPN 是走的专用通道，它是用来给企业传输加密数据用的，所以 VPN 的流量特征很明显，GFW 直接分析你的流量，如果特征匹配，直接封掉。目前就翻墙来说，PPTP 类型的 VPN 基本死的差不多了，L2TP 大部分地区干扰严重很不稳定。 而 Shadowsocks（SS）在传送资料时，用户的手机或电脑以「Socks5 代理」方式与网络服务器进行直接连线，两者之间**只会传送资料**，而不会像 VPN 在资料封包上加入容易被辨认的特征，并采用自行设计的加密演算法进行通讯（Socks5 代理本身不提供加密）。防火墙如果要检测流量特征的话，需要花大量时间。因此 Socks5 连线很容易穿过防火墙，而且被检测到的机会是相当之低。

虽然 SS 不会直接被识别为清楚明确的违禁标识，但是 SS 会被识别为未识别 TCP 业务流量，而正常的网站访问一般是可识别的，比如 SSL 安全类业务流量、微信流量这种。因此 GFW 会盯上这个未识别 TCP 业务流量，通过分析流量特征，知道你可能在翻墙。

ShadowsocksR（SSR）是建基于 Shadowsocks 之上开发的近似翻墙方式。它在 Shadowsocks 之上加入「数据混淆」与「协议转换」功能，可令资料加入模糊处理，令资料封包更难被辨认为 ShadowsocksR 数据，基本上都被识别为普通流量，混在所有的正常流量之中，不容易被有关方面查出，是更为安全的翻墙方法。所以 VPN 和 SS 两者的出发点和着重点就不同，VPN 更注重安全性，SS/SSR 更注重流量的混淆加密。

也就是说打个比方，在一条车流量很大的大路上：用 HTTP 的都是窗户透明且型号统一的出租车；用 HTTPS 的都是窗户不透明但是型号统一的出租车；用 SS、SSR、V2ray 原协议的都是一些窗户不透明的私家车，SS、SSR 用 HTTP/TLS 混淆的都是把私家车染成了出租车的样子；用 VPN 的都是坐的大巴，虽然关着窗户但知道肯定是大巴。未经批准的 VPN 跨境经营活动被明确禁止意味着大巴车不让出境了，当然私家车也是禁止的，只是他还可以伪装成出租车的样子。

2020.02.13

如何上网（二）　——HTTPS 与 SS/SSR

一. HTTPS 代理

HTTPS 代理的流量特征和我们平时访问网站时所产生的 HTTPS 流量几乎一摸一样，GFW 无法分辨，稳定性爆表。

理论上讲，HTTPS 代理无论是安全性，还是在隐匿性，都要比目前最为流行的 shadowsocks 好。事实上，在所有已知的翻墙协议中，无论是 VPN 协议，还是代理协议，它应该都是最好的。V2Ray 的 VMess over TLS 也许能和 HTTPS 代理媲美。但 V2Ray 存在的时间较短、使用者较少、社区也没有 HTTPS 代理活跃（从全球范围上看），故而，相比于 HTTPS 代理，VMess 协议潜在的安全漏洞可能要多。

当然，HTTPS 代理也有它的缺点，其中最大的缺点就是配置复杂。即便能用默认参数就用默认参数，用户自己只作最低限度的配置，对新手而言，这也是一个无比痛苦的过程。更别说，想要正常使用 HTTPS 代理，你还要购买域名和证书这些，非常麻烦。

例如，在搭建支持 HTTPS 的前端代理服务器时候，通常会遇到让人头痛的证书问题。根据 HTTPS 的工作原理，浏览器在访问一个 HTTPS 站点（在线代理网站）时，先与服务器建立 SSL 连接，建立连接的第一步就是请求服务器的证书。而服务器在发送证书的时候，是不知道浏览器访问的是哪个域名的，因为传输域名和其他数据同样需要 SSL 保护，所以不能根据不同域名发送不同的证书。对于在同一个 IP 部署不同 HTTPS 站点，并且还使用了不同证书的情况下，服务端怎么知道该发送哪个证书?

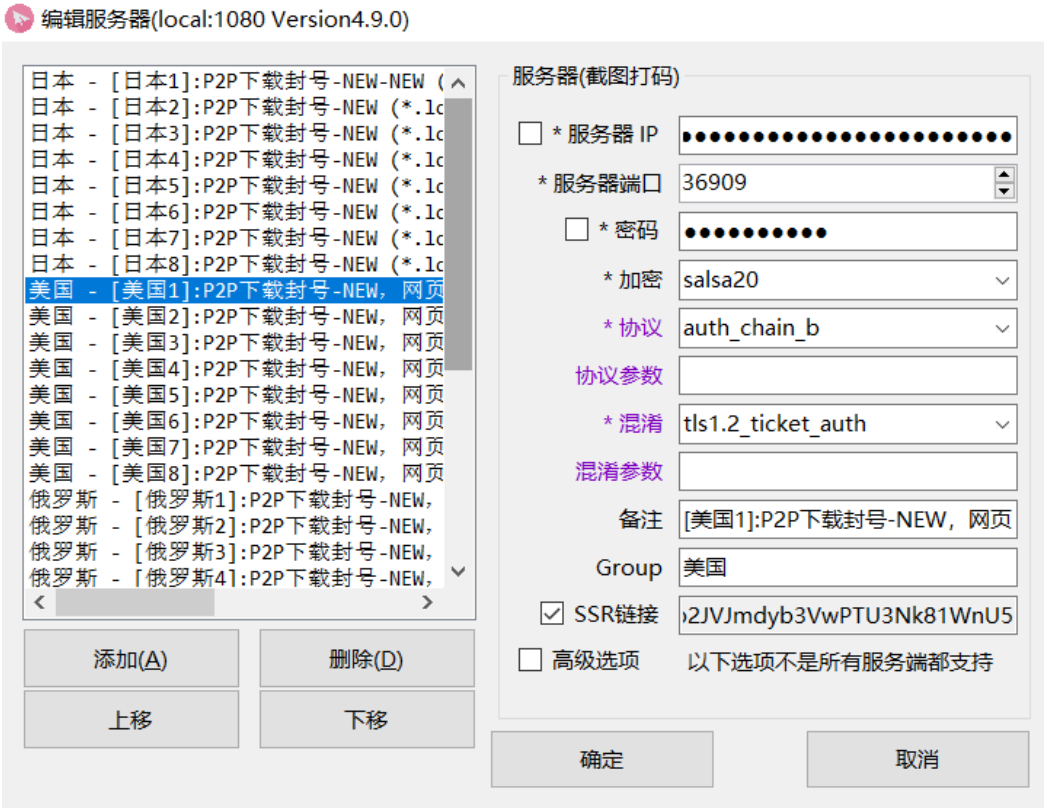
SNI（Server Name Indication）是为了解决一个服务器使用多个域名和证书的 SSL/TLS 扩展。一句话简述它的工作原理就是，在连接到服务器建立 SSL 链接之前先发送要访问站点的域名（Hostname），服务器根据这个域名返回一个合适的证书。这样，域名是可见的，当然由于自己建的代理网站的域名是自己取的，没有被加入黑名单，所以不会被 GFW 拦截掉。之后代理网站再用浏览器的 SNI 扩展，在境外访问那些敏感网站。好在目前，大多数操作系统和浏览器都已经很好地支持 SNI 扩展。但如果浏览器不支持 SNI 扩展，需要自行添加代理网站安全证书完成握手，还需要自行导入所有要访问网站的安全证书，封在数据包里发给在线代理，以备其进一步访问，十分麻烦。

所以，即便是在 shadowsocks 出现之前，HTTPS 代理也没在大陆流行起来。除此之外，HTTPS 代理只能转发 TCP 流量，对 UDP 无能为力。

二. SS、SSR 隐匿手段

SS 采用数据包加密传输给海外服务器的方式来躲避 GFW，让 GFW 无法窥探数据内部是什么，但是 GFW 可以检测各种数据包的外部特征。就像 VPN 带有明显的高度加密的特征，SS 发送的数据包虽然没有明显的某一特征，但其加密方式显然不是普通的 SSL/TLS 安全加密，而是一类无法识别的发往境外的数据包，无法识别也是一种特征。基于这种特征，GFW 会严密监测这种数据包的大小和走向。

SSR 是建基于 SS 之上开发的近似翻墙方式，它在 SS 之上加入「数据混淆」与「协议转换」功能。虽然现在的 SSR 都具有一键安装功能，参数是 SSR 代理商帮忙调整好的，但对于各项参数的理解还是很重要的。



一个典型的 SSR 示例

1. 加密

① RC4-MD5

速度最快，但加密简单，易破解。rc4 加密，md5 校验。

② AES 系列

AES（Advanced Encryption Standard）高级加密标准，这类加密标准是十分常见的对称加密算法。所谓的对称加密算法也就是加密和解密用相同的密钥。

区别	AES-XXX-CFB	AES-XXX-CTR	AES-XXX-GCM
定义			GMAC伽罗瓦消息验证码+CTR
区别	仅加密	仅加密	加密+消息完整性校验
优点	加密可并行	加密解密均可并行计算	加密解密均可并行计算
缺点	解密串行		

对于 CFB 模式来说，其全称为 Cipher Feed Back 模式（密文反馈模式）。在 CFB 模式中，前一个密文分组会被送回到密码算法的输入端。而所谓反馈，这里指的就是返回输入端的意思。

对于 GCM 模式来说，其全称为 Galois/Counter Mode，也就是该对称加密采用 Counter 模式，并带有 GMAC 消息认证码。

③ CHACHA20 系列

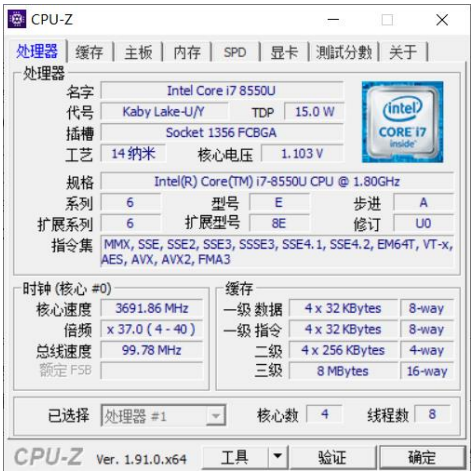
ChaCha2020 是 Salsa20 的改良，20 指 20 轮加密。

对于 ChaCha20-Poly1305，它是由 ChaCha20 流密码和 Poly1305 消息认证码（MAC）结合的一种加密算法，Poly1305 是一种消息完整性校验方式。ChaCha20-Poly1305 是基于 RC4 流加密的一种加密方式，它与 AES 有本质的区别，对于 RC4 而言，已经被证实并不安全，那么为什么还要发展 ChaCha20-Poly1305 呢？原因很简单，兼容性。

对于精简指令集的 ARM 平台，由于没有 AES-NI 指令集，ChaCha20-Poly1305 在同等配置的手机中表现是 AES 的 4 倍（ARM v8 之后加入了 AES 指令，所以在 ARM v8 平台上的设备，AES 方式反而比 chacha20-Poly1305 方式更快，性能更好），这样可减少加密解密所产生的数据量，使得性能更好。

总结：

- ✧ RC4-MD5,安全性几乎没保障，淘汰。
- ✧ AES 兼顾效率和安全。在拥有 AES 指令集的机器上（如右图），效率比 XCHACHA20 更高。推荐：AES-256-GCM。
- ✧ 相比 CHACHA20，XCHACHA20 兼顾效率和安全。在没有 AES 指令集的机器上，效率比 AES 高。推荐：XCHACHA20-IETF-POLY1305。



2. 数据混淆

数据混淆插件用于定义**加密后**的通信协议，通常用于**协议伪装**，部分插件能兼容原协议。

`plain`：表示不混淆，直接使用协议加密后的结果发送数据包。

`http_simple`：并非完全按照 http1.1 标准实现，仅仅做了一个头部的 GET 请求和一个简单的回应，之后依然为原协议流。使用这个混淆后，已在部分地区观察到似乎欺骗了 QoS 的结果。对于这种混淆，它并非为了减少特征，相反的是**提供一种强特征**。这是因为伪装成的 http 本身是不加密的，而这个数据包却加了密，所以要用一个更强的 http 的传输特征，来覆盖他加了密的本身，试图欺骗 GFW 的协议检测。要注意的是应用范围变大以后因特征明显有可能会被封锁。此插件可以兼容原协议（服务端配置为 `http_simple_compatible`），延迟与原协议几乎无异，除了头部数据包外没有冗余数据包，客户端支持自定义参数，参数为 http 请求的 host（这里由于 http 不加密，所以伪装的就不是 SNI），例如设置为 `cloudfront.com` 伪装为云服务器请求，可以使用逗号分割多个 host 如 `a.com,b.net,c.org`，这时会随机使用。不填写参数时，会使用此节点配置的**服务器地址**作为参数。

`http_post`：与 `http_simple` 绝大部分相同，区别是使用 POST 方式发送数据，符合 http 规范，欺骗性更好，但只有 POST 请求这种行为容易被统计分析出异常。可兼容原协议（需要在服务端配置为 `http_post_compatible`），参数配置与 `http_simple` 一样。

`random_head`（不建议使用）：开始通讯前发送一个几乎为随机的数据包，之后为原协议流。目标是让首个数据包根本不存在任何有效信息，让统计学习机制见鬼去吧，但也会**因此成为特征**。比原协议多一次握手导致连接时间会长一些，除了握手过程之后没有冗余数据包，**不支持自定义参数**。

`tls1.2_ticket_auth`（强烈推荐）:模拟 TLS1.2 在客户端有 session ticket 的情况下的握手连接。目前为完整模拟实现，经抓包软件测试完美伪装为 TLS1.2。因为有 ticket 所以没有发送证书等复杂步骤，因而防火墙无法根据证书做判断。同时自带一定的抗重放攻击的能力，以及包长度混淆能力。如遇到重放攻击则会在服务端 log 里搜索到，可以通过 `grep "replay attack"`搜索，可以用此插件发现你所在地区的线路有没有针对 TLS 的干扰。防火墙对 TLS 比较无能为力，抗封锁能力应该会较其它插件强，但遇到的干扰也可能不少，不过协议本身会检查出任何干扰，遇到干扰便断开连接，避免长时间等待，让客户端或浏览器自行重连。此插件可以兼容原协议（服务端配置为 `tls1.2_ticket_auth_compatible`），比原协议多一次握手导致连接时间会长一些，使用 C#客户端开启自动重连时比其它插件表现更好。

客户端**支持自定义参数，参数为 SNI**（非应用层代理是不需要 SNI 的，因为本身就不是 SSL 加密，只是伪装成 SSL 的样子。这里只是为了用假域名迷惑 GFW），**即发送 host 名称的字段**，此功能与 TOR 的 meek 插件相似，例如设置为 `cloudfront.net` 伪装为云服务器请求，可使用逗号分割多个 host 如 `a.com,b.net,c.org`，这时会随机使用。注意，错误设置此参数可能导致连接被断开甚至 IP 被封锁，如不清楚如何设置那么请留空。推荐自定义参数设置为 `cloudflare.com` 或 `cloudfront.net`。服务端暂不支持自定义参数。不填写参数时，会使用此节点配置的**服务器地址**作为参数。

总结：

name	encode speed	bandwidth	RTT	anti replay attack	cheat QoS	anti analysis
plain	100%	100%	0	No	0	/
http_simple	20%/100%	20%/100%	0	No	90	90
http_post	20%/100%	20%/100%	0	No	100	95
random_head (X)	100%	85%/100%	1	No	0	10
tls1.2_ticket_auth	98%	75%/ 95%	1	Yes	100	100

说明：

- ✧ 20%/100%表示首包为 20%，其余为 100%速度（或带宽）。其它的 RTT 大于 0 的混淆，前面的表示在浏览普通网页的情况下平均有效利用带宽的估计值，后一个表示去除握手响应以后的值，适用于大文件下载时（因为此时握手时间可忽略不计）。
- ✧ RTT 表示此混淆是否会产生附加的延迟，1 个 RTT 表示通讯数据一次来回所需要的时间。
- ✧ RTT 不为 0 且没有 anti replay attack 能力的混淆，不论协议是什么，都存在被主动探测的风险，因为 RTT 不为零也是一个特征。 RTT 为 0 的，只要协议不是 `origin`，就没有被主动探测的风险。当然由于原协议本身也存在被主动探测的风险，在没有观察到主动探测行为的情况下，不需要太担心。
- ✧ cheat QoS 表示欺骗路由器 QoS 的能力，100 表示能完美欺骗，0 表示没有任何作用，50 分左右表示较为严格的路由能识别出来。
- ✧ anti analysis 表示抗协议分析能力，`plain` 的时候依赖于协议，其它的基于网友反馈而给出的分值。值为 100 表示完美伪装。

3. 协议转换

此类型的插件用于定义**加密前**的协议，通常用于**长度混淆**及增强**安全性**和**隐蔽性**，部分插件能兼容原协议。

`origin`：表示使用原始 SS 协议，此配置速度最快效率最高，适用于限制少或审查宽松的环境。否则不建议使用。

`verify_deflate`（不建议）：对每一个包都进行 deflate 压缩，对于已经压缩过或加密过的数据将难以压缩，而对于未加密的 html 文本会有不错的压缩效果。因为压缩及解压缩较占 CPU，不建议较多用户同时使用此混淆插件。此插件**不能兼容原协议**，千万不要添加`_compatible` 的后缀。

`auth_sha1_v4`（不建议）：与 `auth_sha1` 对首个包进行 SHA-1 校验，同时会发送由客户端生成的随机客户端 id（4byte）、连接 id（4byte）、unix 时间戳（4byte），之后的通讯使用 Adler-32 作为效验码，对包长度单独校验，以抵抗抓包重放检测，使用较大的长度混淆，使用此插件的服务器与客户机的 UTC 时间差不能超过 24 小时，即只需要年份日期正确即可。兼容原协议（配置 `auth_sha1_v4_compatible`），支持服务端自定义参数，参数为 10 进制整数，表示最大客户端同时使用数。

`auth_aes128_md5` 或 `auth_aes128_sha1`（均推荐）：对首个包的认证部分进行使用 Encrypt-then-MAC 模式以真正免疫认证包的 CCA 攻击，预防各种探测和重防攻击，同时此协议支持单端口多用户。使用此插件的服务器与客户机的 UTC 时间差不能超过 24 小时，即只需要年份日期正确即可，针对 UDP 部分也有做简单的校验。此插件**不能兼容原协议**，支持服务端自定义参数，参数为 10 进制整数，表示最大客户端同时使用数。

`auth_chain_a`（强烈推荐）：对首个包的认证部分进行使用 Encrypt-then-MAC 模式以真正免疫认证包的 CCA 攻击，预防各种探测和重放攻击，**数据流自带 RC4 加密**，同时此协议支持单端口多用户，不同用户之间无法解密数据，每次加密密钥均不相同。使用此插件的服务器与客户机的 UTC 时间差不能超过 24 小时，即只需要年份日期正确即可，针对 UDP 部分也有加密及长度混淆。使用此插件建议加密使用 none。此插件**不能兼容原协议**，支持服务端自定义参数，参数为 10 进制整数，表示最大客户端同时使用数，最小值支持直接设置为 1，此插件能实时响应实际的客户端数量。

总结：

name	encode speed	bandwidth	anti CPA	anti CCA	anti replay attack	anti mid-man detect	anti packet length analysis
• origin	100%	99%	Yes	No	No	No	0
verify_simple	90%	96%	Yes	No	No	No	1
• verify_deflate	30%	97%~110%	Yes	No	No	No	6
verify_sha1	85%	98%/99%	Yes	No	No	No	0
auth_simple (X)	85%	95%	Yes	No	Yes	No	1
auth_sha1 (X)	95%	97%	Yes	No	Yes	No	4
auth_sha1_v2	94%	80%/97%	Yes	No	Yes	No	10
• auth_sha1_v4	90%	85%/98%	Yes	Yes?	Yes	No	10
• auth_aes128_md5	80%	90%/99%	Yes	Yes	Yes	Yes	10
• auth_aes128_sha1	70%	90%/99%	Yes	Yes	Yes	Yes	10

