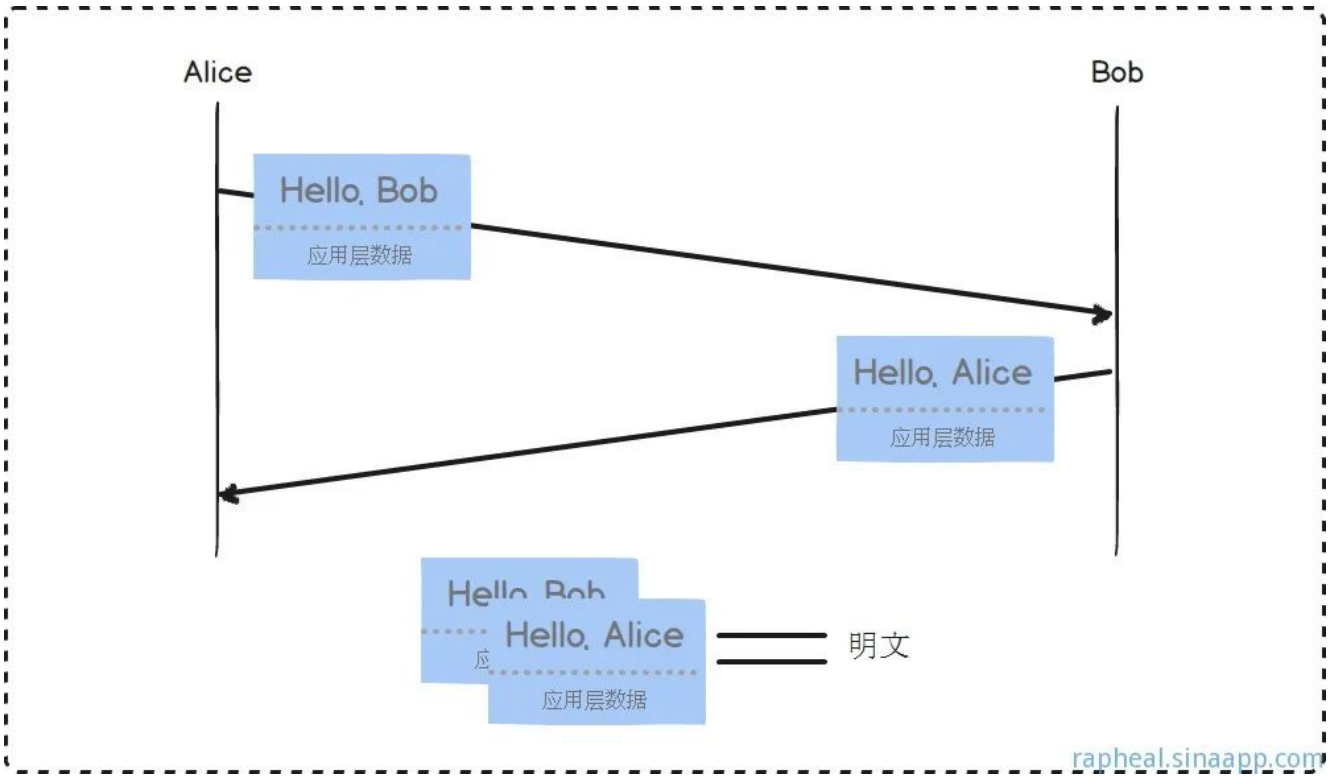
为什么HTTPS协议就比HTTP安全呢？一次安全可靠的通信应该包含什么东西呢，这篇文章我会尝试讲清楚这些细节。

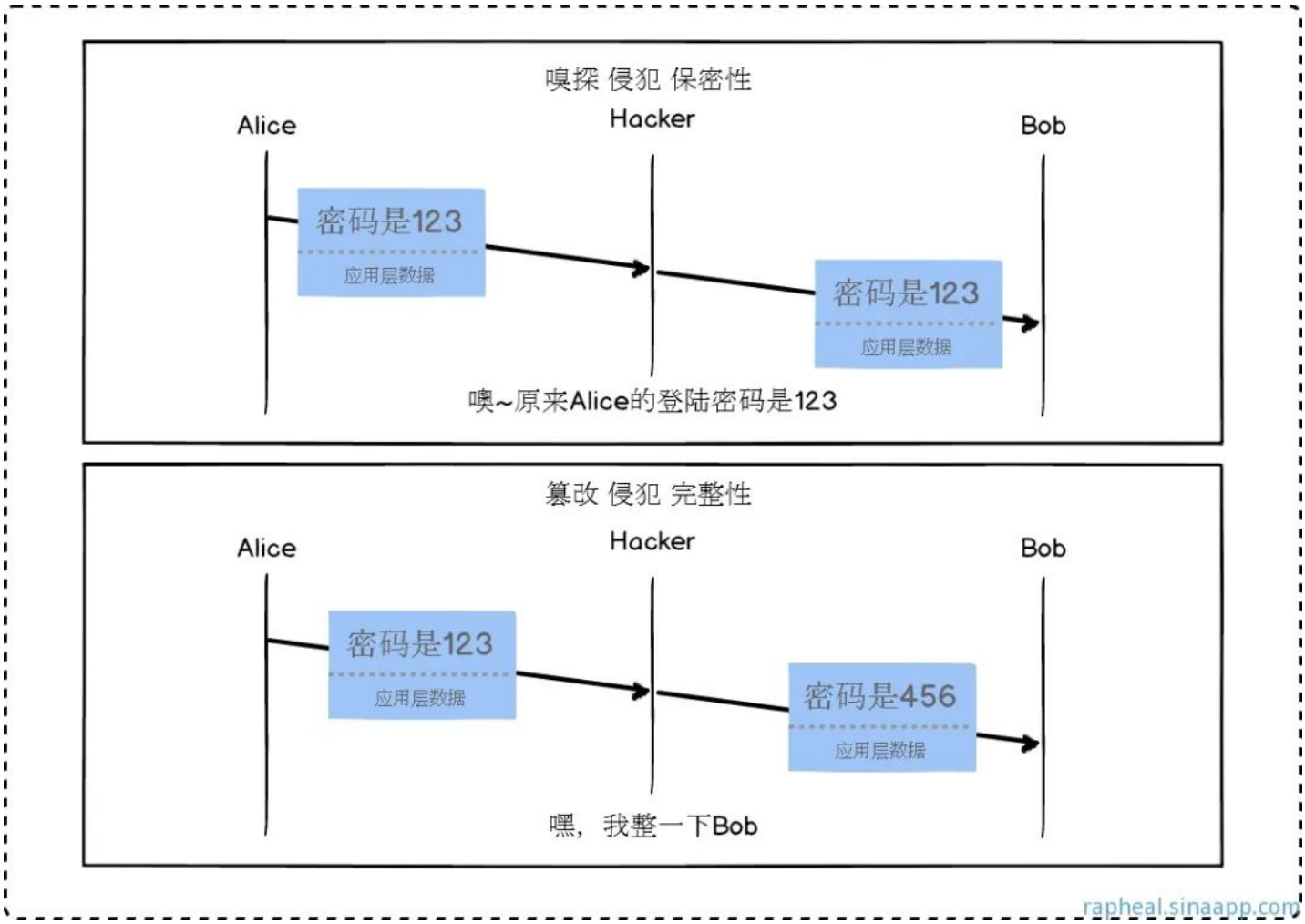
Alice与Bob的通信

我们以Alice与Bob一次通信来贯穿全文，一开始他们都是用明文的形式在网络传输通信内容。



嗅探以及篡改

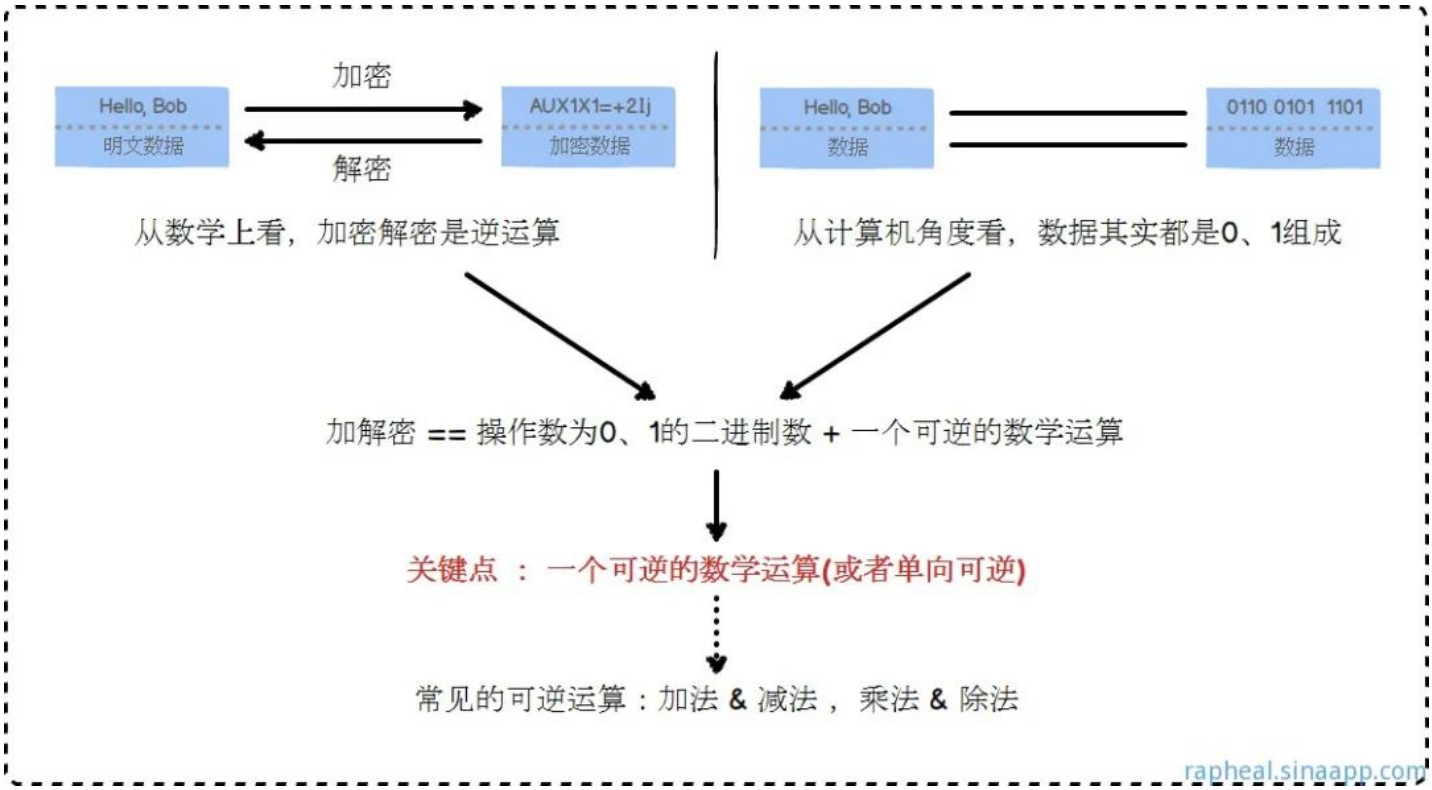
如果在他们的通信链路出现了一个Hacker，由于通信内容都是明文可见，所以Hacker可以嗅探看到这些内容，也可以篡改这些内容。



公众号的文章之前就遇到很多被挟持篡改了内容，插入广告。

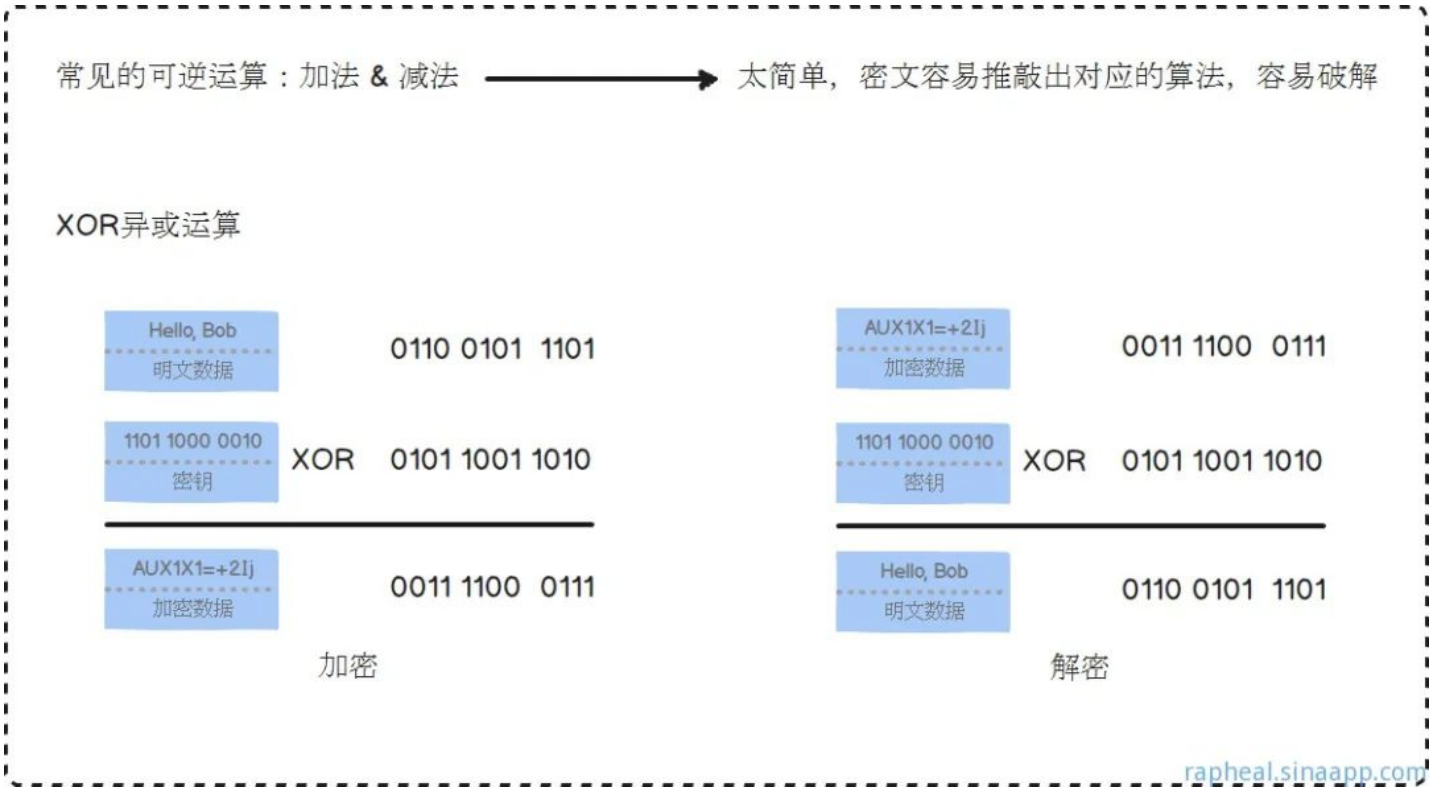
加密解密

既然明文有问题，那就需要对明文进行加密处理，让中间人看不懂内容，于是乎要对原来的内容变成一段看不懂的内容，称为加密，反之则是解密。而本质其实就是一种数学运算的逆运算，类似加法减法，例如发送方可以将 abcd…xyz 每个字母+1映射成 bcd…yza，使得原文的字母变成看不懂的序列，而接收方只需要将每个字母-1就可以恢复成原来的序列，当然这种做法规律太容易被破解了，后边会有个案例示意图。