说明：

- ✧ 以上为浏览普通网页（非下载非看视频）的平均测试结果，浏览不同的网页会有不同的偏差。
- ✧ encode speed 仅用于提供相对速度的参考，不同环境下代码执行速度不同。
- ✧ `verify_deflate` 的 bandwidth（有效带宽）上限 110%仅为估值，若数据经过压缩或加密，那么压缩效果会很差。
- ✧ `verify_sha1` 的 bandwidth 意思为上传平均有效带宽 98%，下载 99%。
- ✧ `auth_sha1_v2` 的 bandwidth 在浏览普通网页时较低（为了较强的长度混淆，但单个数据包尺寸会保持在 1460 以内，所以其实对网速影响很小），而看视频或下载时有效数据比率比 `auth_sha1` 要高，可达 95%，所以不用担心下载时的速度。`auth_sha1_v4` 及 `auth_aes128_md5` 类似。
- ✧ 如果同时使用了其它的混淆插件，会令 bandwidth 的值降低，具体由所使用的混淆插件及所浏览的网页共同决定。
- ✧ 对于抗包长度分析一列，满分为 100，即 0 为完全无效果，5 以下为效果轻微。

4. 不同软件和混淆下的隐匿效果

软件和协议	运营商检测到的类型	显示地址
Shadowsocks	TCP 业务	显示服务器 IP
SS + http_simple（80端口）	上网（Web 方式 get）	显示混淆域名
SSR	TCP 业务	显示服务器 IP
SSR + http_simple	上网（Web 方式 get）	显示混淆域名
SSR + TLS（443）	安全类网页浏览（HTTPS VPN）流量 HTTPS 链接	显示混淆域名
SSR + TLS（非443）	网络连接（网页）HTTPS 链接	显示混淆域名

2020.02.14

常用加密方法

一. 加密方式分类

1. 摘要算法

消息摘要算法的主要特征是加密过程不需要密钥，并且经过加密的数据无法被解密。

特点：无论输入的消息有多长，计算出来的消息摘要的长度总是固定的。

应用：消息摘要算法主要应用在“数字签名”领域，作为对明文的摘要算法。

一般地，只要输入的消息不同，对其进行摘要以后产生的摘要消息也必不相同，但相同的输入必会产生相同的输出。这种算法主要包括:

（1）MD 算法（Message Digest），生成的消息摘要都是 128 位的，包括 MD2，MD4，MD5。

（2）SHA 算法（Secure Hash Algorithm），即安全散列算法，产生固定长度摘要信息:

算法	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
摘要长度	160	224	256	384	512

比较：都是从 MD4 发展而来，它们的结构和强度等特性有很多相似之处:

特性	**MD5**	**SHA-1**
摘要长度	128	160
报文最大长度	无穷大	2^24-1
分组长度	512b	512b
循环中步骤	64	80
基本函数	4	4

2. 对称加密算法

对称加密指加密和解密使用相同密钥的加密算法。

特点：对称加密算法的特点是算法公开、计算量小、加密速度快、加密效率高。不足之处是，交易双方都使用同样钥匙，安全性得不到保证。

应用：数据传输中的加密，防窃取。这种算法主要包括:

（1）DES（Data Encryption Standard）：DES 是一个分组加密算法，典型的 DES 以 64 位为分组对数据加密，加密和解密用的是同一个算法。

（2）AES（Advanced Encryption Standard）：AES 算法基于排列和置换运算。排列是对数据重新进行安排，置换是将一个数据单元替换为另一个。是美国联邦政府采用的一种区块加密标准，这个标准用来替代原先的 DES，运用最为广泛。

比较：AES 弥补了 DES 很多的不足，支持秘钥变长，分组变长，更加的安全，对内存要求非常低。

特性	DES	AES
速度	中	快
秘钥长度	56b	128~256（32 倍数）
分组长度	64b	128~256（32 倍数）
是否可枚举破解	趋于可能	目前不可

3. 非对称加密算法

非对称加密算法需要两个密钥：公开密钥和私有密钥。公钥与私钥是一对，如果用公钥对数据进行加密，只有用对应的私钥才能解密。用私钥进行加密，只有对应的公钥才能进行解密。

特点：算法强度复杂、安全性依赖于算法与密钥。但是由于其算法复杂，而使得加密解密速度没有对称加密解密的速度快。这种算法是由数学上的难解问题构造的。

应用场景：数字签名、秘钥传输加密。这种算法主要包括:

（1）DH（Diffie-Hellman）：非对称算法的基石，秘钥交换协议

（2）RSA（RSA algorithm）：非对称加密的经典，除了可用于非对称加密，也可用于数字签名。

（3）ECC（Elliptic curve cryptography）：椭圆加密算法

比较：使用 RSA，可以进行加密和签名的密钥对。使用 DH，只执行加密，没有签名机制。ECC 和 RSA 相比，在许多方面都有对绝对的优势，下图可以看出，相同秘钥长度下，ECC 更难破译。

攻破时间	RSA/DSA	ECC密钥长度	RSA/ECC
MIPS年	密钥长度		密钥长度比
10^4	512	106	5:1
10^8	768	132	6:1
10^{11}	1024	160	7:1
10^{20}	2048	210	10:1
10^{78}	21000	600	35:1

二. Base64 位加密（可加密解密）

最简单的加密方式，固定规则加密，没有密钥，这种方式只要让别人拿到你的密文，就可以直接解密，只能用来迷惑，一般情况下不单独使用，因为真的并没有什么用，可以和其他加密方式混合起来，作为一层外部包装。

具体转换步骤：

第一步　　将待转换的字符串转为一个个字符；

第二步　　计算每一个字符对应的 ASCII 码十进制；

第三步　　计算出十进制对应的二进制，若不足 8 位，在前面添加 0 进行补全；

第四步　　将 8 位的二进制码，按照 6 个为一组划分，若不能整除 6，在最末添加 0 补足 6 位；

第五步　　计算对应的十进制编码；

第六步　　按照 base64 表，查看对应的字符；

第七步　　将加密后的字符每 3 个分成一组，不足 3 位的添加=进行补全；

第八步　　得到最终结果。

同时得到 base64 位加密后的特征：长度是 3 的倍数；只含有 65 中字符，大写的 A 至 Z，小写的 a 至 z,数字 0 到 9，以及 3 种符号+/ =，=最多两个，且在末尾。

原始字符	J	a	s	m	i	n	e					
ASCII码	74	97	115	109	105	110	101					
二进制值	01001010	01100001	01110011	01101101	01101001	01101110	01100101					
重新进行划分	010010	100110	000101	110011	011011	010110	100101	101110	011001	010000		
Base64	18	38	5	51	27	14	22	46	25	16		
Base64加密	S	m	F	z	b	O	W	u	Z	Q		
每三位	SmF			zbO			WuZ			Q==		
最终结果	SmFzbWluZQ==											

字符串 Jasmine 的加密

A	0	Q	16	g	32	w	48
B	1	R	17	h	33	x	49
C	2	S	18	i	34	y	50
D	3	T	19	j	35	z	51
E	4	U	20	k	36	0	52
F	5	V	21	l	37	1	53
G	6	W	22	m	38	2	54
H	7	X	23	n	39	3	55
I	8	Y	24	o	40	4	56
J	9	Z	25	p	41	5	57
K	10	a	26	q	42	6	58
L	11	b	27	r	43	7	59
M	12	c	28	s	44	8	60
N	13	d	29	t	45	9	61
O	14	e	30	u	46	+	62
P	15	f	31	v	47	/	63

Base64 转换表

三. MD5 加密（加密不可逆）

MD5 的全称是 Message-Digest Algorithm 5（信息-摘要算法）。 MD5 以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，经过了一系列的处理后，算法的输出由**四个 32 位分组**组成，将这四个 32 位分组级联后将生成一个 128 位散列值。

1. 算法流程

在 MD5 算法中，首先需要对信息进行填充，填充第一位为 1，其余为 0，使其字节长度对 512 求余数的结果等于 448。因此，信息的字节长度（Bits Length）将被扩展至 N*512+448，即 N*64+56 个字节（Bytes），N 为一个**非负整数**。然后再在这个结果后面附加一个以 64 位二进制表示的**填充前的信息长度**，如果信息长度大于 2^{64}，则只使用其低 64 位的值，即消息长度对 2^{64} 取余数。经过这两步的处理，现在的信息字节长度=N*512+448+64=(N+1)*512，即长度恰好是 512 的整数倍数。这样做的原因是为满足后面处理中对信息长度的要求。

MD5 中有四个 32 位被称作链接变量（Chaining Variable）的整数参数，以 16 进制表示：

```
A=0x01234567
B=0x89abcdef
C=0xfedcba98
D=0x76543210
```

四个基础非线性函数（异或正规表达为⊕）：

```
F(X,Y,Z) = (X & Y) | ((~X) & Z);
G(X,Y,Z) = (X & Z) | (Y & (~Z));
H(X,Y,Z) = X ^ Y ^ Z;
I(X,Y,Z) = Y ^ (X | (~Z));

(&是与,|是或,~是非,^是异或)
```

四个数据处理函数： Mj 表示第 M 组第 j+1 个子分组（M∈[1,N+1]，j∈[0,15]，M, j∈Z）。ti 是一个唯一的加法常数，ti 可以这样选择：在第 i 步中，ti 是 4294967296*|sin(i)|的整数部分，i 的单位是弧度。

```
1. FF(a,b,c,d,Mj,s,ti): a = b + ((a + F(b,c,d) + Mj + ti) <<< s)
2. GG(a,b,c,d,Mj,s,ti): a = b + ((a + G(b,c,d) + Mj + ti) <<< s)
3. HH(a,b,c,d,Mj,s,ti): a = b + ((a + H(b,c,d) + Mj + ti) <<< s)
4. II(a,b,c,d,Mj,s,ti): a = b + ((a + I(b,c,d) + Mj + ti) <<< s)
```

当设置好这四个链接变量后，就开始进入算法的四轮循环运算，循环的次数是信息中 512 位信息分组的数目 N+1。将上面四个链接变量复制到另外四个变量中：A 到 a，B 到 b，C 到 c，D 到 d。主循环有四轮（MD4 只有三轮），每轮循环都很相似。

一轮进行 16 次操作。每次操作对 a、b、c 和 d 中的其中三个作一次非线性函数运算，然后将所得结果加上第四个变量、文本中的一个子分组 Mj 和一个常数 ti。再将所得结果向右环移一个不定的数 s，并加上 a、b、c 或 d 中之一，得到一个 32 位数。最后用该结果取代 a、b、c 或 d 中之一。每轮操作中分别用到四个非线性函数中的一个。四轮循环 64 次操作结束为一次循环。每次循环结束后，将 A、B、C、D 分别加上 a、b、c、d，N+1 次循环后得到的 A、B、C、D 的级联就是输出结果，A 是低位，D 为高位，DCBA 组成 128 位输出结果。每次循环的流程如下图所示：

```
第一轮
FF(a,b,c,d,M0,7,0xd76aa478)
FF(d,a,b,c,M1,12,0xe8c7b756)
FF(c,d,a,b,M2,17,0x242070db)
FF(b,c,d,a,M3,22,0xc1bdceee)
FF(a,b,c,d,M4,7,0xf57c0faf)
FF(d,a,b,c,M5,12,0x4787c62a)
FF(c,d,a,b,M6,17,0xa8304613)
FF(b,c,d,a,M7,22,0xfd469501)
FF(a,b,c,d,M8,7,0x698098d8)
FF(d,a,b,c,M9,12,0x8b44f7af)
FF(c,d,a,b,M10,17,0xffff5bb1)
FF(b,c,d,a,M11,22,0x895cd7be)
FF(a,b,c,d,M12,7,0x6b901122)
FF(d,a,b,c,M13,12,0xfd987193)
FF(c,d,a,b,M14,17,0xa679438e)
FF(b,c,d,a,M15,22,0x49b40821)
```

```
第二轮
GG(a,b,c,d,M1,5,0xf61e2562)
GG(d,a,b,c,M6,9,0xc040b340)
GG(c,d,a,b,M11,14,0x265e5a51)
GG(b,c,d,a,M0,20,0xe9b6c7aa)
GG(a,b,c,d,M5,5,0xd62f105d)
GG(d,a,b,c,M10,9,0x02441453)
GG(c,d,a,b,M15,14,0xd8a1e681)
GG(b,c,d,a,M4,20,0xe7d3fbc8)
GG(a,b,c,d,M9,5,0x21e1cde6)
GG(d,a,b,c,M14,9,0xc33707d6)
GG(c,d,a,b,M3,14,0xf4d50d87)
GG(b,c,d,a,M8,20,0x455a14ed)
GG(a,b,c,d,M13,5,0xa9e3e905)
GG(d,a,b,c,M2,9,0xfcefa3f8)
GG(c,d,a,b,M7,14,0x676f02d9)
GG(b,c,d,a,M12,20,0x8d2a4c8a)
```

```
第三轮
HH(a,b,c,d,M5,4,0xfffa3942)
HH(d,a,b,c,M8,11,0x8771f681)
HH(c,d,a,b,M11,16,0x6d9d6122)
HH(b,c,d,a,M14,23,0xfde5380c)
HH(a,b,c,d,M1,4,0xa4beea44)
HH(d,a,b,c,M4,11,0x4bdecfa9)
HH(c,d,a,b,M7,16,0xf6bb4b60)
HH(b,c,d,a,M10,23,0xbebfbc70)
HH(a,b,c,d,M13,4,0x289b7ec6)
HH(d,a,b,c,M0,11,0xeaa127fa)
HH(c,d,a,b,M3,16,0xd4ef3085)
HH(b,c,d,a,M6,23,0x04881d05)
HH(a,b,c,d,M9,4,0xd9d4d039)
HH(d,a,b,c,M12,11,0xe6db99e5)
HH(c,d,a,b,M15,16,0x1fa27cf8)
HH(b,c,d,a,M2,23,0xc4ac5665)
```

```
第四轮
II(a,b,c,d,M0,6,0xf4292244)
II(d,a,b,c,M7,10,0x432aff97)
II(c,d,a,b,M14,15,0xab9423a7)
II(b,c,d,a,M5,21,0xfc93a039)
II(a,b,c,d,M12,6,0x655b59c3)
II(d,a,b,c,M3,10,0x8f0ccc92)
II(c,d,a,b,M10,15,0xffeff47d)
II(b,c,d,a,M1,21,0x85845dd1)
II(a,b,c,d,M8,6,0x6fa87e4f)
II(d,a,b,c,M15,10,0xfe2ce6e0)
II(c,d,a,b,M6,15,0xa3014314)
II(b,c,d,a,M13,21,0x4e0811a1)
II(a,b,c,d,M4,6,0xf7537e82)
II(d,a,b,c,M11,10,0xbd3af235)
II(c,d,a,b,M2,15,0x2ad7d2bb)
II(b,c,d,a,M9,21,0xeb86d391)
```

四轮 64 步操作流程图

2. 特点

单向性：目前 MD5 是一种不可逆算法，即使给了加密算法和结果，也是无法逆推还原成原文的。原因有以下两点：

① 在每次循环结束后，是将 A、B、C、D 分别加上 a、b、c、d 得到新的 A、B、C、D，即由两个未知量相加得到的第三个未知量，所以无固定拆分。相当于解以下线性方程组（其中 K 已知，U 未知）：

$$K_1+U_1=U_2，U_2+U_3=U_4，\cdots\cdots，U_{2N}+U_{2N+1}=K2;$$

N > 0 时，方程组有 2N+1 个未知量，N+1 个方程，2N+1 > N+1，方程无唯一解，所以从 K_1 到 K_2 是无唯一路径的。N=0 时，$K_1+U_1=K_2$，有固定 U_1。

② 但即使知道每次循环里 64 次操作的起始值和最终值（如上面的 K_1 和 U_1），也无法得到原文。这是因为 64 次中，每一次的数据处理函数都会产生一个未知量，就是该次操作中被赋值的那个链接变量的下一次取值，除去最后 U_1 确定的四个未知变量，再加上 16 个子分组未知，一共有 76 个未知变量。64 个方程，76 个未知变量，而且还是非线性函数，故无唯一解。

从上面我们也可以看出，如果只有四个子分组 128 个字节输入，则未知量个数等于方程个数，方程可能有解，此时输入长度等于输出长度。所以当输入长度小于等于输出长度时该算法就可能可逆，因为只有输入集小于输出集才会有一一对应的可能，从而避免碰撞性。

恒定性：是任意一段明文数据，经过散列以后，其结果是永远是不变的。

不可预测性：每一点对于原文微小的修改，都会导致 MD5 散列值完全不同。

碰撞性：明文数据经过散列以后的值是定长的，**一定存在**有两段明文散列以后得到相同的结果。

3. 安全性

① 安全

MD5 相对 MD4 所作的改进：增加了第四轮；每一步均有唯一的加法常数；减弱第二轮中函数的对称性；第一步加上了上一步的结果，这将引起更快的雪崩效应（就是对明文或者密钥改变 1bit 都会引起密文的巨大不同）；改变了第二轮和第三轮中访问消息子分组的次序，使其更不相似；近似优化了每一轮中的循环左移位移量以实现更快的雪崩效应，各轮的位移量互不相同。

MD5 的输出为 128 位，若采用纯强力攻击寻找一个消息具有给定 Hash 值的计算困难性为 2128，用每秒可试验 1000000000 个消息的计算机需时 1.07×1022 年。若采用生日攻击法，寻找有相同 Hash 值的两个消息需要试验 264 个消息，用每秒可试验 1000000000 个消息的计算机需时 585 年。

② 不安全

MD5 算法自诞生之日起，就有很多人试图证明和发现它的不安全之处，即存在碰撞（在对两个不同的内容使用 MD5 算法运算的时候，有可能得到一对相同的结果值）。2009 年，中国科学院的谢涛和冯登国仅用了$2^{20.96}$的碰撞算法复杂度，破解了 MD5 的碰撞抵抗，该攻击在普通计算机上运行只需要数秒。

现在有的组织通过穷举字符组合的方式，创建了明文密文对应查询数据库，创建的记录约 90 万亿条，占用硬盘超过 500TB。保存在数据库的加密 MD5 值，当网站存在注入或者其它漏洞时，入侵者极有可能获取用户的密码值，通过 MD5 在线查询或者暴力破解可以得到密码。对此，一种改进是，在使用 MD5 加密算法对明文（口令）加密的基础上，对密文进行了改变，在密文中截取一段数据并丢弃，然后使用随机函数填充被丢弃的数据，且整个过程不改变 MD5 加密后的位数。

4. 应用

当需要保存某些密码信息以用于身份确认时，如果直接将密码信息以明码方式保存在数据库中，不使用任何保密措施，系统管理员就很容易能得到原来的密码信息，这些信息一旦泄露，密码也很容易被得到。为了增加安全性，有必要对数据库中需要保密的信息进行加密，这样，即使有人得到了整个数据库，如果没有解密算法，也不能得到原来的密码信息。

MD5 算法可以很好地解决这个问题，因为它可以将任意长度的输入经过计算得到固定长度的输出，而且只有在明文相同的情况下，才能等到相同的密文，并且这个算法是不可逆的，即便得到了加密以后的密文，也不可能通过解密算法反算出明文。

这样就可以把用户的密码以 MD5 值（或类似的其它算法）的方式保存起来，用户注册的时候，系统是把用户输入的密码计算成 MD5 值，然后再去和系统中保存的 MD5 值进行比较，即进行**校验**，如果密文相同，就可以认定密码是正确的，否则密码错误。

通过这样的步骤，系统在并不知道用户密码明码的情况下就可以确定用户登录系统的合法性。这样不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。但由于网站常用的密码一般也就 6-10 位，这个取值空间还是很好破解的。为了提高破解的难度，还可以将密码 + 用户名作为 MD5 原始输入参数。 2020.02.15

MD5 算法还可以作为一种电子签名的方法来使用，使用 MD5 算法就可以为任何文件（不管其大小、格式、数量）产生一个独一无二的“数字指纹”，借助这个“数字指纹”，通过检查文件前后 MD5 值是否发生了改变，就可以知道源文件是否被改动，文件内容只要有一点改动，MD5 值就会有大的变动。

四. SHA1 加密（加密不可逆）

SHA1 的全称是 Secure Hash Algorithm（安全哈希算法）。

1. SHA1 与 MD5 差异

SHA1 对任意长度明文的预处理和 MD5 的过程是一样的，即预处理完后的明文长度是 512 位的整数倍。但是不同的是，SHA1 的原始报文长度不能超过2^{64}，且 SHA1 生成 160 位的报文摘要。SHA1 算法简单而且紧凑，容易在计算机上实现。