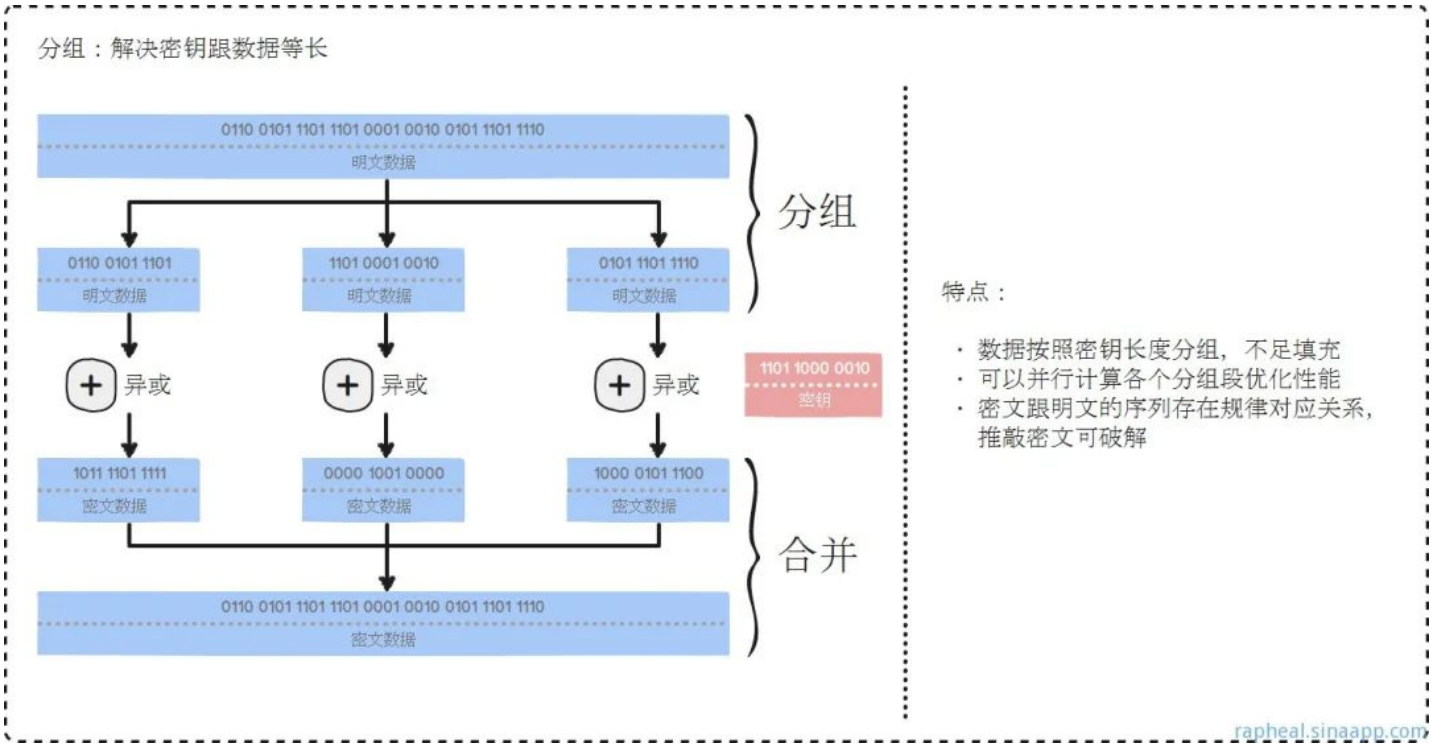
对称加密

如果对2个二进制数A和B进行异或运算得到结果C, 那C和B再异或一次就会回到A，所以异或也可以作为加密解密的运算。

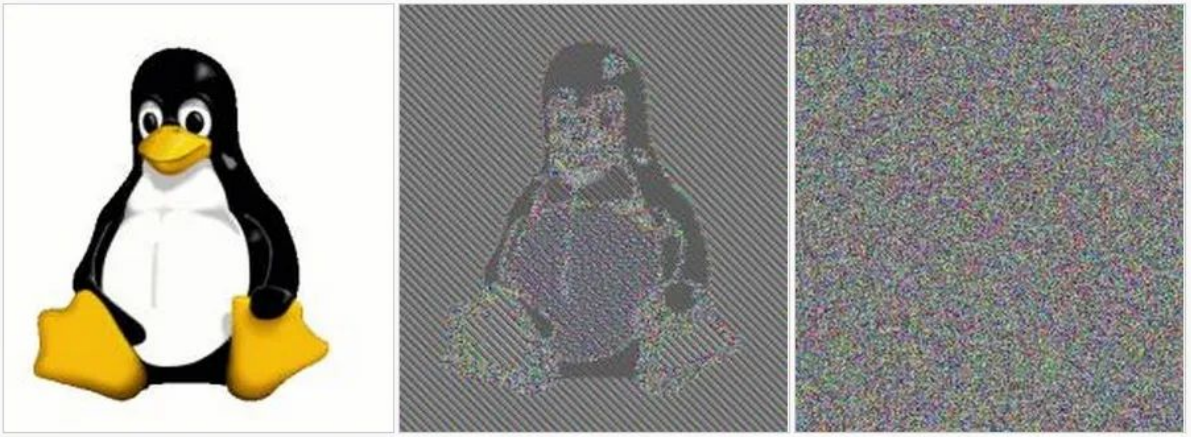


把操作数A作为明文，操作数B作为密钥，结果C作为密文。可以看到加密解密运用同一个密钥B，把这种加解密都用同一个密钥的方式叫做对称加密。

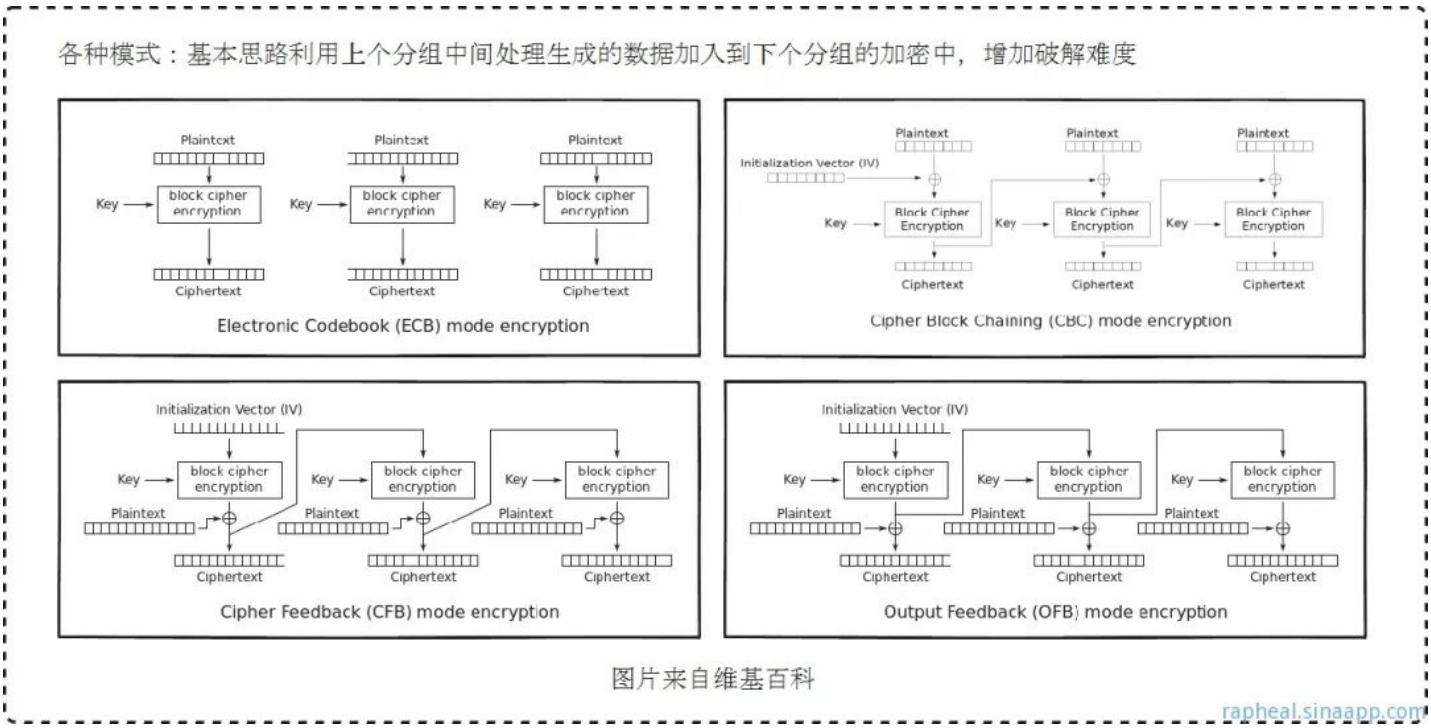
可以看到简单的异或加密/解密操作，需要密钥跟明文位数相同。为了克服这个缺点，需要改进一下，把明文进行分组，每组长度跟密钥一致，分别做异或操作就可以得到密文分片，再合并到一起就得到密文了。



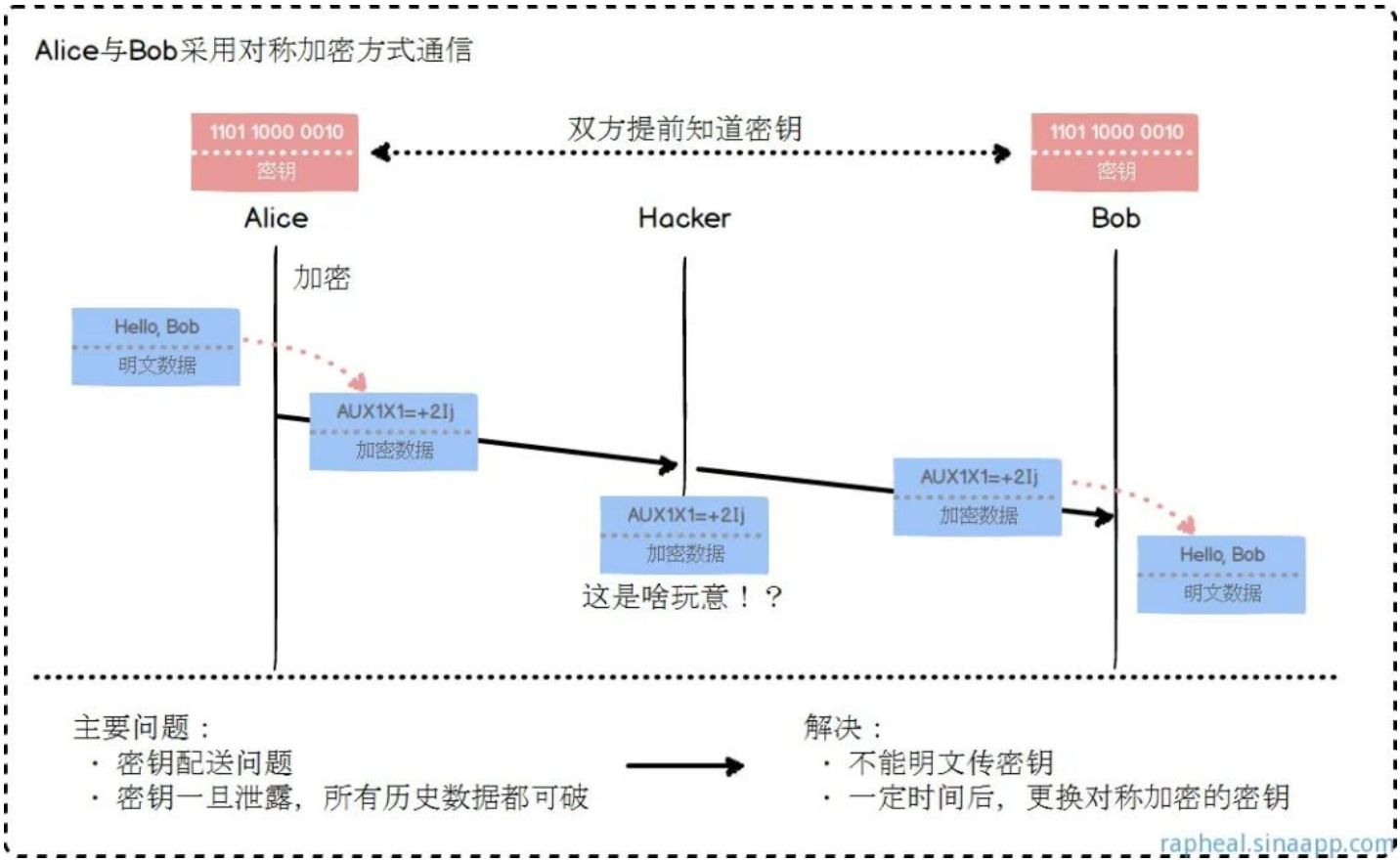
但是这种简单分组的模式也是很容易发现规律，可以从下图看到，中间采用对原图进行DES的ECB模式加密（就是上边提到简单分组的模式）