差异处	MD5	SHA1
摘要长度	128 位	160 位
运算步骤数	64	80
基本逻辑函数数目	4	4
常数数目	64	4

MD5 与 SHA1 的比较

- 安全性：SHA1 所产生的摘要比 MD5 长 32 位。若两种散列函数在结构上没有任何问题的话，SHA1 比 MD5 更安全。
- 速度：两种方法都是主要考虑以 32 位处理器为基础的系统结构。但 SHA1 的运算步骤比 MD5 多了 16 步，而且 SHA1 记录单元的长度比 MD5 多了 32 位。因此若是以硬件来实现 SHA1，其速度大约比 MD5 慢了 25%。
- 简易性：两种方法都是相当的简单，在实现上不需要很复杂的程序或是大量存储空间。然而总体上来讲，SHA1 对每一步骤的操作描述比 MD5 简单。

2. 算法流程

对于任意长度的明文，SHA1 的明文分组过程与 MD5 相类似，首先需要对明文添加位数，使明文总长度为 448（mod512）位。在明文后添加位的方法是第一个添加位是 1，其余都是 0。然后将真正明文的长度（没有添加位以前的明文长度）以 64 位表示，附加于前面已添加过位的明文后，与 MD5 不同的是 SHA1 的明文长度从低位开始填充（倒过来写）。此时的明文长度正好是 512 位的倍数，然后按 512 位的长度进行分组（block），可以划分成 L 份明文分组。

对于 512 位的明文分组，SHA1 将其再分成 16 份子分组（sub-block），每份子分组为 32 位，我们使用M_k（k= 0, 1,……15）来表示这 16 份子分组。之后还要将这 16 份子分组**扩充到 80 份**子分组，我们记为W_k（k= 0, 1,……79），扩充的方法如下，其中⊕表示异或：

$$W_k = M_k , \quad t \in [0,15]$$

$$W_k = (W_{k-3} \oplus W_{k-8} \oplus W_{k-14} \oplus W_{k-16}) <<< 1, \quad t \in [16,79]$$

SHA1 有 5 个链接变量的初始值以 16 进制表示如下：

A=0x67452301； B=0xEFCDAB89； C=0x98BADCFE； D=0x10325476； E=0xC3D2E1F0

SHA1 有 4 轮运算，每一轮包括 20 个步骤，一共 80 步，当第 1 轮运算中的第 1 步骤开始处理时，A、B、C、D、E 五个链接变量中的值先赋值到另外 5 个记录单元 A′，B′，C′，D′，E′中。这 5 个值将保留，用于在第 4 轮的最后一个步骤完成之后与链接变量 A，B，C，D，E 进行求和操作。

SHA1 的 4 轮运算，共 80 个步骤**使用同一个操作程序**，如下：

A, B, C, D, E ← [(A<<< 5) + ft (B, C, D) + E+ Wt + Kt], A, (B<<< 30), C, D

轮	步骤	函数定义
1	0 ≤ t ≤ 19	ft (B, C, D) = (B&C) | (~B&D)
2	20 ≤ t ≤ 39	ft (B, C, D) = B ⊕ C ⊕ D
3	40 ≤ t ≤ 59	ft (B, C, D) = (B&C) | (B&D) | (C&D)
4	60 ≤ t ≤ 79	ft (B, C, D) =B ⊕ C ⊕ D

SHA1 的逻辑函数

轮	步骤	函数定义	轮	步骤	函数定义
1	0 ≤ t ≤ 19	Kt = 5A827999	3	40 ≤ t ≤ 59	Kt = 8F188CDC
2	20 ≤ t ≤ 39	Kt = 6ED9EBA1	4	60 ≤ t ≤ 79	Kt = CA62C1D6

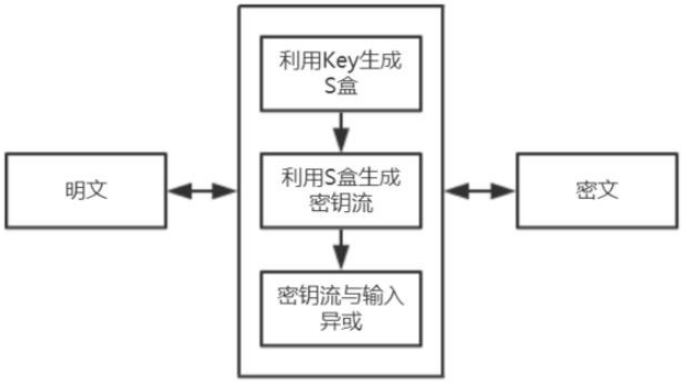
SHA1 的常数 K 取值表

第四轮最后一个步骤的 A，B，C，D，E 输出后，将分别与记录单元 A′，B′，C′，D′，E′中的数值求和运算。其结果将作为输入成为下一个 512 位明文分组的链接变量 A，B，C，D，E，当最后一个明文分组计算完成以后，A，B，C，D，E 中的数据就是最后散列函数值。

五. RC4（Rivest Cipher 4）加密

RC4（Rivest Cipher 4）由 RSA Security 的 Ron Rivest 在 1987 年设计，因为其速度和简单性，这种加密算法已经成为使用最为广泛的流密码。RC4 属于**对称秘钥加密**，被用于常用协议中，包括有线等效保密（WEP），无线网络的安全算法，以及 HTTPS 的安全套接字层（SSL）和传输层安全（TLS）协议。

1. RC4 算法流程



RC4算法流程图

① 先初始化**状态向量 S**（用来作为密钥流生成的种子 1），每一个单元都是一个字节。算法不论执行到什么时候，S 都包含 0-255 的 8 比特数的排列组合，仅仅只是值的位置发生了变换。首先按照升序，给每个字节赋值 0，1，2，3，4，5，6，……，254，255。for (int i=0；i<256；i++) {S[i] = i}

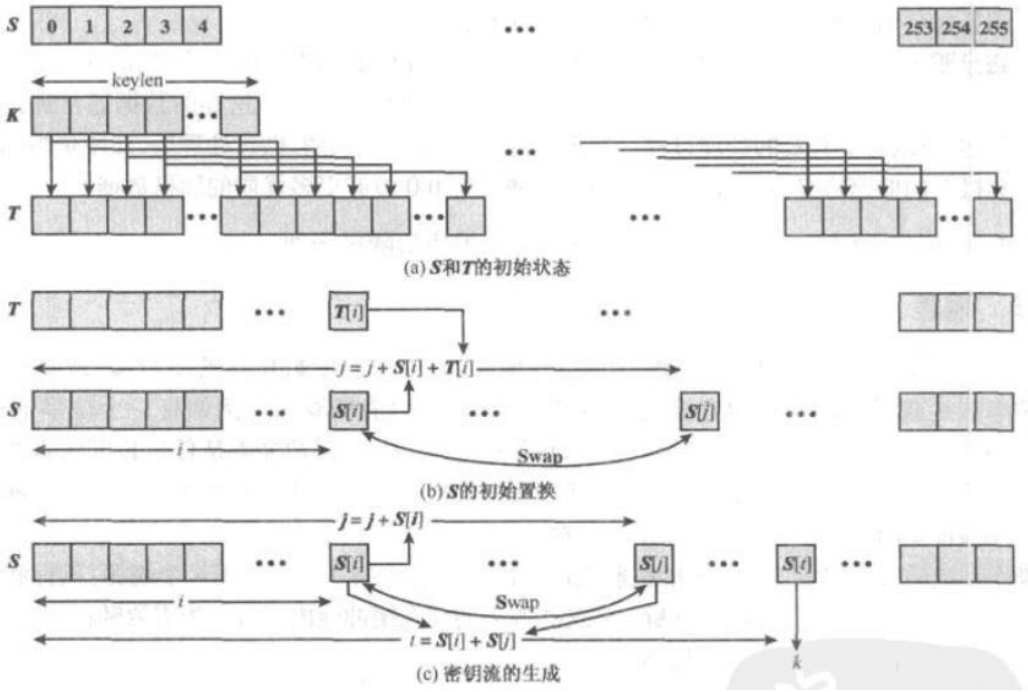
② 由用户输入长度任意的初始**密钥 K**，并填入**暂时向量 T**，如果输入长度小于 256 个字节，则进行轮转，直到填满 256 个字节，例如输入密钥的是 1,2,3,4,5 ，那么填入的是 1,2,3,4,5, 1,2,3,4,5, 1,2,3,4,5……。由上述轮转过程得到 256 个字节的暂时向量 T（用来作为密钥流生成的种子 2）。

③ 对状态向量 S 进行**置换操作**（用来打乱初始种子 1）：初始值 i，j = 0，接下来取 S[i]和 T[i]的值与 j 相加，对 256 取余，更新 j 的值；S[i]与 S[j]的值互换，i 的值加 1，继续循环上述操作，直到 i =255。这样从第 0 个字节开始，一共执行了 256 次，保证状态向量 S 的每个字节都得到处理，几乎是带有一定的随机性了。 j = 0；for (i = 0 ; i < 256 ; i++) {j = (j + S[i] + T[i]) mod 256；swap(S[i] , S[j])；}

④ 采用伪随机数算法（The pseudo-random generation algorithm(PRGA)），利用 S 盒生成密钥流：

i = (i + 1) mod 256；j = (j + S[i]) mod 256；swap(S[i] , S[j])；t = (S[i] + S[j]) mod 256；K = S[t]，这里的

K 就是当前生成的一个秘钥流中的一位，可以直接在这里进行加密，当然也可以将密钥流保存在数组中，最后进行异或操作 data[]=data[]^k（“^”是异或运算符）。



2. 安全性

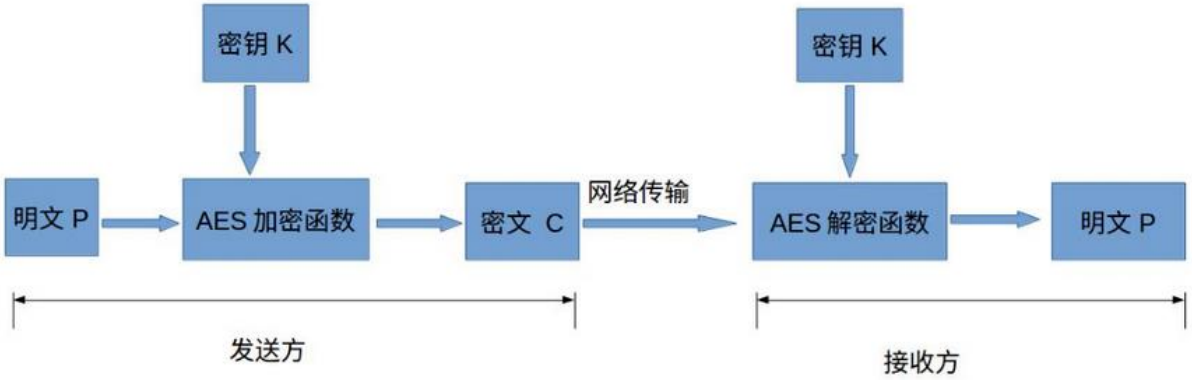
RC4 加密并不安全，RC4 采用初始化向量 IV 和秘钥 K 链接，通过伪随机数算法构成 key。由于**秘钥 K 长期固定**，初始化向量 IV 也是固定的，伪随机数算法看似随机但是固定，所以同一个 IV 对应的 key 相同。又在传输过程中 IV 明文传输，通过大数据分析 IV 的值，可以得到相同 IV 的数据是如何加密的。

虽然不知道 K 的具体值，但通过异或运算，在已知两个密文 C1、C2 和一个密文对应明文 M1 的情况下，可以计算出另外一个密文对应的明文，即$\mathrm{M1} \oplus \mathrm{RC4}\,(\mathrm{IV},\mathrm{K}) = \mathrm{C1}$，$\mathrm{M2} \oplus \mathrm{RC4}\,(\mathrm{IV},\mathrm{K}) = \mathrm{C2}$，于是$\mathrm{M2} = \mathrm{M1} \oplus \mathrm{C1} \oplus \mathrm{C2}$。这也是流密码普遍存在的问题。目前采用中间人攻击方法，通过不断获取不同 IV 值下明文对应的密文，来攻破 HTTPS 保护的网站理论上只需要 75 小时，建议完全停止使用。

所以最好定期更换秘钥，并通过 RSA 进行加密传输！

六. AES 加密（需要密钥才能解密）

AES 加密为**对称密钥加密**，加密和解密都是用同一个解密规则，AES 加密过程是在一个 4×4 的字节矩阵上运作，这个矩阵又称为"状态(state)"，因为密钥和加密块要在矩阵上多次的迭代，置换，组合，所以对加密块和密钥的字节数都有一定的要求，AES 密钥长度的最少支持为 128、192、256，加密块**分组**长度 128 位。这种加密模式有一个最大弱点：甲方必须把加密规则告诉乙方，否则无法解密。保存和传递密钥，就成了最头疼的问题。



ASE 加密流程

- 明文 P：没有经过加密的数据。
- 密钥 K：用来加密明文的密码，在对称加密算法中，加密与解密的密钥是相同的。密钥为接收方与发送方**协商产生**，但不可以直接在网络上传输，否则会导致密钥泄漏，通常是**通过非对称加密算法加密密钥**，然后再通过网络传输给对方，或者直接面对面商量密钥。密钥是绝对不可以泄漏的，否则会被攻击者还原密文，窃取机密数据。实际中，一般是**通过 RSA 加密 AES 的密钥**，传输到接收方，接收方解密得到 AES 密钥，然后发送方和接收方用 AES 密钥来通信。
- AES 加密函数：设 AES 加密函数为 E，则 C = E (K, P)，其中 P 为明文，K 为密钥，C 为密文。也就是说，把明文 P 和密钥 K 作为加密函数的参数输入，则加密函数 E 会输出密文 C。
- 密文 C：经加密函数处理后的数据
- AES 解密函数：设 AES 解密函数为 D，则 P = D (K, C)，其中 C 为密文，K 为密钥，P 为明文。也就是说，把密文 C 和密钥 K 作为解密函数的参数输入，则解密函数会输出明文 P。

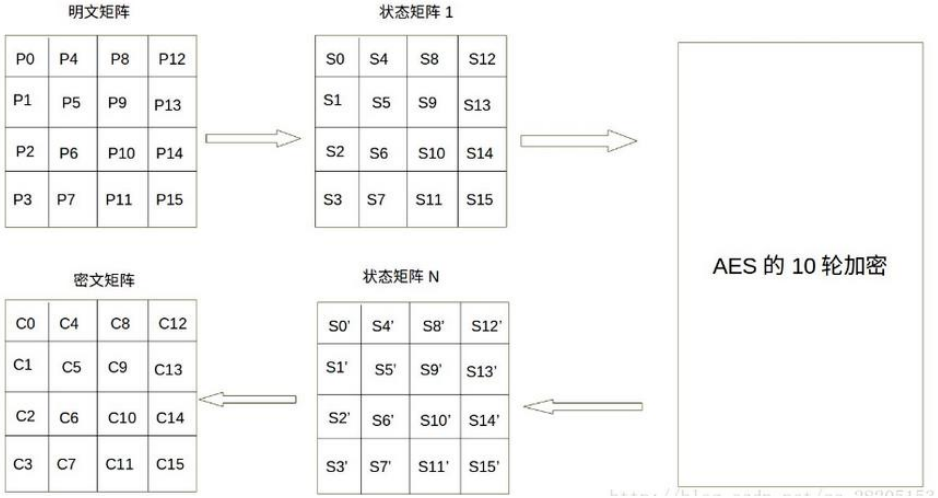
1. AES 的基本流程

AES 为分组密码，即把明文分组，每组长度相等，每次加密一组数据，直到加密完整个明文。在 AES 标准规范中，分组长度只能是 128 位，也就是说，每个分组为 16 个字节（每个字节 8 位）。密钥的长度可以使用 128 位、192 位或 256 位。密钥的长度不同，推荐加密轮数也不同：

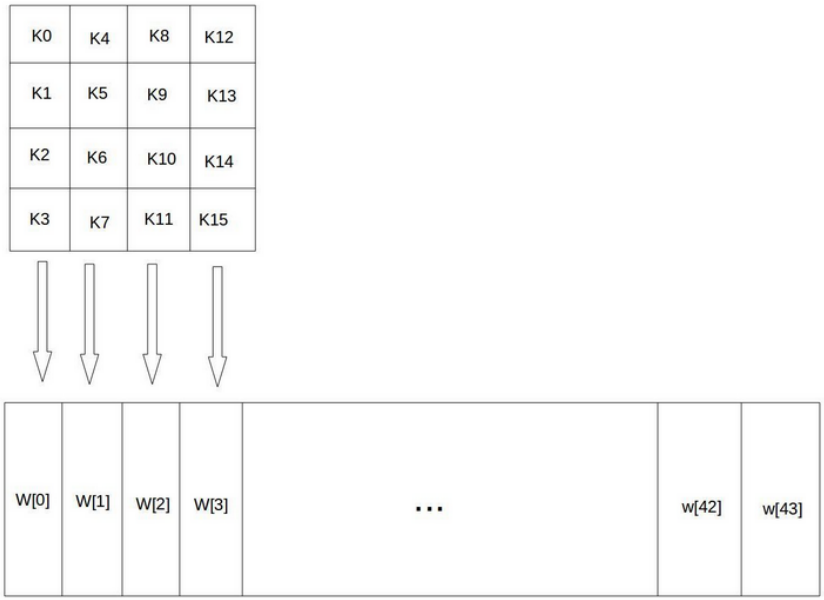
AES	密钥长度 (32 位比特字)	分组长度 (32 位比特字)	加密轮数
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

上面说到，AES 的加密公式为 C = E(K,P)，在加密函数 E 中，会执行一个轮函数，并且执行 10 次这个轮函数，这个轮函数的前 9 次执行的操作是一样的，只有第 10 次有所不同。也就是说，一个明文分组会被加密 10 轮。AES 的核心就是**实现一轮中的所有操作**。

AES 的处理单位是字节，以 AES-128（密钥的长度为 128 位）为例，128 位的**输入明文**分组 P 和输入密钥 K 都被分成 16 个组，每组一个字节 8 位（一个 ASCII 码的长度），分别记为 P = P0，P1，……，P15 和 K = K0，K1，……，K15。一般地，明文分组用以字节为单位的正方形矩阵描述，称为状态矩阵。在算法的每一轮中，状态矩阵的内容不断发生变化，最后的结果作为密文输出。该矩阵中字节的排列顺序为从上到下、从左至右依次排列，如下图所示：

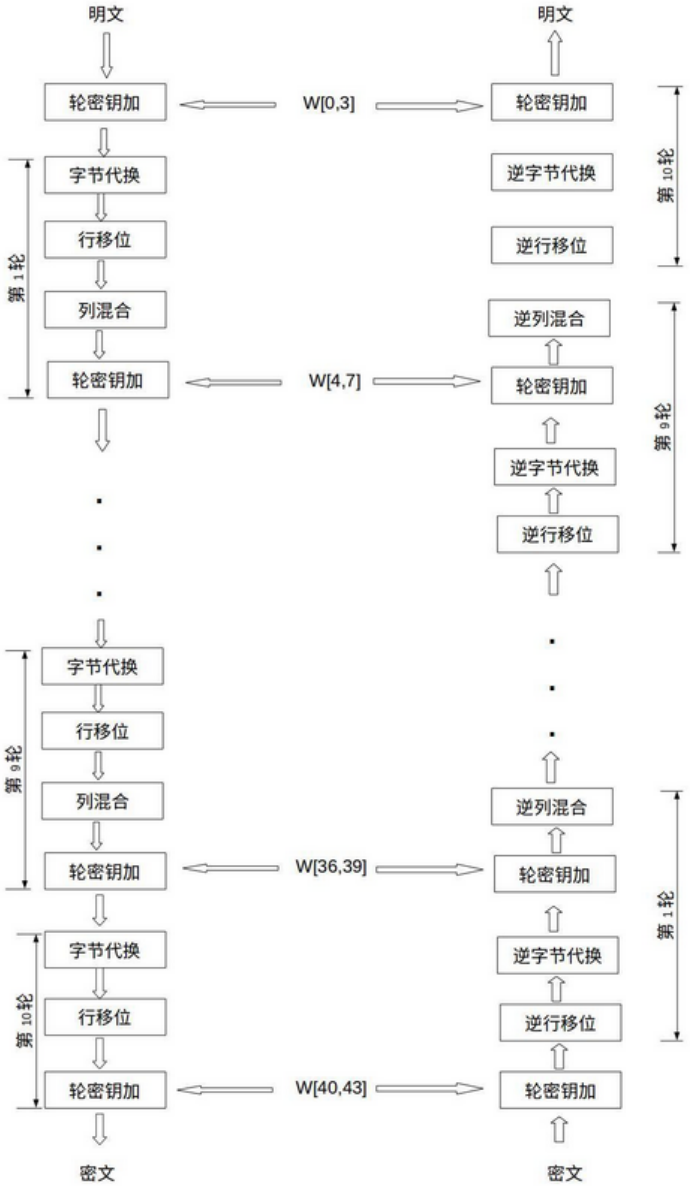


类似地，128 位**密钥**也是用以字节为单位的矩阵表示，矩阵的每一列被称为 1 个 32 位比特字。通过**密钥编排函数**该密钥矩阵被扩展成一个包含 44 元素的序列：W[0]，W[1]，…… ，W[43]，每个元素包含 1 个 32 位比特字。该序列的前 4 个元素 W[0]，W[1]，W[2]，W[3]是原始密钥，用于加密运算中的初始密钥加；后面 40 个元素分为 10 组，每组 4 个 32 位比特字（128 比特），分别用于 10 轮加密运算中的轮密钥加，如下图所示：



AES 的整体结构如下图所示，其中的 W[0,3]是指 W[0]、W[1]、W[2]和 W[3]串联组成的 128 位密钥。加密的第 1 轮到第 9 轮的轮函数一样，包括 4 个操作：**字节代换、行位移、列混合和轮密钥加**。最后一轮迭代不执行列混合。另外，在第一轮迭代之前，先将明文和原始密钥进行一次异或加密操作，即初始轮秘钥加。

图中也展示了 AES 解密过程，解密过程仍为 10 轮，每一轮的操作是加密操作的逆操作。由于 AES 的 4 个轮操作都是可逆的，因此，解密操作的一轮就是顺序执行逆行移位、逆字节代换、轮密钥加和逆列混合。同加密操作类似，最后一轮不执行逆列混合，在第 1 轮解密之前，要执行 1 次密钥加操作。


明文
轮密钥加
W[0,3]
字节代换
行移位
列混合
轮密钥加
第1轮
W[4,7]
字节代换
行移位
列混合
轮密钥加
第9轮
W[36,39]
字节代换
行移位
轮密钥加
第10轮
W[40,43]
密文
明文
轮密钥加
逆字节代换
逆行移位
第10轮
逆列混合
轮密钥加
逆字节代换
逆行移位
第9轮
逆列混合
轮密钥加
逆字节代换
逆行移位
第1轮
轮密钥加
密文

AES 的整体结构

2. AES 的轮加密

下面分别介绍 AES 中一轮的 4 个操作阶段，这 4 分操作阶段使输入位得到充分的混淆。

① 字节代换

AES 的字节代换其实就是一个简单的**查表操作**。AES 定义了一个 S 盒和一个逆 S 盒。S 盒用于加密，逆 S 盒用于解密。

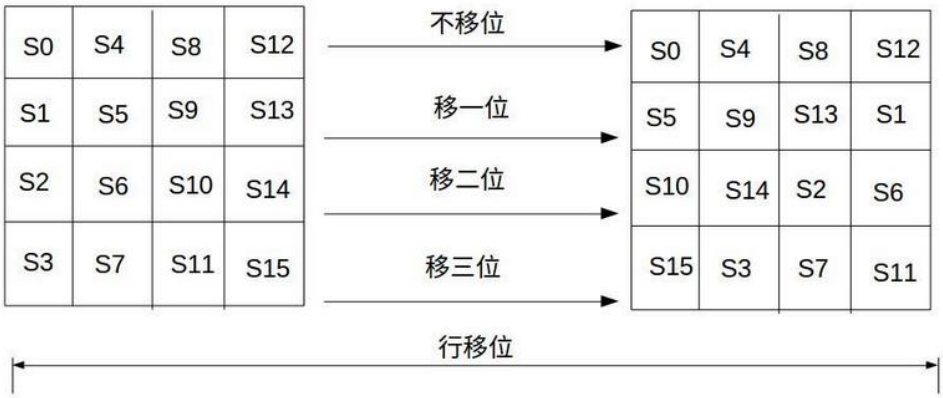
行/列	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0x63	0x7c	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76
1	0xca	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0
2	0xb7	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15
3	0x04	0xc7	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75
4	0x09	0x83	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84
5	0x53	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf
6	0xd0	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8
7	0x51	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2
8	0xcd	0x0c	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73
9	0x60	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb
A	0xe0	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79
B	0xe7	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08
C	0xba	0x78	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a
D	0x70	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e
E	0xe1	0xf8	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf
F	0x8c	0xa1	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16

AES 的 S 盒

状态矩阵中的元素按照下面的方式映射为一个新的字节：把该字节的高 4 位作为行值（16 进制），低 4 位作为列值，取出 S 盒或者逆 S 盒中对应的行的元素作为输出。

② 行移位

行移位是一个简单的左循环移位操作。当密钥长度为 128 比特时，状态矩阵的第 0 行左移 0 字节，第 1 行左移 1 字节，第 2 行左移 2 字节，第 3 行左移 3 字节，如图所示：



行移位的逆变换是将状态矩阵中的每一行执行相反的移位操作，例如 AES-128 中，状态矩阵的第 0 行右移 0 字节，第 1 行右移 1 字节，第 2 行右移 2 字节，第 3 行右移 3 字节。

③ 列混合

列混合变换是通过矩阵相乘来实现的，经行移位后的状态矩阵与固定的矩阵相乘，得到混淆后的状态矩阵，如下图的公式所示：

$$\begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

状态矩阵中的第 j 列(0 ≤j≤3)的列混合可以表示为下图所示：

$$s'_{0,j} = (2 * s_{0,j}) \oplus (3 * s_{1,j}) \oplus s_{2,j} \oplus s_{3,j}$$
$$s'_{1,j} = s_{0,j} \oplus (2 * s_{1,j}) \oplus (3 * s_{2,j}) \oplus s_{3,j}$$
$$s'_{2,j} = s_{0,j} \oplus s_{1,j} \oplus (2 * s_{2,j}) \oplus (3 * s_{3,j})$$
$$s'_{3,j} = (3 * s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 * s_{3,j})$$

其中，矩阵元素的乘法和加法都是定义在基于 GF(2^8)上的二元运算，并不是通常意义上的乘法和加法。这种二元运算的加法等价于两个字节的异或，乘法则复杂一点。对于一个 8 位的二进制数来说，使

用域上的乘法乘以$(00000010)_2$等价于左移 1 位(低位补 0)后，再根据情况同$(00011011)_2$进行异或运算，设 S1 = (a7 a6 a5 a4 a3 a2 a1 a0)$_2$，则 0x02 * S1 如下图所示：

$$(00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0) = \begin{cases} (a_6a_5a_4a_3a_2a_1a_00), a_7 = 0 \\ (a_6a_5a_4a_3a_2a_1a_00) \oplus (00011011), a_7 = 1 \end{cases}$$

也就是说，如果 a7 为 1，则进行异或运算，否则不进行。

类似地，乘以(00000100)可以拆分成两次乘以(00000010)的运算：

$$(00000100) * (a_7a_6a_5a_4a_3a_2a_1a_0) = (00000010) * (00000010) * (a_7a_6a_5a_4a_3a_2a_1a_0)$$

乘以$(0000\ 0011)_2$可以拆分成先分别乘以$(0000\ 0001)_2$和$(0000\ 0010)_2$，再将两个乘积异或：

$$\begin{aligned}(00000011) * (a_7a_6a_5a_4a_3a_2a_1a_0) &= [(00000010) \oplus (00000001)] * (a_7a_6a_5a_4a_3a_2a_1a_0) \\ &= [(00000010) \oplus (a_7a_6a_5a_4a_3a_2a_1a_0)] \oplus (a_7a_6a_5a_4a_3a_2a_1a_0)\end{aligned}$$

因此，我们只需要实现乘以 2 的函数，其他数值的乘法都可以通过组合来实现。

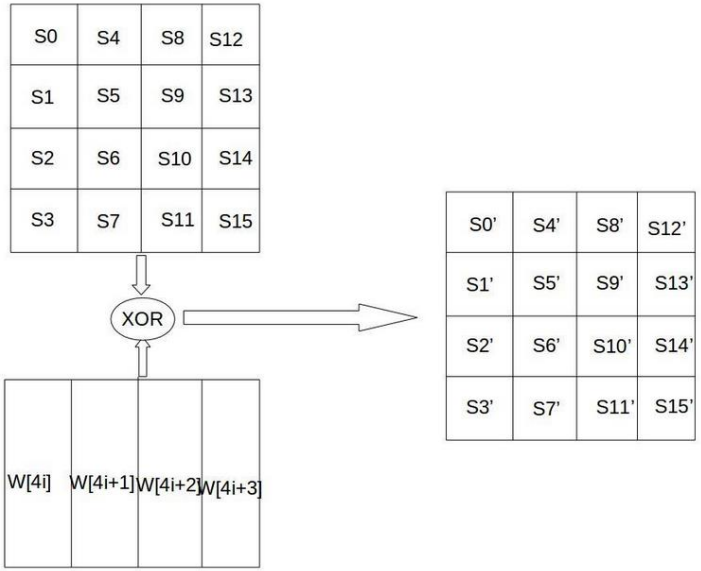
逆向列混合变换可由下图的矩阵乘法定义：

$$\begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

④ 轮密钥加

轮密钥加是将 128 位轮密钥字节 Ki 与状态矩阵中的数据进行**逐位异或**操作。其中，密钥 Ki 中每个字 W[4i]，W[4i+1]，W[4i+2]，W[4i+3]为 32 位比特字，包含 4 个字节。轮密钥加过程可以看成是字逐位异或的结果，也可以看成字节级别或者位级别的操作。也就是说，可以看成 S0、S1、S2、S3 组成的 32 位字与 W[4i]的异或运算。

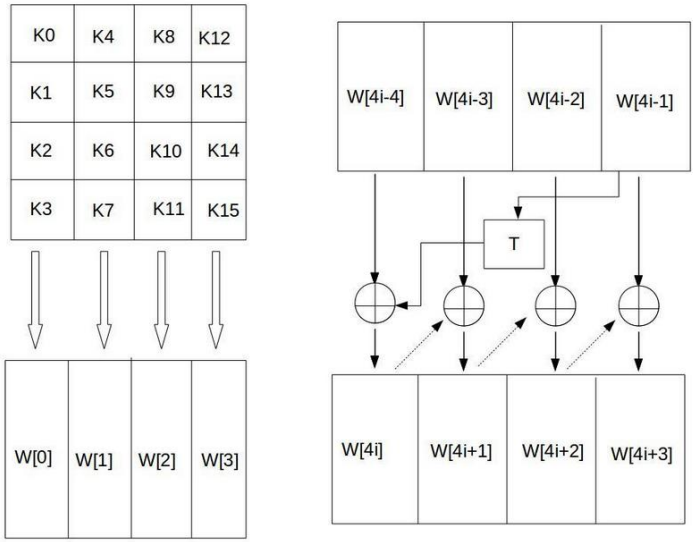
轮密钥加的逆运算同正向的轮密钥加运算完全一致，这是因为异或的逆操作是其自身。轮密钥加非常简单，但却能够影响 S 数组中的每一位。



轮秘钥加示意图

⑤ 秘钥扩展

AES 首先将初始密钥输入到一个 4*4 的状态矩阵中，如下图所示:



这个 4*4 矩阵的每一列的 4 个字节组成一个字，矩阵 4 列的 4 个字依次命名为 W[0]、W[1]、W[2]和 W[3]，它们构成一个以字为单位的数组 W。

接着，对 W 数组扩充 40 个新列，构成总共 44 列的扩展密钥数组。新列以如下的递归方式产生：

a. 如果 i 不是 4 的倍数，那么第 i 列由如下等式确定： W[i] = W[i-4] ⊕W[i-1]

b. 如果 i 是 4 的倍数，那么第 i 列由如下等式确定： W[i] = W[i-4] ⊕T(W[i-1])

其中，T 是一个函数，函数 T 由 3 部分组成：字循环、字节代换和轮常量异或，这 3 部分的作用分别如下：

a. 字循环：将 1 个字中的 4 个字节循环左移 1 个字节。即 [b0, b1, b2, b3] 变成 [b1, b2, b3, b0] 。

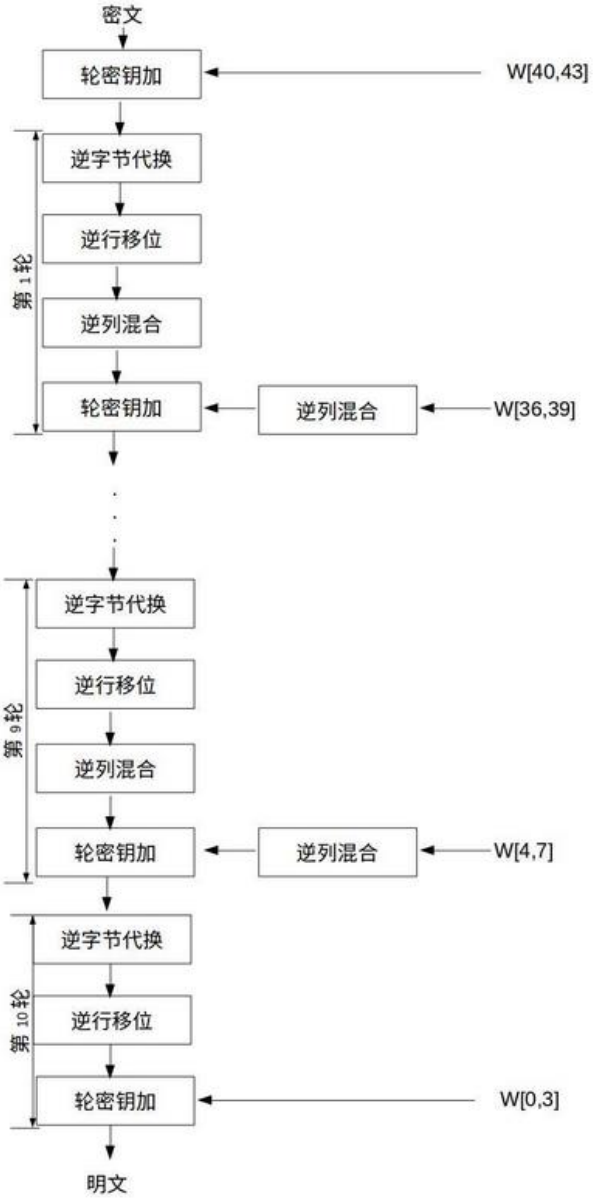
b. 字节代换：对字循环的结果使用 S 盒进行字节代换。

c. 轮常量异或：将前两步后的结果同轮常量 Rcon[j]进行异或，其中 j 表示轮数（j∈ [1,10], j∈Z）。轮常量 Rcon[j]是一个字，其值见下表。

j	**1**	**2**	**3**	**4**	**5**
Rcon[j]	01 00 00 00	02 00 00 00	04 00 00 00	08 00 00 00	10 00 00 00
j	**6**	**7**	**8**	**9**	**10**
Rcon[j]	20 00 00 00	40 00 00 00	80 00 00 00	1B 00 00 00	36 00 00 00

3. AES 解密

在前面的图中，有 AES 解密的流程图，可以对应那个流程图来进行解密。下面介绍的是另一种等价的解密模式，流程图如下图所示。这种等价的解密模式使得解密过程各个变换的使用顺序同加密过程的顺序一致，只是用逆变换取代原来的变换。



2020.02.16

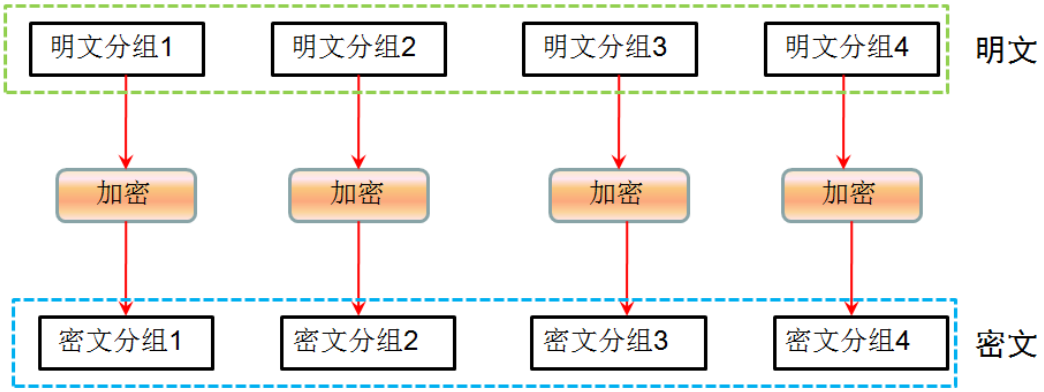
AES 另一种等价解密模式

4. AES 的模式

分组密码在加密时明文分组的长度是固定的，而实用中待加密消息的数据量是不定的，数据格式可能是多种多样的。为了能在各种应用场合安全地使用分组密码，通常对不同的使用目的运用不同的工作模式。

① 电码本模式（ECB，Electronics Codebook mode）（不应使用）

将整个明文分成若干段相同的小段，然后对每一小段进行加密；解密直接过程相反，前后明文、密文之间互不干扰。明文分组与密文分组是一一对应的关系，因此如果明文中存在多个相同的明文分组，则这些明文分组会转换为相同的密文分组。



ECB 模式的加密

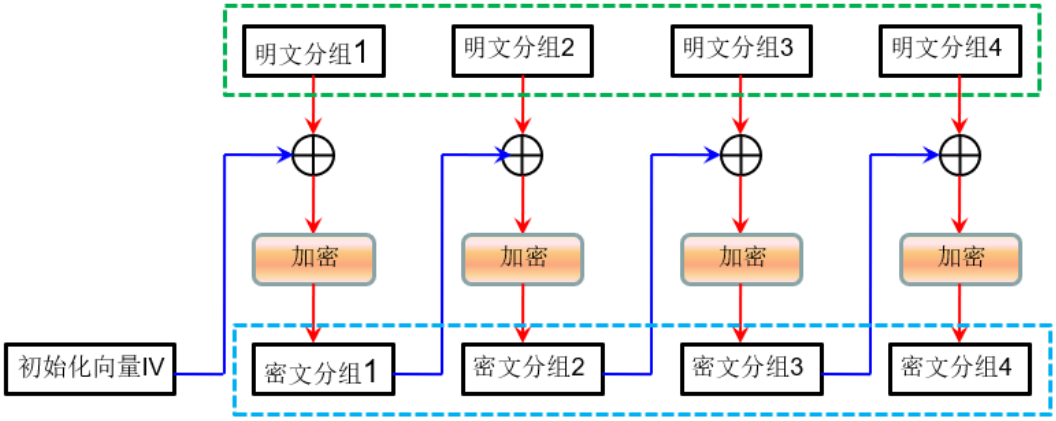
优点：（1）操作简单，易于实现；（2）分组独立，易于并行；（3）误差不会被传送。

缺点：（1）明文中的重复排列会反映在密文中，并可以以此为线索来破译密码；（2）掩盖不了明文**结构信息**，难以抵抗统计分析攻击；（3）通过删除、替换密文分组可以对明文进行操作；（4）对包含某些比特错误的密文解密时，对应的分组会出错；（5）不能抵御重放攻击。