很明显，原图一些特征在加密后还是暴露无遗，因此需要再改进一把。一般的思路就是将上次分组运算的结果/中间结果参与到下次分组的运算中去，使得更随机混乱，更难破解。以下图片来自维基百科：

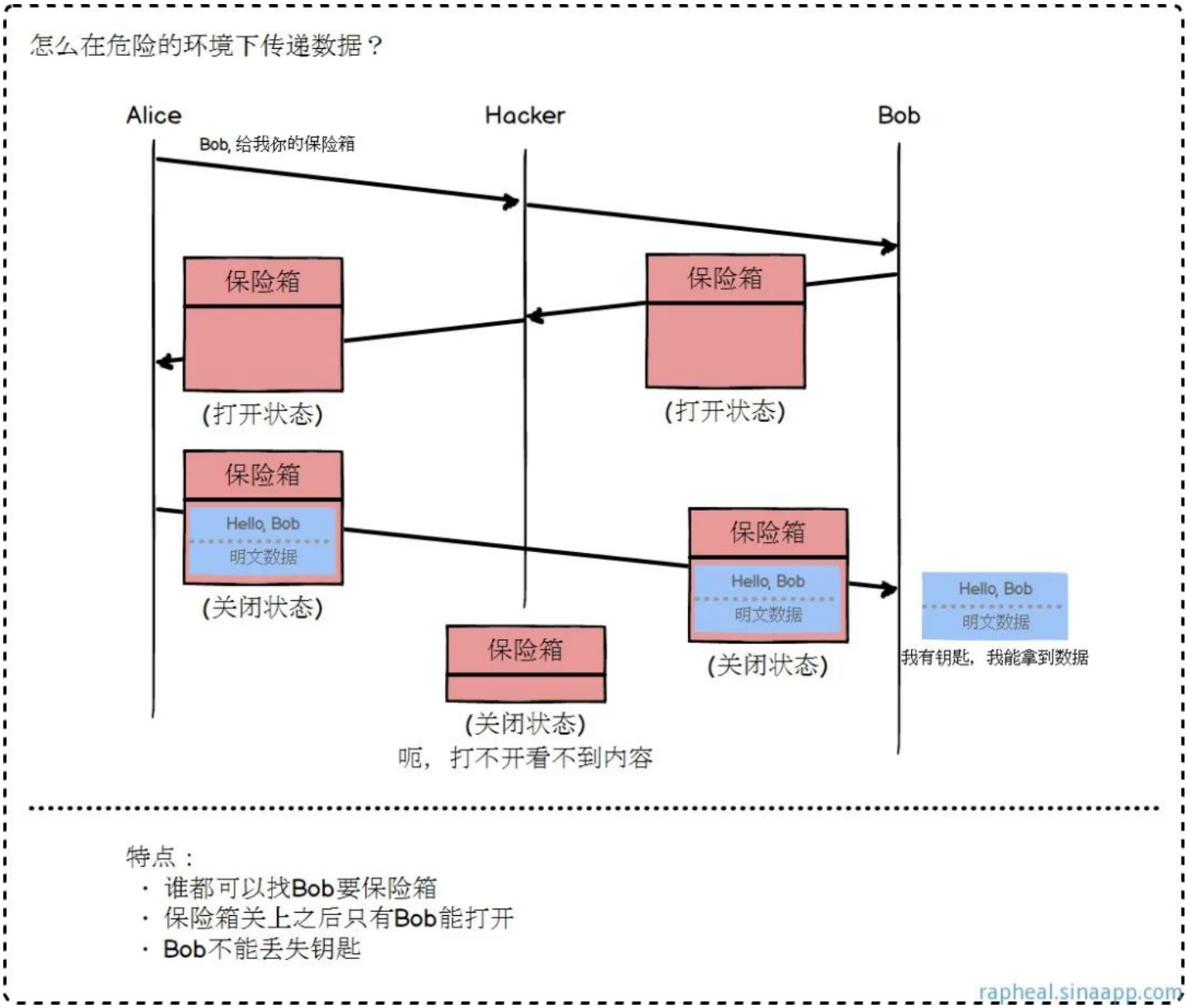


经过改良后，Alice与Bob如果能提前拿到一个对称加密的密钥，他们就可以通过加密明文来保证他们说话内容不会被Hacker看到了。

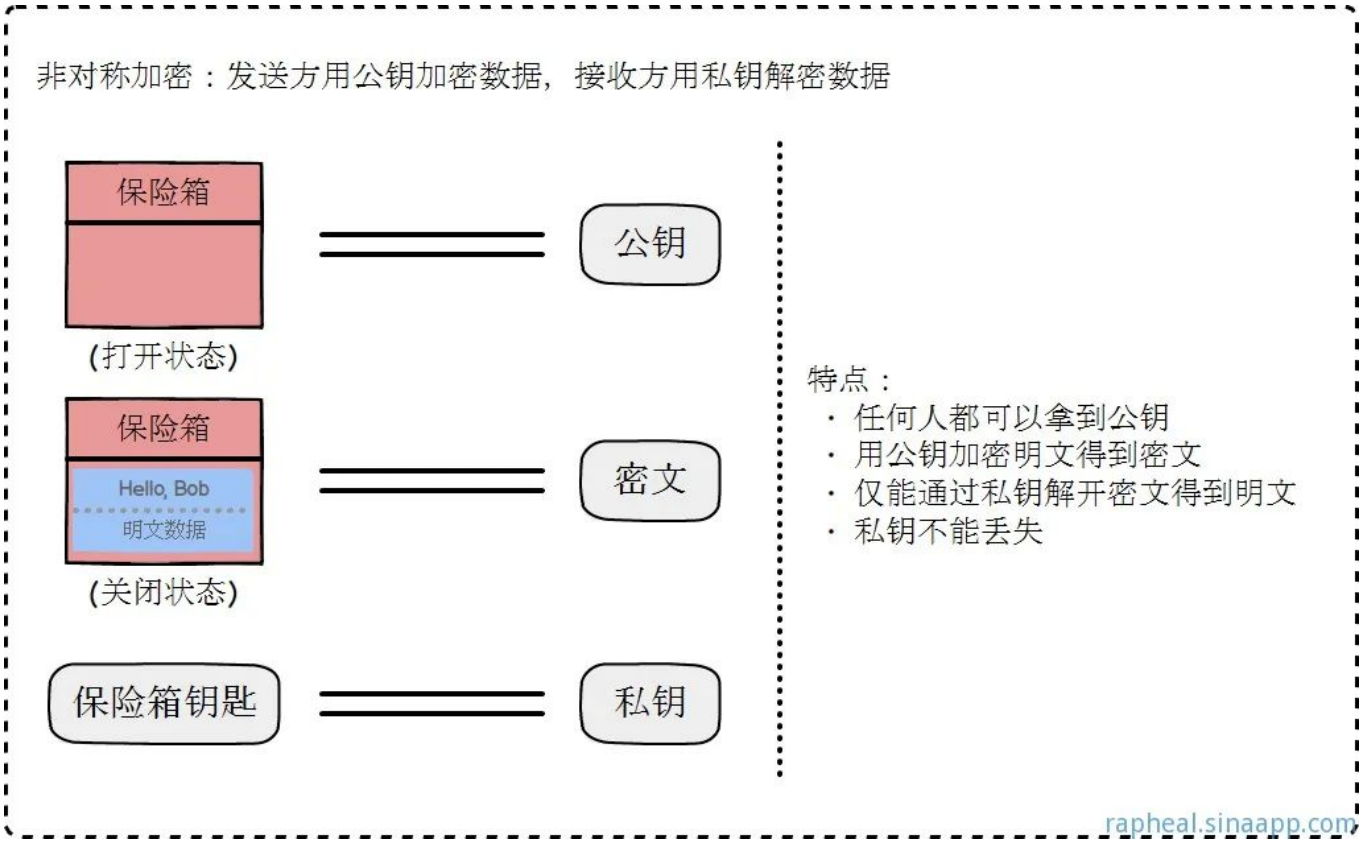


非对称加密

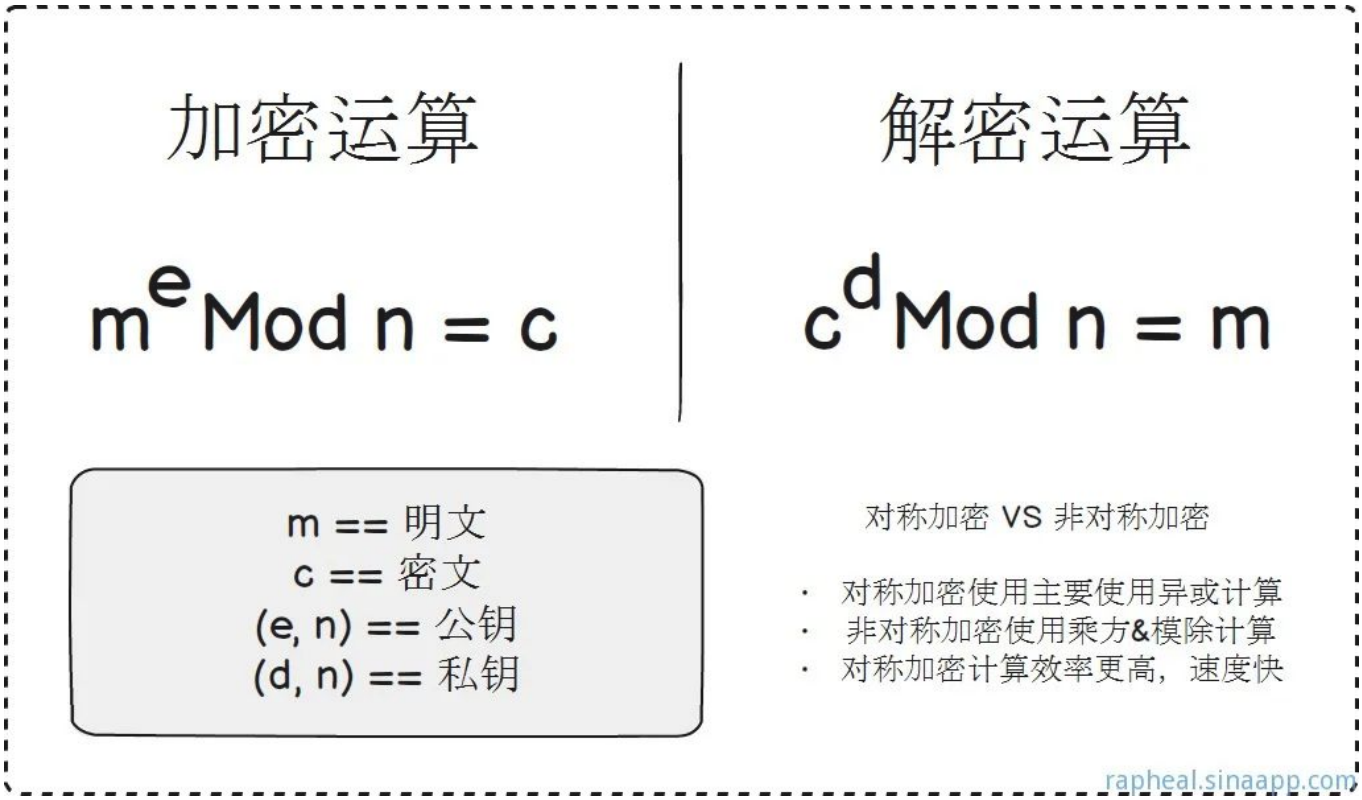