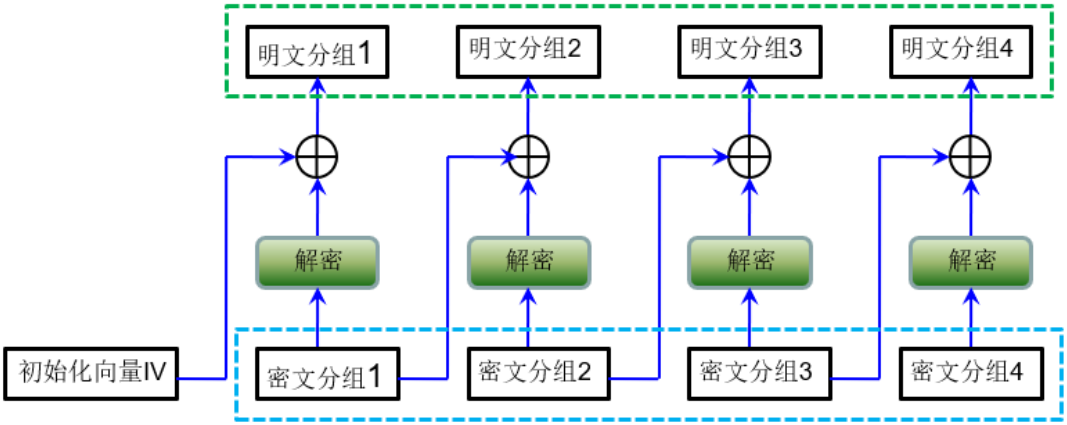
对 ECB 模式的攻击：攻击模式为主动攻击，与窃听者只能被动地进行窃听不同，主动攻击者则可以主动介入发送者和接收者之间的通信过程，进行阻碍通信或者篡改密文等活动。攻击者能够改变密文的顺序。当接收者对密文解密时，由于密文分组的顺序被改变了，因此相应的明文顺序也被改变了（比如对调收付款人）。也就是说,攻击者不需要破解密文，也不需要知道分组密码算法，只要知道哪个分组记录了什么样的数据（电文的格式）即可。

② **密码分组链模式（CBC，Cipher Block Chaining mode）**（推荐使用）

先将明文切分成若干小段，然后每一小段与初始块或者上一段的密文段进行异或运算后，再与密钥进行加密。由于异或是对称操作，解密的时候只需要与前一段密文再进行一次异或即可。



CBC 模式的加密



CBC 模式的解密

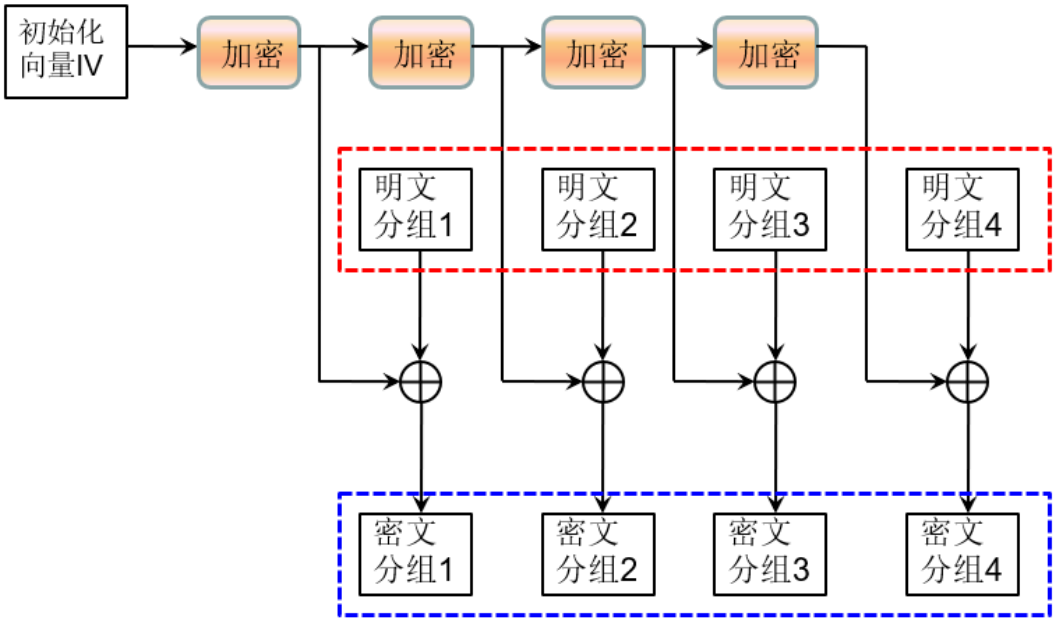
优点：（1）能掩盖明文结构信息，明文的重复排列不会反映在密文中（因为与前一个密文异或）；（2）不容易主动攻击（对调密文会导致明文混乱）；（3）解密可以并行计算；（4）能够解密任意密文分组；（5）安全性好于 ECB，适合传输长度**长的报文**，是 SSL 和 IPsec 的标准。

缺点：（1）加密不利于并行计算；（2）传递误差——前一个出错则后续全错；（3）第一个明文块需要与一个初始化向量 IV（Initialization Vector）进行抑或，初始化向量 IV 的选取比较复杂。

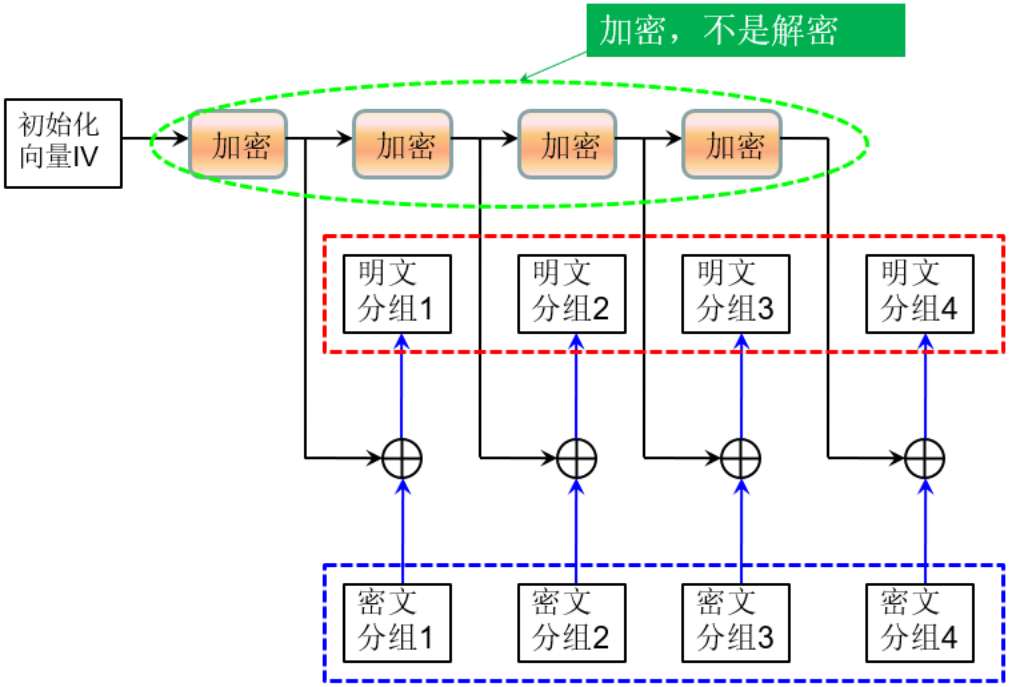
初始化 IV 的选取方式：固定 IV，计数器 IV，随机 IV（只能得到伪随机数，用的最多），瞬时 IV（难以得到瞬时值）。要注意每次加密都要更换初始化向量，如果每次加密是使用同一个初始化相量，当用同一密钥对同一明文加密时，所得到的明文是相同的。如果破译者隔一周内到两份相同的密文，无需破译密码，就可做出判断，这份密文和上周的密文一样，两份密文解密所得到的明文也是一样的。

对 CBC 模式的攻击：比特反转，如果攻击者能够对初始化向量中的任意比特进行反转（1 变 0，0 变 1），则明文分组中相应的比特也会反转。但是，攻击者却很难通过对密文分组的攻击，达到让明文分组中对应比特（期望的比特）反转的目的。因为如果改变了密文分组中的某一位，解密后明文分组中的多个比特会发生变化（来自每个分组的同一位）。

③ 输出反馈模式（OFB，Output Feedback mode）（推荐用 CTR 模式代替）



OFB 模式的加密



OFB 模式的解密

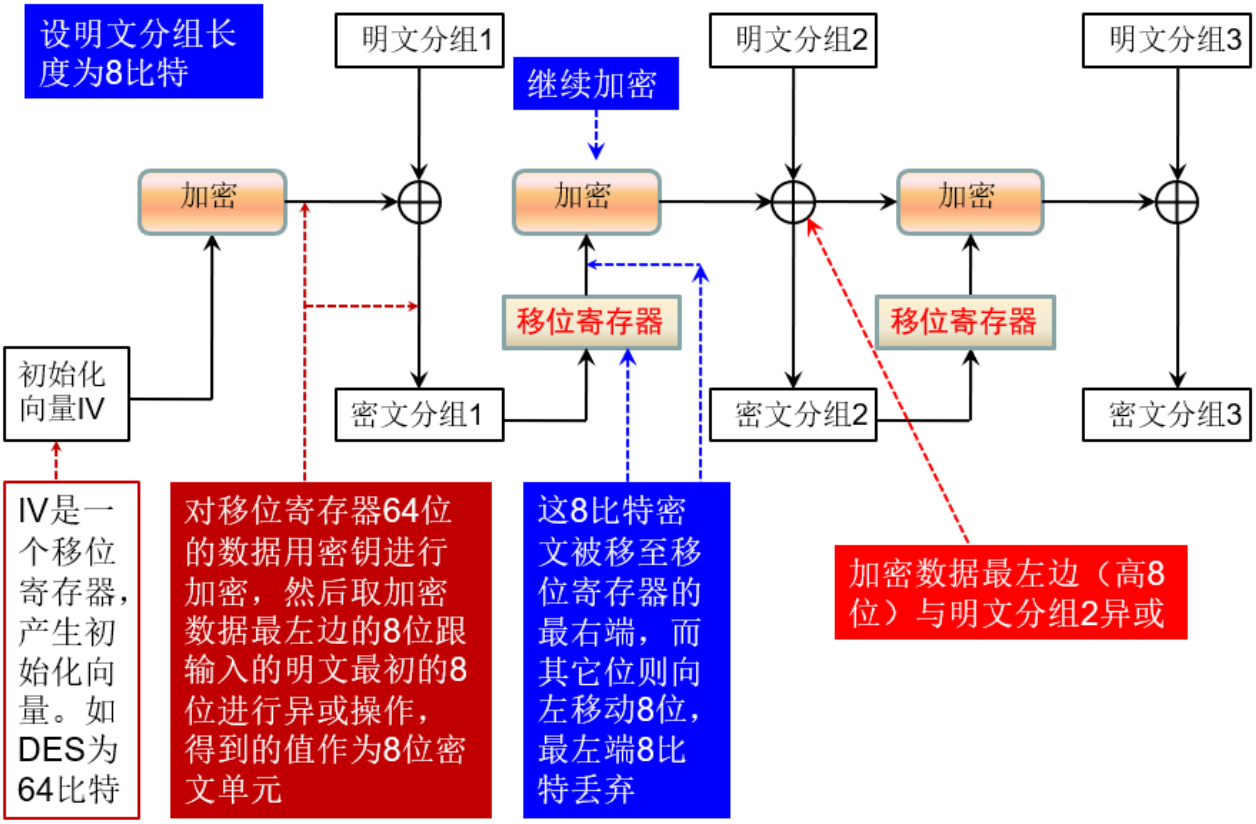
OFB 模式并不是通过 AES 对明文直接加密，而是通过将明文分组和初始化向量每一轮 AES 的输出进行 XOR 来产生密文分组。解密同样是利用了异或操作的可逆性。

优点：（1）隐藏了明文模式；（2）结合了分组加密和**流密码**（分组密码转化为流模式）；（3）可以及时加密传送小于分组的数据；（4）不需要填充；（5）可事先进行加密、解密的准备；（6）加密、解密使用相同的结构；（7）对包含某些比特错误的密文解密时，只有明文中相对应的比特出现错误。

缺点：（1）不利于并行计算；（2）需要生成秘钥流；（3）对明文的主动攻击是可能的，主动攻击者反转密码中的某些比特时，明文分组中相对应的比特也会反转。推荐使用 CTR 模式代替。（4）对 OFB 模式可以实施重放攻击（replay attack），解决这样的方法需要采用消息认证技术。

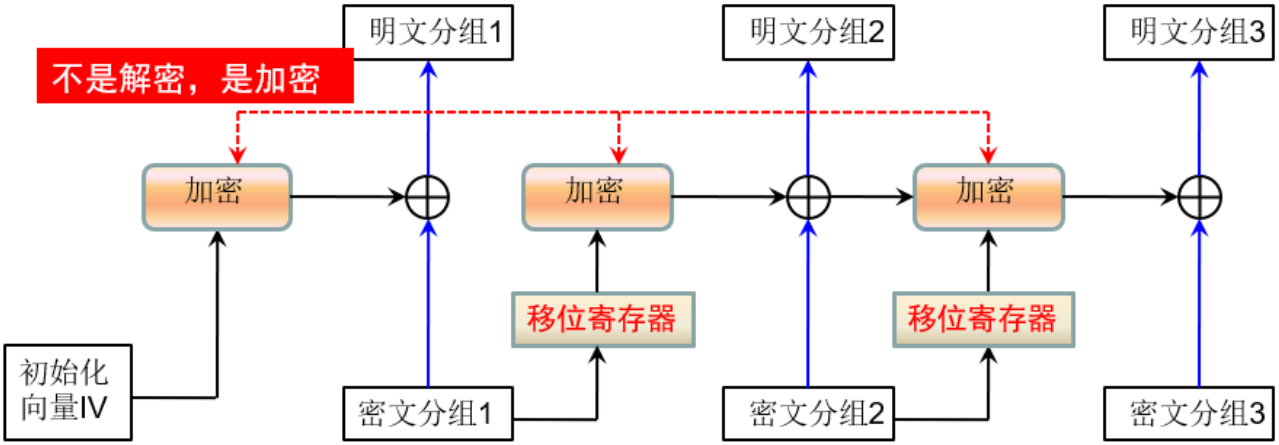
④ **密码反馈模式（CFB，Cipher Feedback mode）**（推荐用 CTR 模式代替）

把分组密码当作流密码使用，即密码反馈模式可将 DES 分组密码置换成流密码。流密码具有**密文和明文长度一致**、运行实时的性质，这样数据可以在比分组小得多的单元里进行加密。如果需要发送的每个字符长为 8 比特，就应使用 8 比特密钥来加密每个字符。如果长度超过 8 比特，则造成浪费。但是要注意，由于 CFB 模式中分组密码是以流密码方式使用，所以加密和解密操作完全相同，因此无法适用于公钥密码系统，只能适用于对称密钥密码系统。密码反馈模式也需要一个初始量，无须保密，但对每条消息必须有一个不同的初始量（防止同样的明文产生相同的密文）。



CFB 模式的加密

解密同样是利用了异或操作的可逆性：



CFB 模式的解密

特点：加密的明文必须按照一个字节或者一位进行处理，即将分组密码转换为流密码；假设它的输出是 s 位，s 位的大小可以是 1 位、8 位、 64 位或者其他大小，表示为 CFB-1, CFB-8, CFB-64 等

优点：密码反馈模式具有流密码的优点，也拥有 CBC 模式的优点。（1）可以处理任意长度的消息，能适应用户不同数据格式的需要；（2）不需要填充；（3）解密支持并行计算；（4）能够解密任意密文分组。

缺点：但是它也拥有 CBC 模式的缺点，即也会导致错误传播。（1）对信道错误较敏感，且会造成错误传播；（2）加密不能并行计算；（3）密码反馈模式也会降低数据加密速度，无论每次输出多少位，都需要事先用密钥 K 加密一次，再与相等的明文位异或，所以即使一次输出为 1 位，也要经过相同的过程。（4）对 CFB 模式可以实施重放攻击（replay attack），解决这样的方法需要采用消息认证技术。

⑤ 计数器模式（CTR，Counter mode）

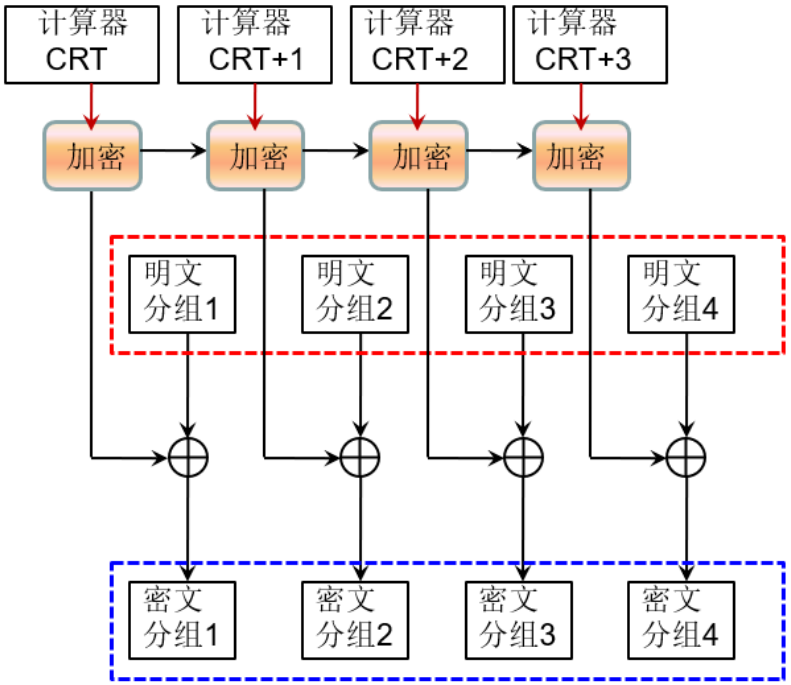
完全的流模式。将瞬时值与计数器连接起来，然后对此进行加密产生密钥流的一个密钥块，再进行 XOR 操作。CTR 模式中计数器生成方法为：每次加密时都会生成一个不同的值（nonce）来作为计数器的初始值。当分组长度为 128 比特时，计数器的初始值可能是像如下这样的形式：

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 00 01

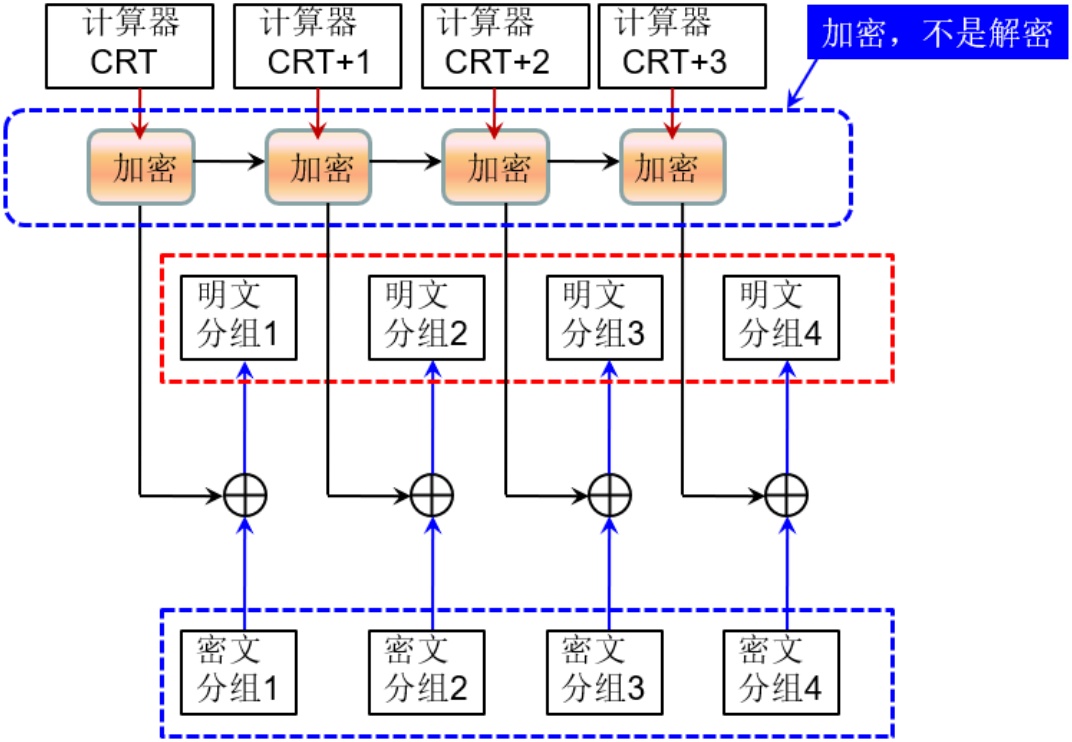
（nonce） 分组序号

前 8 个字节为 nonce，这个值在每次加密时都是不同的。后 8 个字节为分组序号，这个部分是逐次累加的。在加密过程中，计数器的值会产生如下变化：

66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 00 01 明文分组1的计数器值（初值）
66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 00 02 明文分组2的计数器值（初值）
66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 00 03 明文分组3的计数器值（初值）
66 1F 98 CD 37 A3 8B 4B 00 00 00 00 00 00 00 04 明文分组4的计数器值（初值）