刚刚还引发另一个问题，这个对称加密用到的密钥怎么互相告知呢？如果在传输真正的数据之前，先把密钥传过去，那Hacker还是能嗅探到，那之后就了无秘密了。于是乎出现另外一种手段：



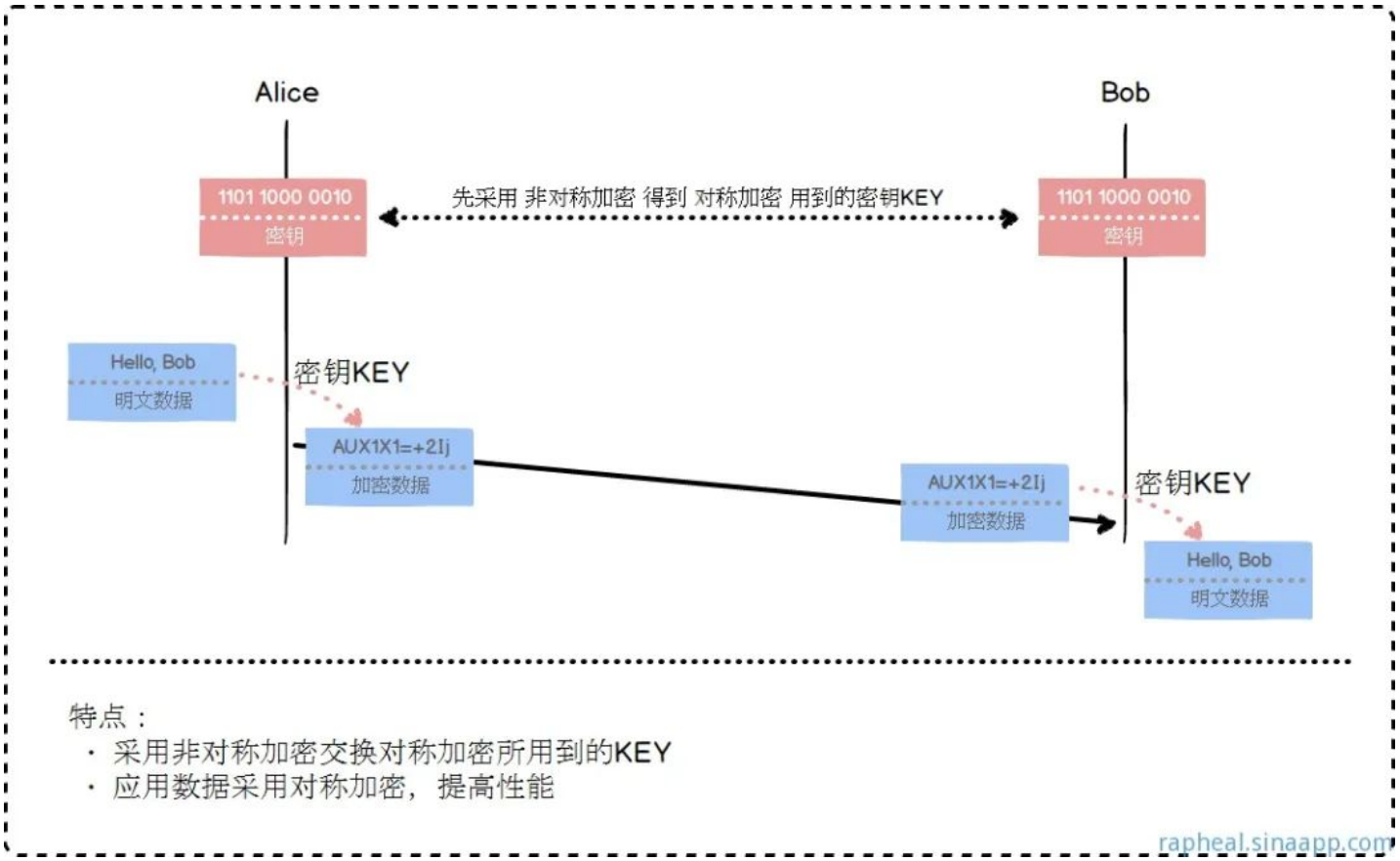
这就是非对称加密，任何人都可以通过拿到Bob公开的公钥对内容进行加密，然后只有Bob自己私有的钥匙才能解密还原出原来内容。



RSA就是这样一个算法，具体数学证明利用了大质数乘法难以分解、费马小定理等数学理论支撑它难以破解。相对于前边的对称加密来说，其需要做乘法模除等操作，性能效率比对称加密差很多。

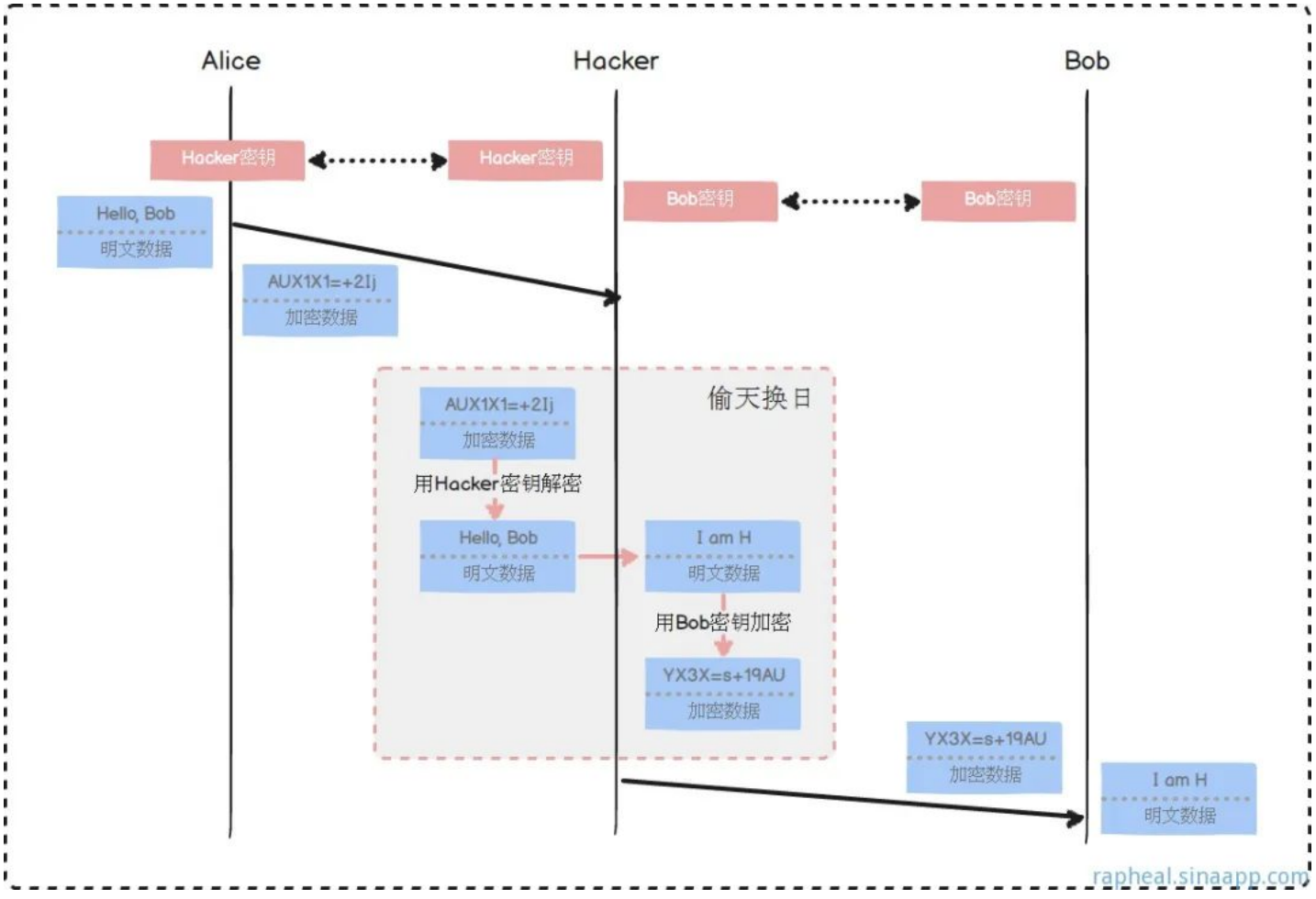


由于非对称加密的性能低，因此我们用它来先协商对称加密的密钥即可，后续真正通信的内容还是用对称加密的手段，提高整体的性能。

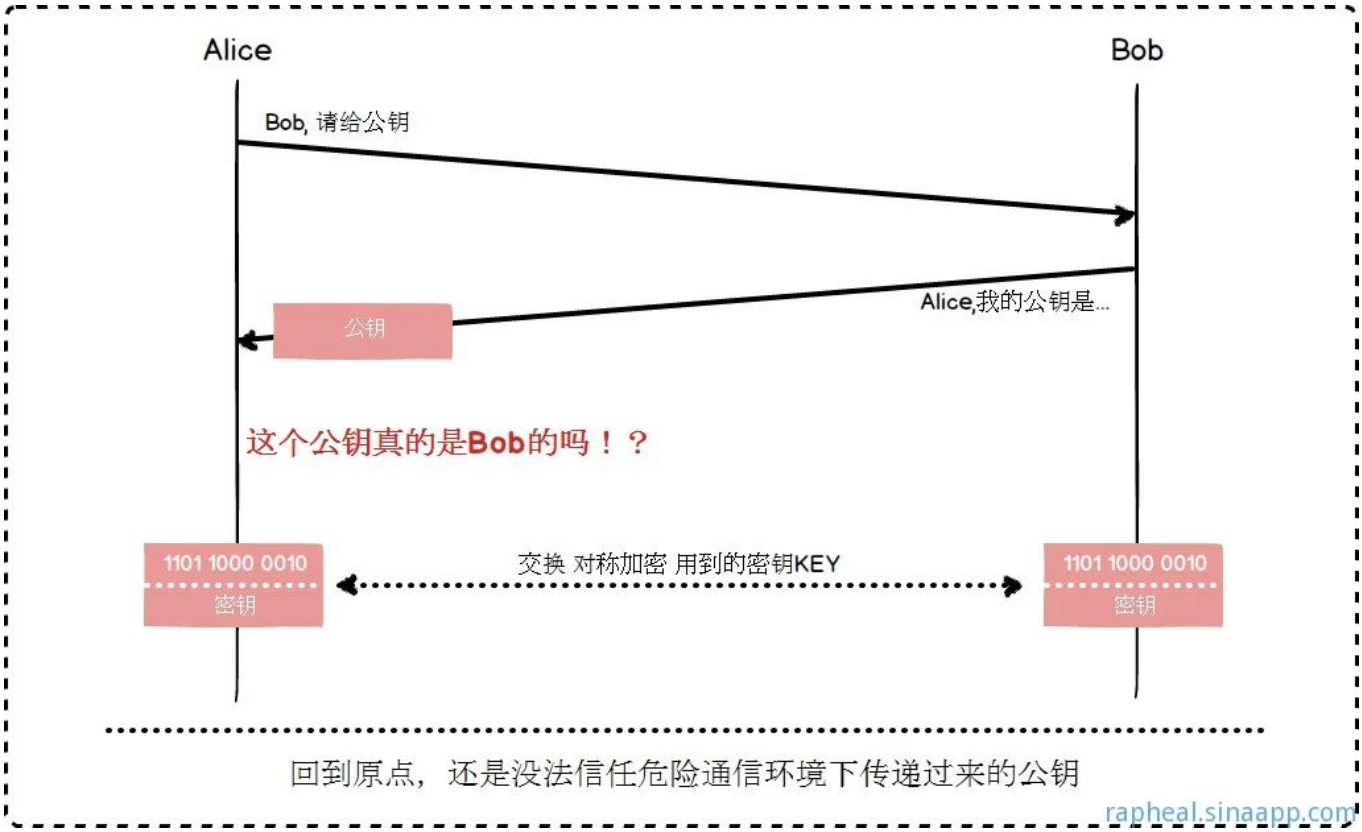


认证

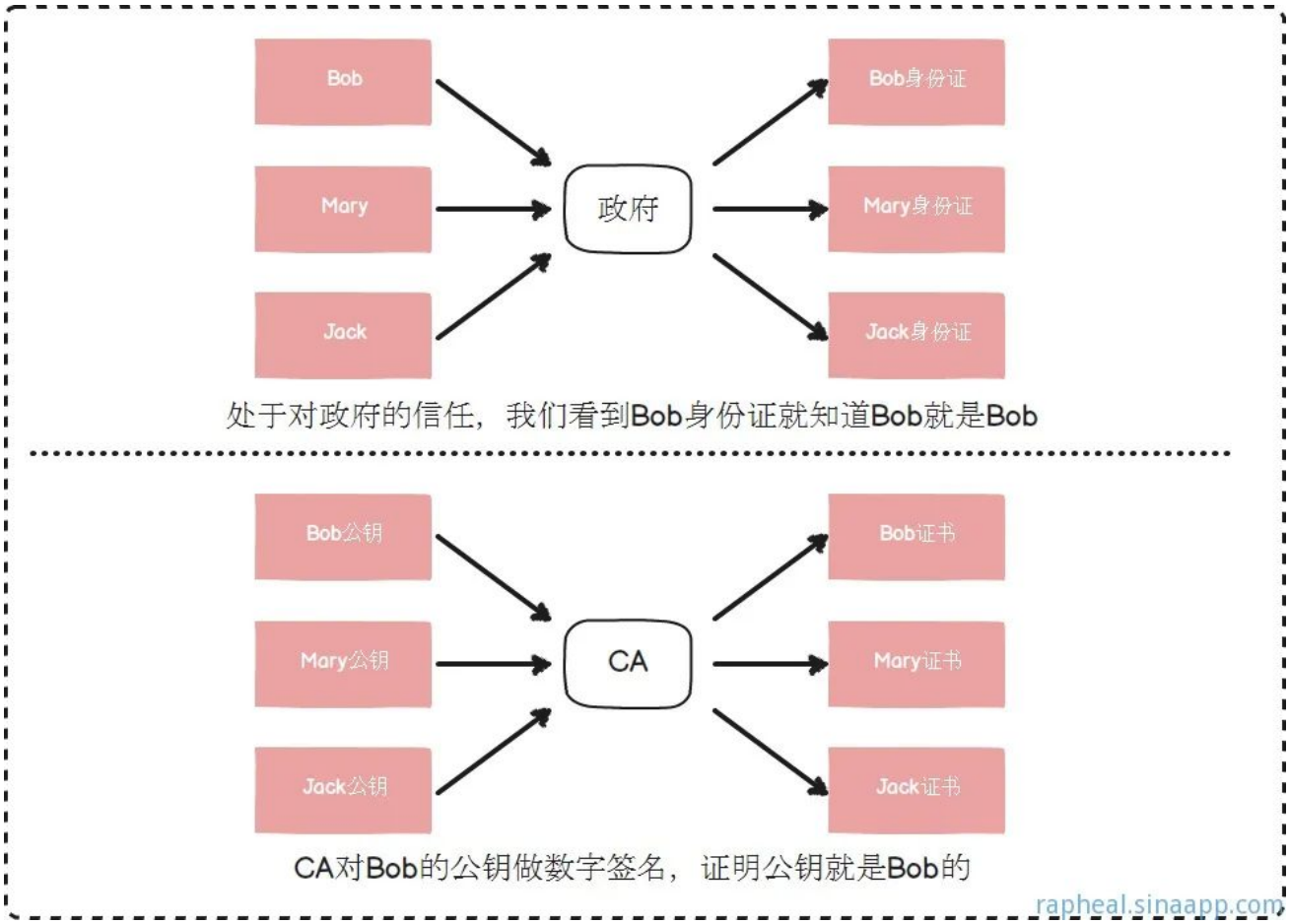
上边虽然解决了密钥配送的问题，但是中间人还是可以欺骗双方，只要在Alice像Bob要公钥的时候，Hacker把自己公钥给了Alice，而Alice是不知道这个事情的，以为一直都是Bob跟她在通信。