CTR 模式的加密



CTR 模式的解密

优点：（1）不泄露明文；（2）仅需实现加密函数；无需填充；（3）加密、解密均可并行计算，速度快；（4）可事先进行加密、解密的准备；（5）对包含某些比特错误的密文解密时，只有明文中相对应的比特出现错误。（6）如果有好的瞬时值选择策略，采用 CTR，否则采用 CBC。如加密成绩单，可选用 CTR，因为学号唯一，可作为瞬时值。（7）CBC 需要填充，因为密文不一定是分组的倍数；CTR 不用填充，因为是对固定长的初始化向量加密。（8）CBC 需要实现加密和解密函数；CTR 实现简单，仅需实现加密函数。

缺点：（1）需要瞬时值 IV，难以保证 IV 的唯一性；（2）主动攻击者反转密码中的某些比特时，明文分组中相对应的比特也会反转。（3）使用重复瞬时值，CBC 会泄露初始明文块，CTR 会泄露所有信息。

七. Salsa20 算法

1. 算法设计背景

2004 年 ECRYPT 启动了 eSTREAM 流密码计划的研究项目，Salsa20 是最终胜出的 7 个算法之一。Salsa20 是由 Daniel J. Bernstein 提出的基于 hash 函数设计的流密码算法，其核心部分是一个基于 32 比特加、比特异或以及旋转操作的 512 比特输入 512 比特输出的 hash 函数。

现有的对 Salsa20 攻击方法包括线性分析、差分分析、非随机性分析、相关密钥分析和滑动分析等，其中以滑动攻击尤其是偶数轮的滑动攻击方式最为有效。在软件仿真方面，算法可以在目前的 x86 处理器上实现 4 至 14 cycles/byte 的加密效率；且具备尚可的硬件性能。

Salsa20 没有申请专利，属于公开的应用算法，为了在多个公共领域的共同架构，Bernstein 对算法的实现优化做了进一步研究和改动。

2. 算法介绍

（1）运算符定义

word：word 是一个 32 比特的二进制数，即集合 $\{0,1,\cdots,2^{32}-1\}$ 中的元素之一。

（2）异或和移位计算

两个 word u，v 的异或用公式表示为：$u \oplus v = \sum_i 2^i \left(u_i + v_i - 2u_i v_i\right)$

设非零整数 c，$c \in \{0,1,2,3\ldots\}$，对一个 word u 的 c 比特左移位表示为 $u <<< c$，用公式表示为：$u <<< c = \sum_i 2^{i+c \bmod 32} u_i$

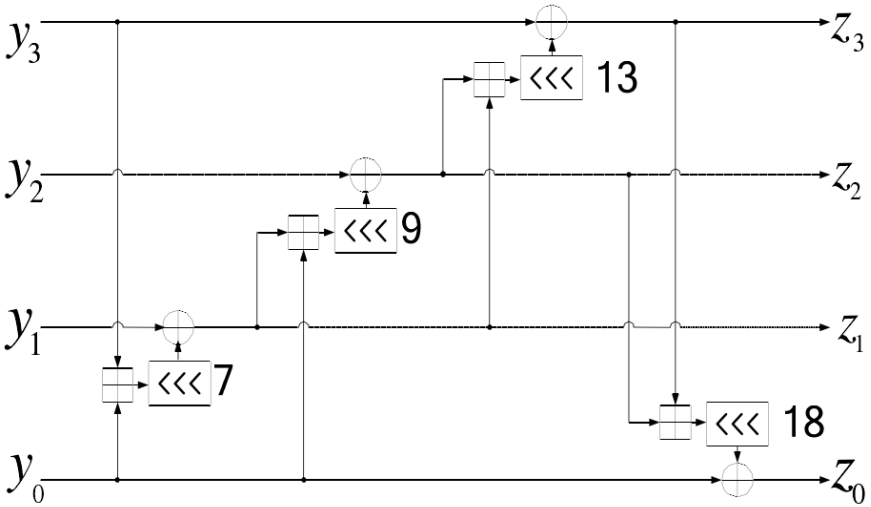
（3）quarter round 函数

设 $y = (y_0, y_1, y_2, y_3)$，输出 $z = quarterround(y) = (z_0, z_1, z_2, z_3)$，则

$$z_1 = y_1 \oplus \left((y_0 + y_3) <<< 7\right) \qquad z_2 = y_2 \oplus \left((z_1 + y_0) <<< 9\right)$$

$$z_3 = y_3 \oplus \left((z_2 + z_1) <<< 13\right) \qquad z_0 = y_0 \oplus \left((z_3 + z_2) <<< 18\right)$$

两者关系如图所示：



Salsa20 的 quarter round 函数

（4）row round 函数

设 $y=(y_0,y_1,y_2,y_3,\cdots,y_{15})$，输出 $z=quarterround(y)=(z_0,z_1,z_2,z_3,\cdots,z_{15})$，则：

$$(z_0,z_1,z_2,z_3)=quarterround(y_0,y_1,y_2,y_3) \qquad (z_5,z_6,z_7,z_4)=quarterround(y_5,y_6,y_7,y_4)$$

$$(z_{10},z_{11},z_8,z_9)=quarterround(y_{10},y_{11},y_8,y_9) \qquad (z_{15},z_{12},z_{13},z_{14})=quarterround(y_{15},y_{12},y_{13},y_{14})$$

若将输入 $(y_0,y_1,y_2,y_3,\cdots,y_{15})$ 看作是一个方阵，则 row round 函数将方阵的每一行重新排列后作为 quarter round 函数的输入，从而并行地修改了所有行。

$$\begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$$

（5）column round 函数

设 $x=(x_0,x_1,x_2,x_3,\cdots,x_{15})$，输出 $y=quarterround(x)=(y_0,y_1,y_2,y_3,\cdots,y_{15})$，则:

$$(y_0,y_4,y_8,y_{12})=quarterround(x_0,x_4,x_8,x_{12}) \qquad (y_5,y_9,y_{13},y_1)=quarterround(x_5,x_9,x_{13},x_1)$$

$$(y_{10},y_{14},y_2,y_6)=quarterround(x_{10},x_{14},x_2,x_6) \qquad (y_{15},y_3,y_7,y_{11})=quarterround(x_{15},x_3,x_7,x_{11})$$

若将输入 $(x_0,x_1,x_2,x_3,\cdots,x_{15})$ 看作是一个方阵，则 column round 函数将方阵的每一列重新排列后作为 quarter round 函数的输入，从而并行地修改了所有列。

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}$$

（6）double round 函数

double round 变换由一个列变换（column round）与一个行变换（row round）组成:

$$doubleround(X)=rowround(columnround(X))$$

（7）little endian 函数

设 $b=(b_0,b_1,b_2,b_3)$，则 $littleendian(b)=b_0+2^8b_1+2^{16}b_2+2^{24}b_3$，little-endian 函数是将 8 比特序列转变为 32 比特的可逆函数。

（8）Salsa20 的加密函数

Salsa20 算法的核心是一个 hash 函数，输入和输出均为 512 比特，算法的初始状态是由 16 个 word 组成的 4×4 矩阵，其中包括 8 个密钥k_0，k_1，……，k_7，4 个常数σ_1、σ_2、σ_3、σ_4，2 个初始 IV（nonce）n_0、n_1，2 个时间标号（block counter）c_0、c_1，假设用 X 表示初始状态，则 X 为:

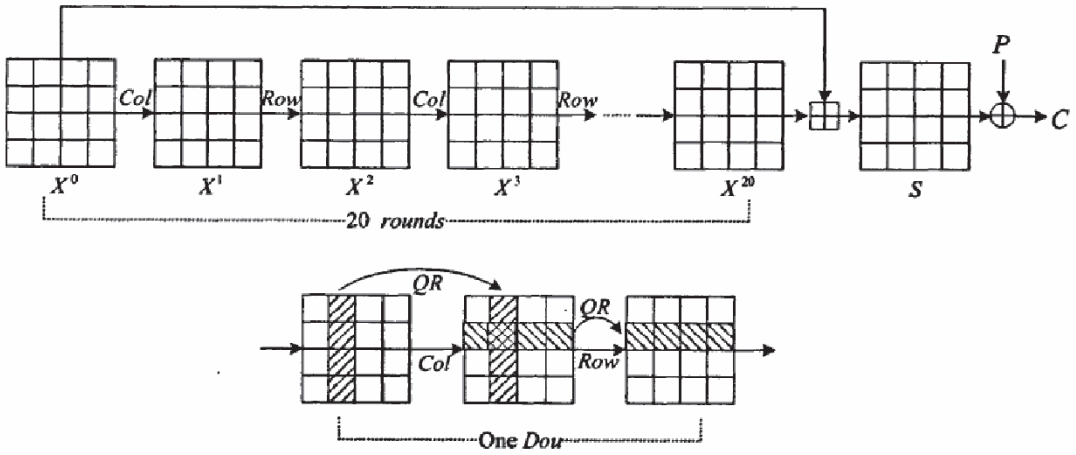
$$X=\begin{pmatrix}\sigma_0 & k_0 & k_1 & k_2\\ k_3 & \sigma_1 & n_0 & n_1\\ c_0 & c_1 & \sigma_2 & k_4\\ k_5 & k_6 & k_7 & \sigma_3\end{pmatrix}$$

密钥有两种类型，分别为 128 比特与 256 比特。若长度为 128 比特，则后四个密钥依次重发前四个密钥的值。从安全性角度考虑，使用 256 比特密钥更加安全。下面针对 256 比特的密钥长度版本对算法进行说明。

Salsa20 首先对 8-word 的密钥进行扩展，即将上述初始状态 X 经过 10 轮 double round 变换后的输出与 X 进行模 232 加。可以用公式简单的表示为：

$$Salsa20(X)=X+doubleround^{10}(X)$$

$Salsa20(X)$ 是 Salsa20 算法的**密钥流**，它异或一个 16-word 的明文分组就得出对该明文分组加密后的密文。设 P 是一个明文，C 是相应的密文，加密函数为 $C=P\oplus S$，解密函数为 $C=P\oplus S$。具体加密过程如下图所示。



Salsa20 的加密原理图

3. 安全性分析

随着 Salsa20 成为 eSTREAM 工程的最终胜选算法之一，关于它的各项密码学性质以及如何对 Salsa20 进行攻击逐渐成为广大学者研究的热点。目前对 Salsa20 的分析已经取得了一些结果，Paul Crowley 给出了对简化至 5 轮的 Salsa20 的一种截断差分攻击。Simon Fischer 等分析了 Salsa20 的密钥和初始 IV 的在载入时存在的问题。他们论证了对 6 轮 Salsa20 的攻击可以进行密钥恢复，同时他们观察 7 轮后仍然存在非随机性。随后 Yukiyasu Tsunoo 公开表示在 4 轮 Salsa20 后的输出差分概率有一个重要的偏差。利用这个偏差，他们进一步展示攻击密钥长度为 256 比特的版本 Salsa20 简化至 8 轮的可能性，该攻击的计算复杂度较穷举攻击略低。

此外，作为针对 Salsa20 最有效的攻击方法之一，滑动攻击能大幅降低算法破译的时间复杂度。David Wagner 和 Alex Biryukov 于 1999 年提出了滑动攻击，滑动攻击使密码体制的轮数变得无关紧要，通过分析密钥编排（key schedule，指从种子密钥得到轮密钥的过程）并挖掘其弱点来攻破密码。它适用于具有如下性质的密码算法：

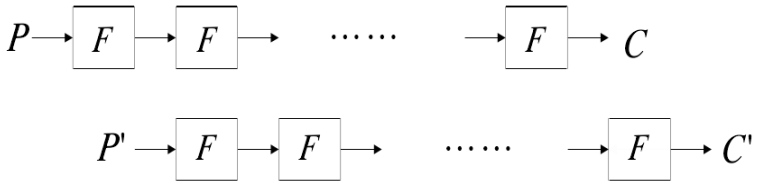
迭代过程具有一定的自相似性，且在很多情况下与迭代轮函数的具体性质和执行轮数无关。简单地说，就是任何可以被分解为若干次轮函数（假设该轮函数为 F）的算法都可以尝试此攻击（这个 F 可能包含多于原算法的一个轮函数）。由于密钥是直接包含在内部状态中的，通过滑动攻击，攻击者将能够恢复算法运行的所有内部状态，也同时能对 Salsa20 施行相关密钥恢复攻击。

首先引入一些符号。假设算法输入为 n 比特，密钥为 K_1，……，K_m。

滑动攻击首先将加密程序划分为完全相同的轮函数 F，F 可能由原算法中一轮以上的函数组成。例如，如果一个加密算法使用轮换的密钥体制，每一轮切换使用 K_1 和 K_2 这两个密钥，则 F 应该由两轮组成，每一个 K_i 在 F 中至少出现一次。

下一步是收集$2^{n/2}$个明密文对。根据不同加密算法各自的特点，可能不需要这么多个，但是根据生日悖论，我们期望总数不超过$2^{n/2}$。用(P, C) 表示这些明密文对，并利用它们来寻找滑动对(P', C')。一个滑动对(slid pair)是指满足如下性质的两个明密文对：$F(P_i) = P_i'$，$F(C_i) = C_i'$，且对应的密钥也相同。一旦我们找到了这样的滑动对，就可以利用已知明文攻击来攻破密码体制。

滑动攻击的基本思想是将一个加密过程相对于另一个加密过程“滑动”一轮。如下图所示：



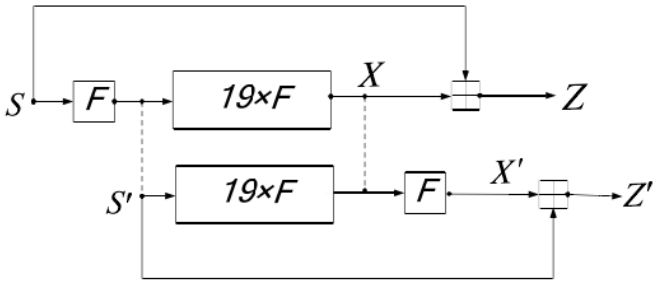
滑动攻击原理图

寻找滑动对的具体过程依加密体制而定，每一个不尽相同，但基本方法是一致的。所依据的事实是：从 F 的一次迭代(iteration)中分离出密钥是相对容易的。随意挑选明密文对(P, C)和(P', C')，如果$F(P_i) = P_i'$，$F(C_i) = C_i'$且对应的密钥也相同，我们就找到了一个滑动对，否则检测下一个明密文对。在$2^{n/2}$个明密文对中总可以找到一个滑动对，根据加密体制的结构，还有一小部分的误判断 (false-positives)。这些误判可以通过将密钥用在不同的明密文对并检测加密结果是否正确来排除。对一个好的密码体制来说，错误的密钥能正确加密两个及其以上明文消息的概率是非常低的。有时加密体制的结构能大大降低所需的明密文对数目，因此也降低了很多工作量。最明显的一个例子是使用一个密钥的 Feistel 结构。原因如下，给定$P=(L_0, R_0)$,必须寻找$P'=(R_0, L_0 \oplus F(R_0, K))$。这将可能的成对消息从$2^n$降到$2^{n/2}$，所以我们只需收集至多$2^{n/4}$个明密文对来找到滑动对。

在 Salsa20 中，double round 结构可以重写为一个 column round 紧接着一个矩阵转置，以及另一个 column round 和第二个矩阵转置。我们将 F 定义为由一个 column round 紧接着一个矩阵转置组成的函数。这样 10 轮的 double round 就转化成为运行了 20 轮的 F。如果我们有两个三元组$(key1, nonce1, counter1)$和$(key2, nonce2, counter2)$，那么

$$F[\text{初始状态}1(key1, nonce1, counter1)]$$
$$=\text{初始状态}2(key2, nonce2, counter2)$$

这个性质在轮运算的每个对应点都成立。但是 Salsa20 最后的前馈操作破坏了这个性质。我们将初始状态 1 和初始状态 2 称为一个滑动对，它们的关系如下图所示：



Salsa20 滑动对

在初始状态中，4 个 word σ_1、σ_2、σ_3、σ_4 是常量，剩下的 12 个 word 可以任意选取，因此总共有 2^{384} 种可能的初始状态。如果期望某个初始状态输入 F 后输出初始状态 2，可以得到四个 word 操作等式。这意味着我们可以为初始状态 1 任选 8 个 word，剩下的 4 个 word 由等式及初始状态 2 的 word 决定。这将得到 2^{256} 个可能的滑动对。

对于密钥长度为 128 比特的 Salsa20，不存在这样的滑动对。原因是初始状态 1 中少了随意选取 4 个 word 的自由权，而初始状态 2 多了 4 个 word 级方程式。通过 F 可以得到两个方程：$S' = \mathrm{F}(S)$，$X' = \mathrm{F}(X)$。我们将这些矩阵内的 word 表示为：

$$S = \begin{pmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & n_0 & n_1 \\ c_0 & c_1 & \sigma_2 & k_4 \\ k_5 & k_6 & k_7 & \sigma_3 \end{pmatrix} \qquad S' = \begin{pmatrix} \sigma_0 & k'_0 & k'_1 & k'_2 \\ k'_3 & \sigma_1 & n'_0 & n'_1 \\ c'_0 & c'_1 & \sigma_2 & k'_4 \\ k'_5 & k'_6 & k'_7 & \sigma_3 \end{pmatrix}$$

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \qquad X' = \begin{pmatrix} x'_0 & x'_1 & x'_2 & x'_3 \\ x'_4 & x'_5 & x'_6 & x'_7 \\ x'_8 & x'_9 & x'_{10} & x'_{11} \\ x'_{12} & x'_{13} & x'_{14} & x'_{15} \end{pmatrix}$$

$$Z=\begin{pmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{pmatrix} \qquad Z'=\begin{pmatrix} z'_0 & z'_1 & z'_2 & z'_3 \\ z'_4 & z'_5 & z'_6 & z'_7 \\ z'_8 & z'_9 & z'_{10} & z'_{11} \\ z'_{12} & z'_{13} & z'_{14} & z'_{15} \end{pmatrix}$$

注意到(S, S')对角线上的 word 相同，因此 $k'_1+k'_2=0$，$k'_3+n'_1=0$，$c'_1+c'_0=0$，$k'_7+k'_6=0$。

由于F的弱扩散性，攻击者对初始状态S'的每一列都能写出独立的方程组。

将关系式$S'=\mathrm{F}(S)$中的 quarter round 运算、$X'=\mathrm{F}(X)$中的 quarter round 运算及 Salsa20 最后的反馈操作联立，我们可以得到一个由 16 个方程组成的方程组。这个方程组可以简化为 4 个方程。要想解出第一个方程，必须猜测两个变量。剩下的三个方程中，总有两个变量是知道的：要么是由猜测得来的S，要么是计算得来的S'。因此不需要再猜测任何变量就可以解出来。

从而针对某一列的方程组，尽管所有的变量都是未知的，可以通过仅猜测其中两个变量解出来。通过 4 次复杂度为$O(2^{64})$的猜测，攻击者就可以完全恢复 512 比特的秘密初始状态S。这表明我们可以很容易地将 Salsa20（不包括对角线常量）和一个随机函数区分开来，因为同样的任务，后者需要$O(2^{511})$才能解决。由于我们能够恢复所有的内部状态，这种攻击也能对 Salsa20 施行相关密钥恢复攻击，因为密钥是直接包含在内部状态中的。