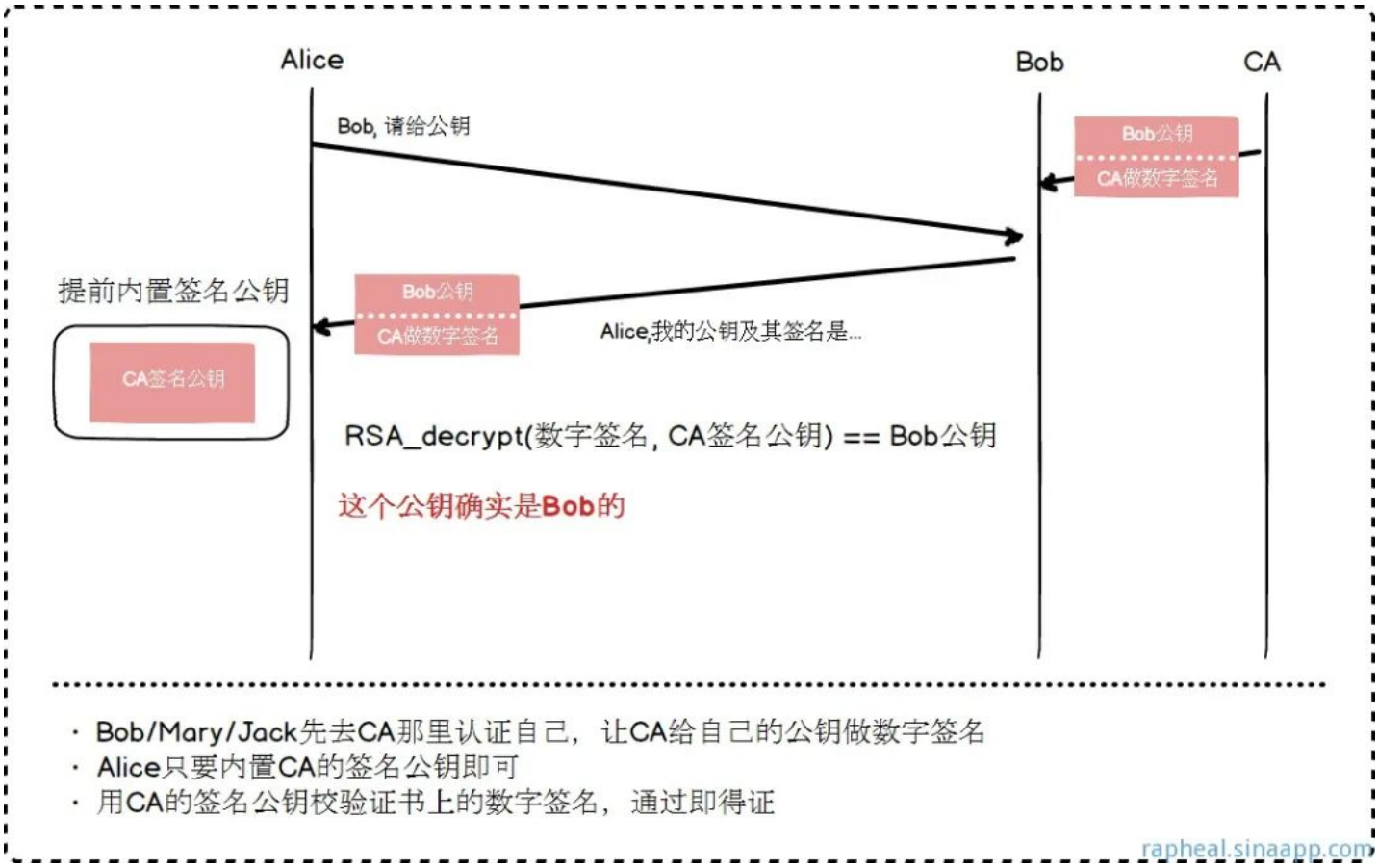
要怎么证明现在传过来的公钥就是Bob给的呢？在危险的网络环境下，还是没有解决这个问题。



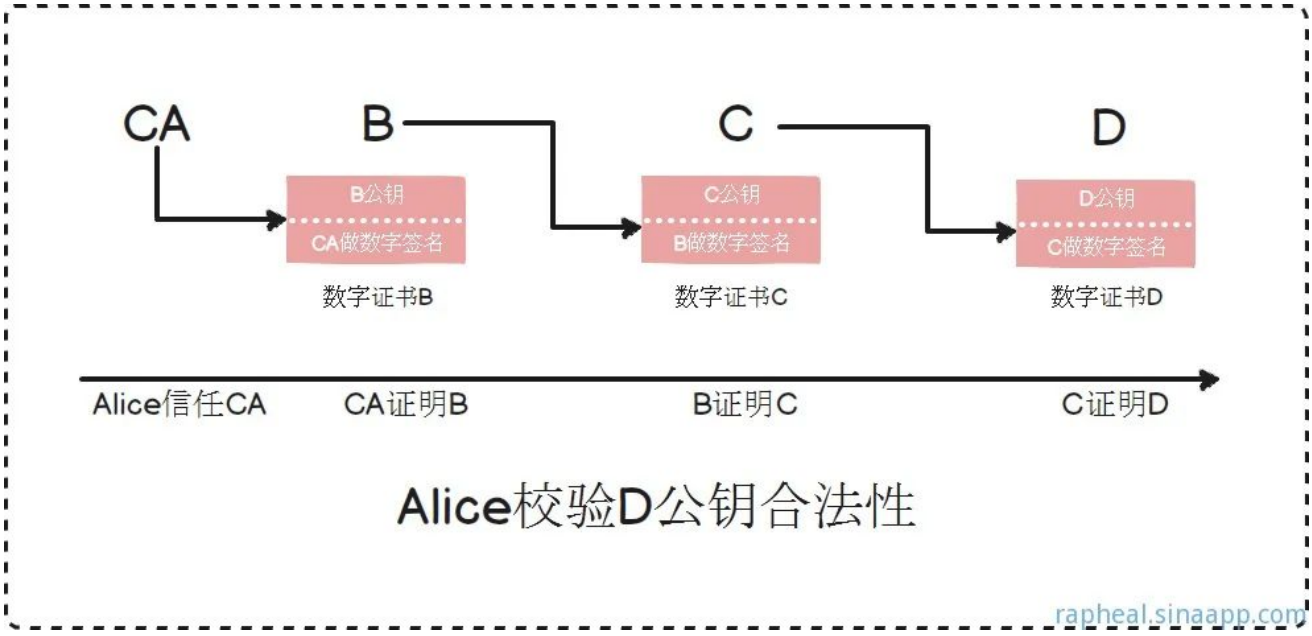
一般我们现实生活是怎么证明Bob就是Bob呢？一般都是政府给我们每个人发一个身份证（假设身份证没法伪造），我只要看到Bob身份证，就证明Bob就是Bob。
网络也可以这么做，如果有个大家都信任的组织CA给每个人出证明，那Alice只要拿到这个证明，检查一下是不是CA制作的Bob证书就可以证明Bob是Bob。所以这个证书里边需要有两个重要的东西：Bob的公钥+CA做的数字签名。