初始状态已知 word	Salsa20 的滑动攻击	随机猜测
一无所知	$O(2^{66})$	$O(2^{511})$
对角线	$O(2^{34})$	$O(2^{255})$
对角线，n'，c'	$O(2^{33})$	$O(2^{159})$
对角线，n，c	$O(1)$	$O(2^{127})$
对角线，n，c，n'，c'	$O(1)$	$O(2^{63})$

滑动攻击与随机预言机的时间复杂度比较

4. 仿真情况

```
Weighted average (Simple Imix):
Encryption speed (cycles/byte): 9.34
Encryption speed (Mbps): 2392.86
Overhead: 22.6%
```

八. chacha20-poly1305 加密

ChaCha20-Poly1305 是 Google 所采用的一种新式加密算法，性能强大，在 CPU 为精简指令集的 ARM 平台上尤为显著（ARM v8 前效果较明显），在同等配置的手机中表现是 AES 的 4 倍（ARM v8 之后加入了 AES 指令，所以在这些平台上的设备，AES 方式反而比 chacha20-Poly1305 方式更快，性能更好），可减少加密解密所产生的数据量进而可以改善用户体验，减少等待时间，节省电池寿命等。谷歌选择了 **ChaCha20 流密码**和伯恩斯坦的 **Poly1305 消息认证码**（MAC）取代过去一直在互联网安全领域使用的基于 OpenSSL 的 RC4 密码，最初是为了保证能够在 Android 手机上的 Chrome 浏览器和谷歌网站间的 HTTPS（TLS/SSL）通讯。在谷歌采用 TLS（安全传输层协议）不久后，ChaCha20 和 Poly1305 均用在了 OpenSSH 中的 ChaCha20-Poly1305 新算法中，这使得 OpenSSH 可能避免因编译时间对 OpenSSL 产生依赖。ChaCha20 还用于 OpenBSD（一种多平台类 UNIX 操作系统）中的 RC4 随机数生成器，在 DragonFlyBSD 中作为内核的伪随机数产生器（Cryptographically Secure Pseudo-Random Number Generator，简称 CSPRNG）的子程序。

chacha20-ietf-poly1305 也是一种新式加密算法，比 chacha20-poly1305 更快。其中 ietf 表示互联网工程任务组(Internet Engineering Task Force ,*IETF*)。

1. ChaCha20

ChaCha 系列流密码，作为 Salsa 密码的改良版，具有更强的抵抗密码分析攻击的特性，“20”表示该算法有 20 轮的加密计算。

由于是流密码，故以字节为单位进行加密，安全性的关键体现在密钥流生成的过程，即所依赖的伪随机数生成器（PRNG）的强度，加密过程即是将密钥流与明文逐字节异或得到密文；反之，解密是将密文再与密钥流做一次异或运算得到明文。步骤与 Salsa20 相似。

2. Poly1305 消息认证码

消息认证码（带密钥的 Hash 函数）:密码学中，通信实体双方使用的一种验证机制，保证消息数据完整性的一种工具。构造方法由 M.Bellare 提出，安全性依赖于 Hash 函数，故也称带密钥的 Hash 函数。消息认证码是基于密钥和消息摘要所获得的一个值，可用于数据源发认证和完整性校验。

Poly1305 是 Daniel.J.Bernstein 创建的消息认证码，用于检测消息的完整性和验证消息的真实性，现常在网络安全协议（SSL/TLS）中与 Salsa20 或 ChaCha20 流密码结合使用。Poly1305 消息认证码的输入为 32 字节（256bit）的密钥和任意长度的消息比特流，经过一系列计算生成 16 字节（128bit）的摘要。

3. 性能

对于 AES 这样的块加密算法来说，在某些硬件上运行的非常快，比如现在的服务器和台式机都有 AES-NI 加速指令。而如果没有加速指令，纯粹通过软件运行，性能是很低的。

而流密码算法刚好反过来，软件实现性能更高，大部分移动设备（比如手机）由于没有 AES-NI 支持，运行 AES 加密是很缓慢的。运行 RC4 这样的流加密算法速度相对较快，可惜的是 RC4 早已经被证明不安全了。

这个时候 ChaCha20-Poly1305 流密码算法来了，除了安全性外，它在移动设备上运行的性能较高。大部分观点认为，在移动设备上（没有 AES-NI 指令），ChaCha20-Poly1305 算法的性能是 AES-128-GCM 的三倍；当然在台式机和服务器上，AES-128-GCM 性能比 ChaCha20-Poly1305 更高。

在 HTTPS 协议（TLS 协议中）中，密码库（OpenSSL、NSS）在进行数据加密的时候，涉及到两种算法，分别是对称加密算法和 MAC 算法（Message Authentication Codes，消息验证码），为保证机密性和完整性，这两种算法必须同时存在。对于密码库（比如 OpenSSL）来说，如果由它结合处理对称加密算法和 MAC 算法，有的时候会出现安全问题。这时候就出现了 AEAD 加密模式（AE：Authenticated encryption ； AD：Associated-Data），它在内部自行处理加密和 MAC 运算，无须密码库（比如 OpenSSL）处理，安全性更高，而 ChaCha20-Poly1305 也采用 AEAD 加密模式。

单独从加密算法的角度来看，分为块加密算法和流密码加密算法，RC4 是一种流密码加密算法，但由于安全问题，已经基本不在 HTTPS 中使用了，块加密算法比较流行的就是 AES 算法。而 ChaCha20-Poly1305 是一种流加密算法。

下图 GCM 表示该对称加密采用 Counter 模式，并带有 GMAC 消息认证码。

<table>
<tr><th>加密模式</th><th>加密性</th><th>完整性</th><th>密码库处理逻辑</th></tr>
<tr><td rowspan="2">传统加密模式</td><td>RC4(流加密算法)</td><td>MAC</td><td>单独处理</td></tr>
<tr><td>AES-CBC（块加密算法）</td><td>MAC</td><td>单独处理</td></tr>
<tr><td rowspan="2">AEAD</td><td colspan="2">AES-GCM（块加密算法）</td><td rowspan="2">无须单独处理</td></tr>
<tr><td colspan="2">ChaCha20-Poly1305(流加密算法)</td></tr>
</table>

九. RSA 加密（公钥加密，私钥解密）

RSA 加密为**非对称密钥加密**，加密和解密用的密钥是不同的，这种加密方式是用数学上的难解问题构造的，通常加密解密的速度比较慢，适合偶尔发送数据的场合（如传输 AES 秘钥）。优点是密钥传输方便。它是目前最重要的加密算法，计算机通信安全的基石，保证了加密数据不会被破解。

简单来说，RSA 加密过程如下：甲乙双方通讯，乙方（接收方）生成公钥和私钥，甲方获取公钥，并对信息加密（公钥是公开的，任何人都可以获取），甲方用公钥对信息进行加密，此时加密后的信息只有私钥才可以破解，所以只要私钥不泄漏，就能保证信息的安全性。

1. RSA 加密过程

RSA 的加密过程可以使用一个通式来表达：

$$\text{密文}=\text{明文}^{E} \bmod N$$

也就是说 RSA 加密是对明文的 E 次方后除以 N 后求余数的过程。从通式可知，只要知道 E 和 N 任何人都可以进行 RSA 加密了，所以说 E、N 是 RSA 加密的密钥，也就是说 E 和 N 的组合就是公钥，我们用(E,N)来表示公钥：

$$\text{公钥} = (E, N)$$

不过 E 和 N 不并不是随便什么数都可以的，它们都是经过严格的数学计算得出的，关于 E 和 N 拥有什么样的要求及其特性后面会讲到。E 是加密（Encryption）的首字母，N 是数字（Number）的首字母。

2. RAS 解密过程

RSA 的解密同样可以使用一个通式来表达：

$$\text{明文}=\text{密文}^{D} \bmod N$$

也就是说对密文进行 D 次方后除以 N 的余数就是明文，这就是 RSA 解密过程。知道 D 和 N 就能进行解密密文了，所以 D 和 N 的组合就是私钥：

$$\text{私钥} = (D, N)$$

从上述可以看出 RSA 的加密方式和解密方式是相同的，加密是求“E 次方的 mod N”；解密是求“D 次方的 mod N”。此处 D 是解密（Decryption）的首字母；N 是数字（Number）的首字母。

3. 生成密钥对

既然公钥是（E，N），私钥是（D，N），所以密钥对即为（E，D，N），但密钥对是怎样生成的？步骤如下：求 N；求 L（L 为中间过程的中间数）；求 E；求 D。

① 求 N：

准备两个互质数 p，q。这两个数不能太小，太小则会容易破解，将 p 乘以 q 就是 N。如果互质数 p 和 q 足够大，那么根据目前的计算机技术和其他工具，至今也没能**从 N 分解出 p 和 q**。换句话说，只要密钥长度 N 足够大（一般 1024 足矣），基本上不可能从公钥信息推出私钥信息。

$$N = p * q$$

② 求 L：

L 是 p－1 和 q－1 的最小公倍数，可用如下表达式表示

$$L = \text{lcm}（p－1，q－1）$$

③ 求 E：

E 必须满足两个条件：E 是一个比 1 大比 L 小的数，且 E 和 L 的最大公约数为 1；用 gcd (X,Y)来表示 X，Y 的最大公约数:

$$1 < E < L \quad 且 \quad \text{gcd}（E，L）= 1$$

之所以需要 E 和 L 的最大公约数为 1，是为了保证一定存在解密时需要使用的数 D。现在我们已经求出了 E 和 N 也就是说我们已经生成了密钥对中的公钥了。

④ 求 D：

数 D 是由数 E 计算出来的，数 D 必须保证足够大。D、E 和 L 之间必须满足以下关系:

$$1 < D < L \quad 且 \quad (E * D) \bmod L = 1$$

只要 D 满足上述条件，则通过 E 和 N 进行加密的密文就可以用 D 和 N 进行解密。条件“E * D mod L = 1”是为了保证密文解密后的数据就是明文。现在私钥也已经生成了，密钥对也就自然生成了。

求 N	N = p * q ； p，q 为质数
求 L	L = lcm（p－1，q－1） ； L 为 p－1、q－1 的最小公倍数
求 E	1 < E < L，gcd（E，L）=1；E，L 最大公约数为 1（E 和 L 互质）
求 D	1 < D < L，(E * D) mod L = 1

4. 相关数学知识

① 互质关系

如果两个正整数，除了 1 之外没有其他公因子，我们称这两个数是互质关系。比如 15 和 32，说明不是质数也可以构成互质关系。关于互质关系，有以下结论：

- 任意两个质数构成互质关系，比如 13 和 61；
- 一个数是质数，另一个数只要不是前者的倍数，两者就构成互质关系，比如 3 和 10；
- 如果两个数中，较大的那个数是质数，则两者构成互质关系，比如 97 和 57；
- 1 和任意一个自然数都是互质关系；
- p 是大于 1 的整数，则 p 和 p-1 构成互质关系，比如 57 和 56；
- p 是大于 1 的奇数，则 p 和 p-2 构成互质关系，比如 17 和 15。

② 欧拉函数

任意给定正整数 n，计算在小于等于 n 的正整数之中，有多少个与 n 构成互质关系。计算这个值的方法叫做欧拉函数。以 φ(n)表示。如 φ(8) = 4，应为在 1 到 8 之中，与 8 形成互质关系的有 1，3，5，7。

φ(n)的计算方法并不复杂，我们就分情况分析。

第一种情况：如果 n=1，则 φ(1)=1。应为 1 与任何(包括自己)都构成互质关系。

第二种情况：如果 n 是质数，则 φ(n)=n-1。应为质数与每个小于他的数都构成互质关系。

第三种情况：如果 n 是质数的某一个次方，即 n = p^k (p 为质数，k≥1，k∈Z)，则

$$\varphi(p^k) = p^k - p^{k-1} = p^k * \left(1 - \frac{1}{p}\right)$$

如 φ(8) = φ(2^3) = $2^3 - 2^2 = 4$。这是应为只有当一个数不包含质数 p 时，才能与 n 互质。而包含质数 p 的数一共有p^{k-1}个，即 1*p、2*p、……、$p^{k-1} * p$。

第四种情况：n 可以分解成两个互质的整数之积。若 n = $p_1 * p_2$，则 φ(n) = φ(p_1p_2) = φ(p_1)φ(p_2)，即积的欧拉函数等于各个因子的欧拉函数之积。如：φ(56)=φ(7*8)=φ(7)*φ(8)=6*4=24。

第五种情况：因为任意大于 1 的整数，都可以写成一系列质数的积。

$$n = p_1^{k_1} * p_2^{k_2} * \dots\dots * p_r^{k_r}$$

根据上一种情况的结论：

$$\varphi(n) = \varphi\left(p_1^{k_1}\right) * \varphi\left(p_2^{k_2}\right) * \dots\dots * \varphi\left(p_r^{k_r}\right)$$

再根据第三种情况的结论：

$$\varphi(n) = p_1^{k_1} * p_2^{k_2} * \dots\dots * p_r^{k_r} * \left(1 - \frac{1}{p_1}\right) * \left(1 - \frac{1}{p_2}\right) * \dots\dots * \left(1 - \frac{1}{p_r}\right)$$

等价于

$$\varphi(n) = n * \left(1 - \frac{1}{p_1}\right) * \left(1 - \frac{1}{p_2}\right) * \dots\dots * \left(1 - \frac{1}{p_r}\right)$$

比如：φ (1323) = φ (3^2*7^2) = 1323 (1-1/3) (1-1/7) = 756

③ 欧拉定理

欧拉定理是指：如果两个正整数 a 和 n 互质，则 n 的欧拉函数 φ(n)可以让下面的式子成立：

$$a^{\varphi(n)} \equiv 1 \ (mod\ n)$$

即 a 的 φ(n)次方减去 1，被 n 整除。比如，3 和 7 互质，φ(7)=6，(3^6-1)/7=104

如果正整数 a 与质数 p 互质，应为 φ(p) = p-1，所以欧拉函数可写成：

$$a^{p-1} \equiv 1 \ (mod\ p)$$

这个就是著名的费马小定理。

④ 模反元素

如果两个正整数 a 和 n 互质，那么一定可以找到整数 b，使得 ab-1 被 n 整除。

$$ab \equiv 1 \ (mod\ n)$$

比如：3 和 11 互质，那么 3 的模反元素是 4，应为 3*4-1 可以被 11 整除。4 加减 11 的整数倍数都是 3 的模反元素。欧拉定理可以用来证明模反元素必然存在，且为 a 的 φ(n)-1 次方：

$$a^{\varphi(n)} = a * a^{\varphi(n)-1} \equiv 1 \ (mod\ n)$$

5. RSA 加密可行性证明

① 加密

加密要用到公钥(N, E)。假设 Bob 要向 Alice 发送加密信息 m，他就要用 Alice 的公钥(N, E)对 m 进行加密。其中 m 必须是整数（字符串可以取 ASCII 值或 Unicode 值），且 m 必须小于 n。

加密就是计算式中的 c：$m^E \equiv c \pmod n$

假设 m=65，Alice 的公钥(N, E) = (3233, 17)，等式如下：$65^{17} \equiv 2790 \pmod{3233}$。所以密文 c 等于 2790，Bob 就把 2790 发给 Alice。

② 解密

Alice 收到 Bob 发来的 2790 后，就用自己的私钥(3233,2755)进行解密，解密式为 $c^D \equiv m \pmod n$，也就是 c 的 d 次方除以 n 的余数就是 m。因为$2790^{2753} \equiv 65 \pmod{3233}$，因此得到原文 65。

③ 证明

证明为什么用私钥就能解密，就是要证明:

$$c^D \equiv m\ (mod\ n)$$

因为加密规则是：$m^E \equiv c \pmod n$，于是，c 可以写成：$c = m^E - k * n$，其中 k 为正整数。

将 c 代入要我们要证明的那个解密规则:

$$(m^E - k * n)^D \equiv m\ (mod\ n)$$

等同于求证:

$$m^{ED} \equiv m\ (mod\ n)$$

因为 $ED \equiv 1 \pmod{\varphi(n)}$，所以 $ED = h * \varphi(n) + 1$，其中 h 为正整数。将 ED 代入:

$$m^{h*\varphi(n)+1} \equiv m\ (mod\ n)$$

接下来，分成两种情况证明上面这个式子。

a. 当 m 与 n 互质，根据欧拉定理，此时 $m^{\varphi(n)} \equiv 1 \pmod n$，得到:

$$m^{h*\varphi(n)+1} = (m^{\varphi(n)})^h * m \equiv m\ (mod\ n)$$

原式得证。

b. 当 m 与 n 不互质时，此时，由于 n 等于质数 p 和 q 的乘积，所以 m 必然等于 k * p 或 k * q。以 m = k * p 为例，考虑到这时 m 与 q 必然互质，则根据欧拉定理，下面的式子成立:

$$(k * p)^{q-1} \equiv 1\ (mod\ q)$$

进一步得到：

$$[(k*p)^{q-1}]^{h*(p-1)} \times k*p \equiv kp\ (mod\ q)$$

即：

$$(k*p)^{E*D} \equiv kp\ (mod\ q)$$

改写成：

$$(k*p)^{E*D} = t*q + k*p$$

上式 t 必然能被 p 整除（因为右边是 p 的倍数），即 $t = t'*p$

所以

$$(k*p)^{E*D} = t'*p*q + k*p$$

因为$m = k*p$，$n = p*q$，所以：

$$m^{ED} = t'*n + m \equiv m\ (mod\ n)$$

证毕。

④ 拓展证明

前面证明了当 ED ≡ 1 (mod φ(n))时，RSA 加密可逆，但是在前面的描述中，ED ≡ 1 (mod L)，而在 RSA 加密中，L = lcm（p − 1，q − 1），这是一个更苛刻的条件。下面给出证明：

前面部分同上，要证明 $m^{ED} \equiv$ m (mod n) ，也就是 $m^{ED} \equiv$ m (mod p ∗ q) ，只需证明 $m^{ED} \equiv$ m (mod p) 和 $m^{ED} \equiv$ m (mod q) **同时成立**。

对于所有的整数 m，由于 p 和 q 是不同的素数，m 与 p、q 必然互质。E、D 是满足 ED ≡ 1 (mod L) 的正整数，其中 L = lcm (p − 1, q − 1) 可以被 p-1 和 q-1 整除，有 $ed - 1 = h(p-1) = k(q-1)$，其中 h 和 k 是非负整数。

于是 $m^{ED} = m^{ED-1} * m = m^{h(p-1)} * m = \left(m^{(p-1)}\right)^h * m = 1^h * m = m\ (mod\ p)$

同理 $m^{ED} = m^{ED-1} * m = m^{k(q-1)} * m = \left(m^{(q-1)}\right)^k * m = 1^k * m = m\ (mod\ q)$

命题得证。

6. RSA 的可靠性

在 RSA 私钥和公钥生成的过程中，共出现过 p, q, N ,φ(N), E, D，其中(N, E 组成公钥)，其他的都不是公开的，一旦 D 泄露，就等于私钥泄露。那么能不能根据 N, E 推导出 D 呢？

$E * D \equiv 1 (\mathrm{mod}\ \varphi(N))$，只有知道 E 和 φ(N)，才能算出 D；

E 已知，$\varphi(N) = (p-1) * (q-1)$。只有知道 p 和 q，才能算出 φ(N)；

$N = p * q$，只有将 n 分解才能算出 p 和 q。

所以，只有将 n 质因数分解，才能算出 d，也就意味着私钥破译。但是，大整数的质因数分解是非常困难的。

7. 质数的选择

首先要使用概率算法来验证随机产生的大的整数是否是质数，这样的算法比较快而且可以消除掉大多数非质数。假如有一个数通过了这个测试的话，那么要使用一个精确的测试来保证它的确是一个质数。除此之外这样找到的 p 和 q 还要满足一定的要求，首先它们不能太靠近，此外 p-1 或 q-1 的因子不能太小，否则的话 N 也可以被很快地分解。

寻找质数的算法不能给攻击者任何信息，比如这些质数是怎样找到的？尤其产生随机数的软件必须非常好。要求是随机和不可预测。这两个要求并不相同。一个随机过程可能可以产生一个不相关的数的系列，但假如有人能够预测出（或部分地预测出）这个系列的话，那么它就已经不可靠了。比如有一些非常好的随机数算法，但它们都已经被发表，因此它们不能被使用，因为假如一个攻击者可以猜出 p 和 q 一半的位的话，那么他们就已经可以轻而易举地推算出另一半。

另外，2^{128} 里容纳的质数是很多的，也不需要担心质数被用完。

8. RSA 加密算法的缺点

a. 产生密钥很麻烦，受到质数产生技术的限制，因而难以做到一次一密，但也最好使用随机数填充保证即使传输相同的明文，每次的密文都不同，否则攻击者能够识别到同一个信息都是何时被发送。

b. 运算速度慢：由于进行的都是大数计算，使得 RSA 最快的情况也比 AES 慢上好几倍，无论是软件还是硬件实现，速度一直是 RSA 的缺陷，所以一般只用于少量数据的加密。RSA 的速度是对应同样安全级别的对称密码算法的 1/1000 左右。一般使用对称算法来加密数据，然后用 RSA 来加密对称密钥，然后将用 RSA 加密的对称密钥和用对称算法加密的消息发送出去。

9. 明文、密文、密钥的长度

不管明文长度是多少，RSA 生成的密文长度总是固定的。生成密文的长度等于密钥长度。密钥长度越大，生成密文的长度也就越大，加密的速度也就越慢，而密文也就越难被破解掉。我们必须通过定义密钥的长度在"安全"和"加解密效率"之间做出一个平衡的选择。但是明文长度不能超过密钥长度。

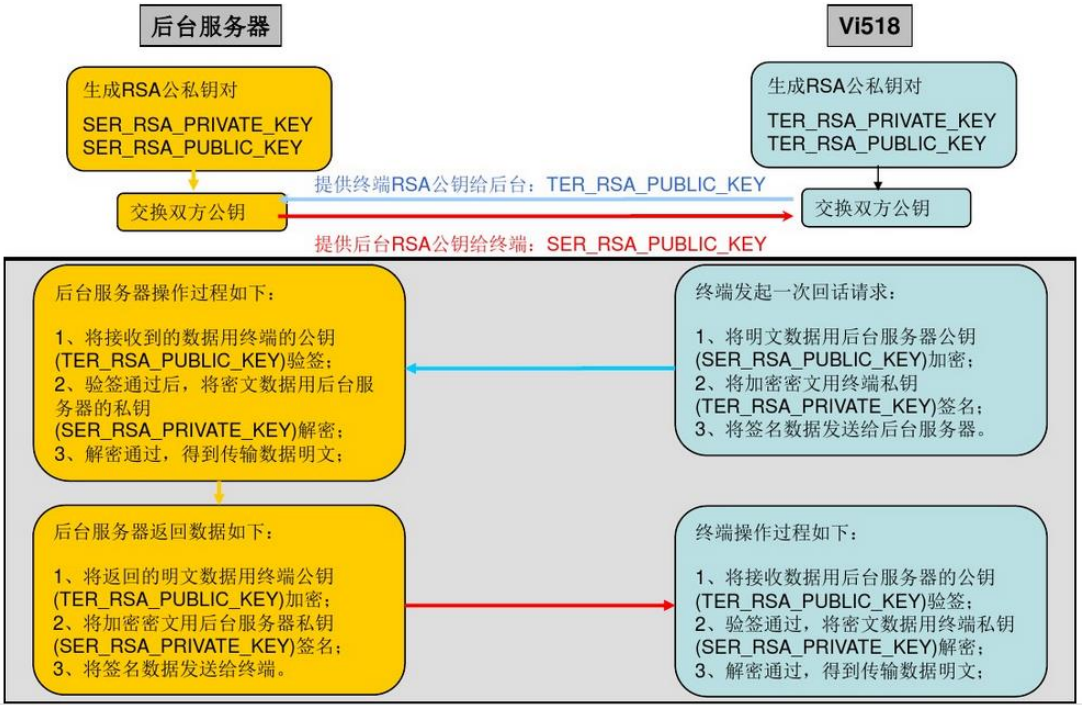
明文长度 < 密文长度 = 秘钥长度

比如 Java 默认的 RSA 加密实现不允许明文长度超过密钥长度减去 11(单位是字节，也就是 byte)。也就是说，如果我们定义的密钥长度为 1024，生成的密钥长度就是 1024 位 / 8 位/字节 = 128 字节，那么我们需要加密的明文长度不能超过 128 字节 -11 字节 = 117 字节。

10. RSA 加密的应用场景

① 客户端认证：用 RSA 加密 MD5 进行传输，用来校验原文在传输过程中是否发生了变化。

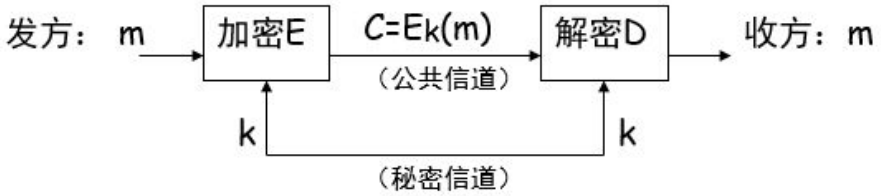
② RSA 签验，如下图所示：



常见攻击方式

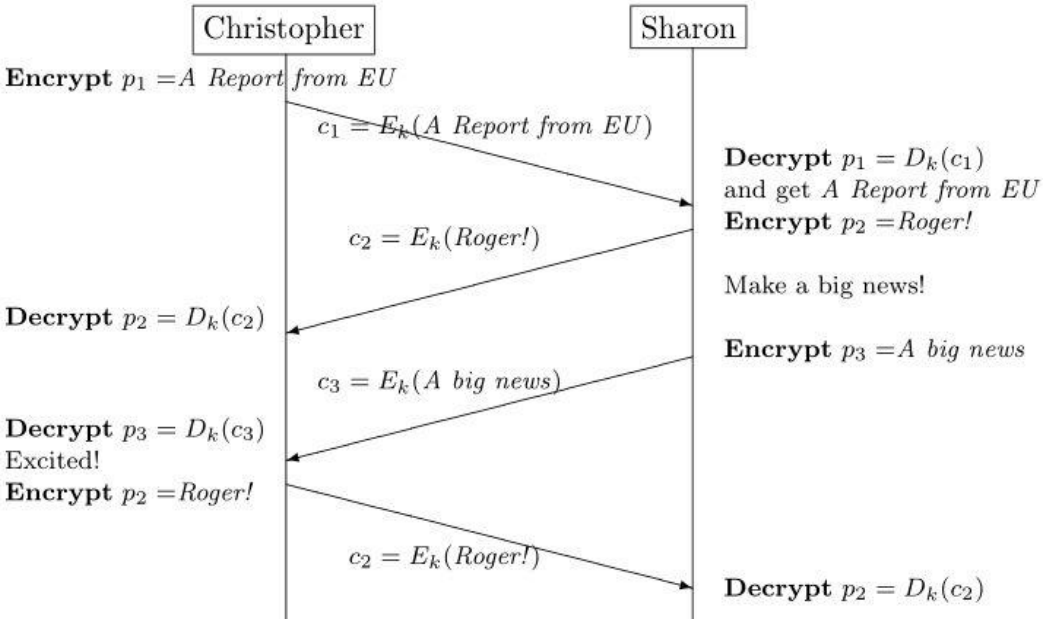
一．明文、密文攻击类

以对称加密体系为例说明（实际对于非对称加密体系也是同理）:



实际在我们的网络安全模型中一般假设所有的加密算法都是公开的，密码体系的安全性依赖于秘钥的安全性，秘钥只有发送方和接收方知道，一旦第三方得知这个秘钥，那他们的通信就被攻破了。

这里我们首先假设通信双方，Sharon 和 Christopher 已经完成了秘钥交换，秘钥 k 是双方已知的，开始进行如下通信:



那么通信过程中一共产生的消息是三组明密文对(p_1, c_1)，(p_2, c_2)和 (p_3, c_3)。

① **唯密文攻击：**只知道密文，也就是c_1，c_2，c_3，那只能通过统计特性分析其中有什么规律了。

② **已知明文攻击：**得到了一些给定的明文和对应的密文，在这里可以是$\{(\mathrm{p}_1,\mathrm{c}_1),(\mathrm{p}_2,\mathrm{c}_2),(\mathrm{p}_3,\mathrm{c}_3)\}$的任意非空子集。但是加密哪些信息，间谍无法控制，所以只能**被动的获知**明文和密文以及明文和密文之间的对应。

③ **选择明文攻击：**除了上面的基础，攻击者还可以**任意创造**一条明文比如“Excited”，并得到其加密后的密文。比如用一定的手段渗透 Sharon 的系统，但是不能直接攻破秘钥，于是只能以她的身份发“Excited”，然后用抓包或者别的方法得到她发送出来的加密的消息。

④ **选择密文攻击：**除了已知明文攻击的基础，攻击者还可以任意制造或者选择一些密文，并得到其解密后的明文。比如用一定的手段在通信过程中伪造消息替换真实消息（但是消息验证码变了），然后窃取 Sharon 获得并解密的结果。通过被制造出来的密文的解密结果，可以推断出传输的密文对应的明文。

以 RSA 公钥加密算法，非对称密码为例：设攻击者为 A，密文接受者为 T，公钥对为(E,n)，私钥为 D，T 收到的密文为 c，c 对应的明文为 m。现在 A 想知道 $m = c^D \ mod\, n$，但是他不想分解 n 去得到 D。于是 T 找了一个随机数 r，$r < n$。他进行如下计算：

$x = r^E \ mod\, n$（对 r 用 T 的公钥加密，得到临时密文 x）

$y = (x * c) \ mod\, n$（将临时密文 x 与密文 c 相乘）

$t = r^{-1} \ mod\, n$

A 利用了 RSA 加密和解密过程的特点，即：如果 $x = r^E \ mod\, n$，那么$r = x^D \ mod\, n$。现在 A 要做的是使 T 用 D 对 y 签名：$u = y^D \ mod\, n$。A 需要获得 u，然后计算 $m = (t * u) \ mod\, n$。

计算结果是这样推导的：

$$(t * u) \ mod\, n = (r^{-1} * y^D) \ mod\, n = (r^{-1} * x^D * c^D) \ mod\, n = c^D \ mod\, n = m$$

总结起来就是已知密文 2（c_2），通过得到明文 1（m_1）对应的密文 1（c_1），进行如下运算：$m_2 = D\,(c_1 * c_2) \div m_1$，选择密文攻击就是攻破了$D\,(c_1 * c_2)$这一步。

解决方式：发送前在消息末尾加上 MAC 来验证密文有没有被改变。

问题：如果攻击者可以接触到受攻击者的解密器，直接用解密器解密想知道的密文不就好了。

⑤ **选择文本攻击：**可以制造任意明文/密文并得到对应的密文/明文，就是上面两者的结合。

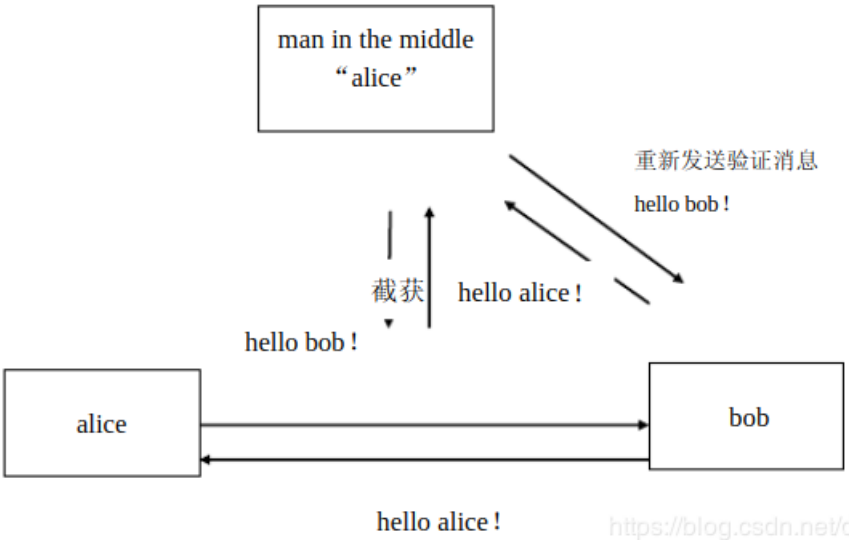
以上**攻击强度自上至下由弱到强**。选择密文攻击和选择明文攻击的不同之处就是加解密方向不同。实际上这些攻击都是 Cryptanalyze，使用哪一种取决于攻击者（cryptanalyst）掌握的资源和手段。

二. 重放攻击与中间人攻击

1. 重放攻击

Replay Attacks，又称为又称重播攻击、回放攻击或新鲜性攻击（Freshness Attacks），是指攻击者发送一个目的主机已接收过的包，来达到欺骗系统的目的，主要用于身份认证过程，破坏认证的正确性。

它是一种攻击类型，这种攻击会不断恶意或欺诈性地重复一个有效的数据传输，重放攻击可以由发起者，也可以由拦截并重发该数据的敌方进行。攻击者利用网络监听或者其他方式盗取认证凭据，之后再把它重新发给认证服务器。从这个解释上理解，加密可以有效防止会话劫持，但是却防止不了重放攻击。重放攻击任何在网络通讯过程中都可能发生，是计算机世界黑客常用的攻击方式之一。



重放攻击示意图

在网络安全中，通过其他方式（非网络监听）盗取 Cookie 与提交 Cookie 也是一种重放攻击。我们有时候可以轻松的复制别人的 Cookie 直接获得相应的权限。(Cookie，是一小段保存在客户机中的文本信息，这个文件与特定的 Web 文档关联在一起，保存了该客户机访问这个 Web 文档时的信息，当客户机再次访问这个 Web 文档时这些信息可供该文档使用)。

防御方案：

① 时间戳验证

请求时加上客户端当前时间戳，**同时签名**（签名是为了防止会话被劫持，时间戳被修改），服务端对请求时间戳进行判断，如超过 5 分钟，认定为重放攻击，请求无效。

但时间戳无法完全防止重放攻击。

② 序号

顾名思义，在客户端和服务端通讯时，先定义一个初始序号，并协商递增方法，每次递增。这样，服务端就可以知道是否是重复发送的请求。

③ 挑战与应答的方式

一般采用这种方式来防御重放攻击。

客户端请求服务器时，服务器会首先生成一个随机数，然后返回给客户端，客户端带上这个随机数，访问服务器，服务器比对客户端的这个参数，若相同，说明正确，不是重放攻击。

这种方式下，客户端每次请求时，服务端都会先生成一个挑战码，客户端带上应答码访问，服务端进行比对，若挑战码和应答码不对应，视为重放攻击。

④ HTTPS 防重放攻击

对于 HTTPS，每个 socket 连接都会验证证书，交换密钥。攻击者截获请求，重新发送，因为 HTTPS 不同，**密钥也不同**，后台解密后是一堆乱码，所以 HTTPS 本身就是防止重放攻击的，除非能复制 socket，或者进行中间人攻击。

⑤ Session

事实上 Web 应用程序是通过 2 种方式来判断和跟踪不同用户的：Cookie 或者 Session（也叫做会话型 Cookie）。其中 Cookie 是存储在本地计算机上的，过期时间很长，所以针对 Cookie 的攻击手段一般是盗取用户 Cookie 然后伪造 Cookie 冒充该用户；而 Session 由于其存在于服务端，随着会话的注销而失效（很快过期），往往难于利用。所以一般来说 Session 认证较之 Cookie 认证安全。

2. 中间人攻击

中间人攻击（Man-in-the-Middle Attack，简称“MITM 攻击”）是一种“间接”的入侵攻击，这种攻击模式是通过各种技术手段将受入侵者控制的一台计算机虚拟放置在网络连接中的两台通信计算机之间，这台计算机就称为“中间人”。然后入侵者把这台计算机模拟一台或两台原始计算机，使“中间人”能够与原始计算机建立活动连接并允许其读取或修改传递的信息，然而两个原始计算机用户却认为他们是在互相通信。通常，这种“拦截数据——修改数据——发送数据”的过程就被称为 “会话劫持”。

在网络安全方面，MITM 攻击的使用是很广泛的，曾经猖獗一时的 SMB 会话劫持、DNS 劫持等技术都是典型的 MITM 攻击手段。在黑客技术越来越多的运用于以获取经济利益为目标的情况下时，MITM 攻击成为对网银、网游、网上交易等最有威胁并且最具破坏性的一种攻击方式。

谈及 MITM 时，并不是只有一种方式可以造成损害——答案是四种！一般说来，有嗅探、数据包注入、会话劫持和 SSL 剥离。

嗅探：数据包嗅探是一种用于捕获流进和流出系统/网络的数据包的技术。网络中的数据包嗅探就好像电话中的监听。如果使用正确，数据包嗅探是合法的；许多公司出于“安全目的”都会使用它。

数据包注入：在这种技术中，攻击者会将恶意数据包注入常规数据中。这样用户便不会注意到文件/恶意软件，因为它们是合法通讯流的一部分。在中间人攻击和拒绝式攻击中，这些文件是很常见的。

会话劫持：你曾经遇到过“会话超时”错误吗？如果你进行过网上支付或填写过一个表格，你应该知道它们。在你登录进你的银行账户和退出登录这一段期间便称为一个会话。这些会话通常都是黑客的攻击目标，因为它们包含潜在的重要信息。在大多数案例中，黑客会潜伏在会话中，并最终控制它。这些攻击的执行方式有多种，比如 SSL 劫持。

SSL 劫持攻击：攻击者传输过程中伪造证书，将服务器的公钥替换为自己的公钥，这样客户端的公钥加密消息中间人就可以用自己的私钥破解，再用劫持的服务器公钥加密好传给服务器。但是对于客户端，如果中间人伪造了证书，在校验过程中就会**提示证书错误**，这时候就怕你点继续访问，此时中间人就可以获取客户端和服务端之间的通讯。

SSL 剥离：SSL 剥离或 SSL 降级攻击是 MITM 攻击的一种十分罕见的方式，但是也是最危险的一种。众所周知，SSL/TLS 证书通过加密保护着我们的通讯安全。在 SSL 剥离攻击中，攻击者使 SSL/TLS 连接剥落，随之协议便从安全的 HTTPS 变成了不安全的 HTTP。于是在发送方给中间人的是 HTTP，而中间人传给接收方的是 HTTPS。

防御方案：

对于 DNS 劫持，要记得检查本机的 HOSTS 文件，以免被攻击者加了恶意站点进去，即用假的 IP 地址对应正常的域名；其次要确认自己使用的 DNS 服务器是 ISP 提供的。

对于数据包注入，只要数据是 SSL 加密的就不会被轻易注入数据。而对于 HTTPS 证书的攻击，只要保证访问网站是 HTTPS 而且证书没问题就可以避免了。

至于**局域网内**各种各样的会话劫持（局域网内的代理除外），因为它们都要结合嗅探以及欺骗技术在内的攻击手段，必须依靠 ARP 和 MAC 做基础，所以网管应该使用交换式网络（通过交换机传输）代替共享式网络（通过集线器传输），这可以降低被窃听的机率。当然要想根除会话劫持，还必须使用静态 ARP、捆绑 MAC+**IP** 等方法来限制欺骗，以及采用认证方式的连接等。

但是对于“代理中间人攻击”而言，以上方法就难以见效了，因为代理服务器本来就是一个“中间人”角色，攻击者不需要进行任何欺骗就能让受害者自己连接上来，而且代理也不涉及 MAC 等因素，所以一般的防范措施都不起作用。

3. 两者区别

重放攻击是攻击者获取客户端发送给服务器端的包，不做修改，原封不动的发送给服务器用来实现某些功能。中间人攻击是攻击者把自己当作客户端与服务器端的中间人，客户端发送的信息会被攻击者截取然后做一些操作再发送给服务器端，服务器端响应返回的包也会被攻击者截取然后再发送给客户端。

重放攻击是欺骗服务器，比如说客户端发送给服务器端一个包的功能是查询某个信息，攻击者拦截到这个包，然后想要查询这个信息的时候，把这个包发送给服务器，服务器就会做相应的操作，返回查询的信息。中间人攻击是欺骗双方，相对于客户端来说攻击者是服务器端，相对于服务器段来说攻击者是客户端，从而获取所有的信息。中间人的目的是让 A 和 B 两人都觉得自己在和对方进行直接的对话，但其实 A 和 B 两人相互发送的信息是在由一个中间人转交。