前边说到用公钥进行加密，只有拥有私钥的人才能解密。数字证书有点反过来：用私钥进行加密，用公钥进行解密。CA用自己的私钥对Bob的信息（包含Bob公钥）进行加密，由于Alice无条件信任CA，所以已经提前知道CA的公钥，当她收到Bob证书的时候，只要用CA的公钥对Bob证书内容进行解密，发现能否成功解开（还需要校验完整性），此时说明Bob就是Bob，那之后用证书里边的Bob公钥来走之前的流程，就解决了中间人欺骗这个问题了。
这种方式也是一种防抵赖的方式，让对方把消息做一个数字签名，只要我收到消息，用对方的公钥成功解开校验这个签名，说明这个消息必然是对方发给我的，对方不可以抵赖这个行为，因为只有他才拥有做数字签名的私钥。

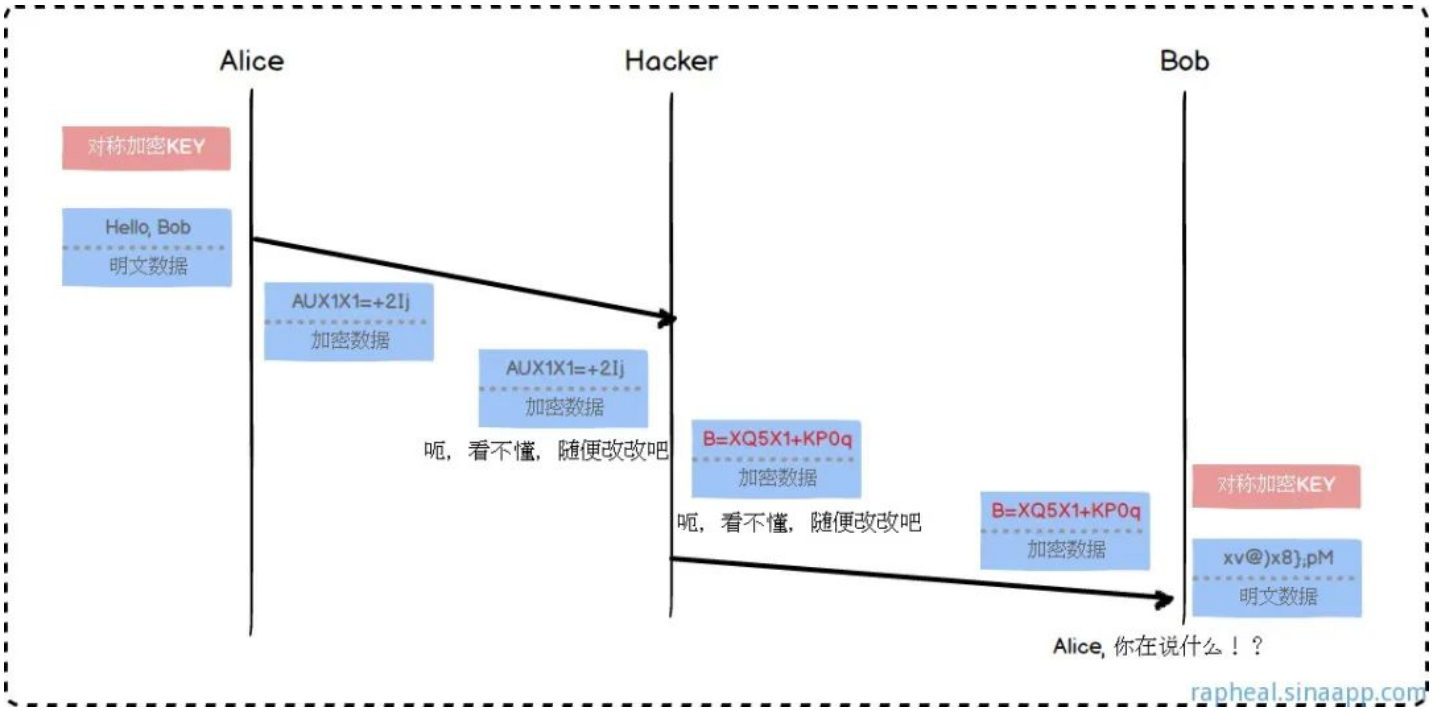


CA其实是有多级关系，顶层有个根CA，只要他信任B，B信任C，C信任D，那我们基本就可以认为D是可信的。

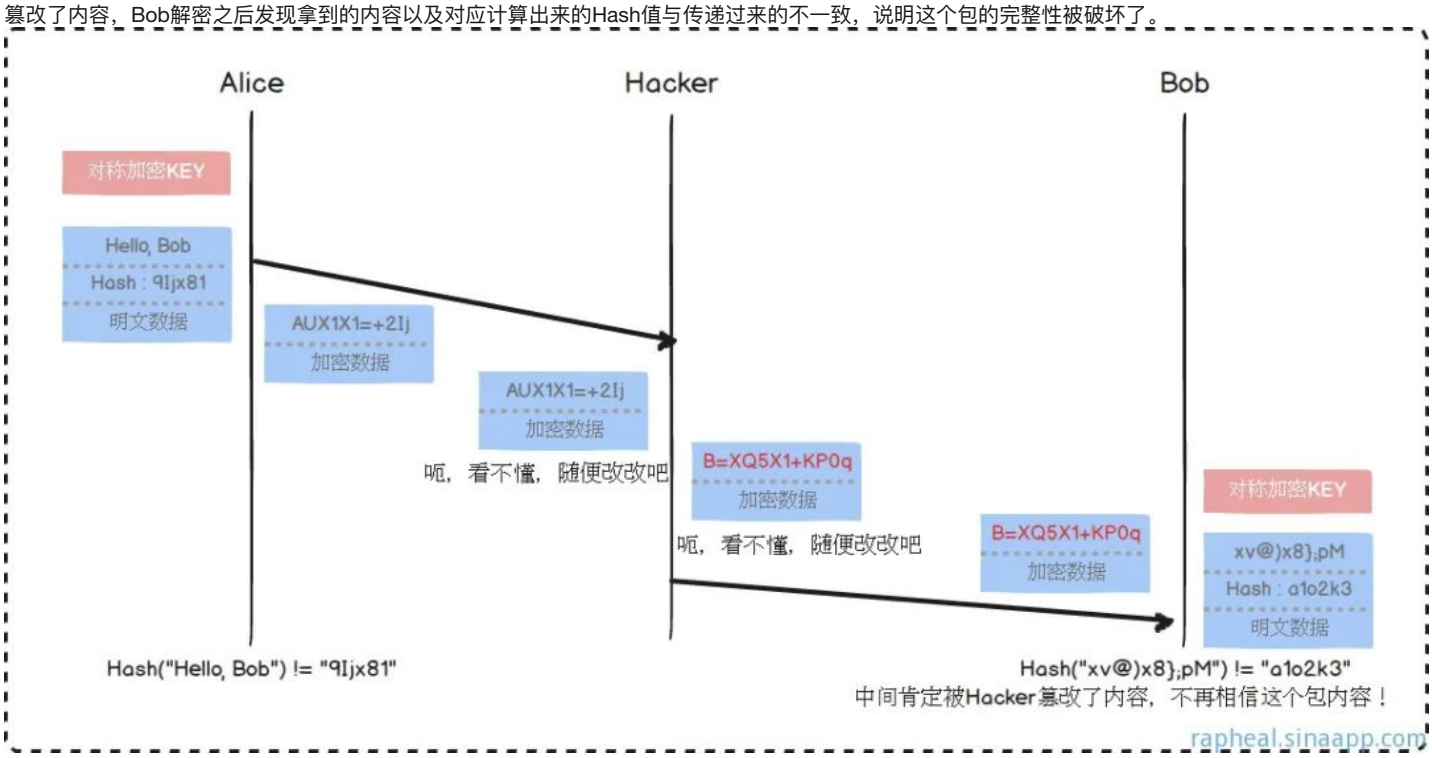


完整性

上边基本上已经解决了保密性和认证，还有一个完整性没有保障。虽然Hacker还是看不懂内容，但是Hacker可以随便篡改通信内容的几个bit位，此时Bob解密看到的可能是很乱的内容，但是他也不知道这个究竟是Alice真实发的内容，还是被别人偷偷改了的内容。



单向Hash函数可以把输入变成一个定长的输出串，其特点就是无法从这个输出还原回输入内容，并且不同的输入几乎不可能产生相同的输出，即便你要特意去找也非常难找到这样的输入（抗碰撞性），因此Alice只要将明文内容做一个Hash运算得到一个Hash值，并一起加密传递过去给Bob。Hacker即便篡改了内容，Bob解密之后发现拿到的内容以及对应计算出来的Hash值与传递过来的不一致，说明这个包的完整性被破坏了。

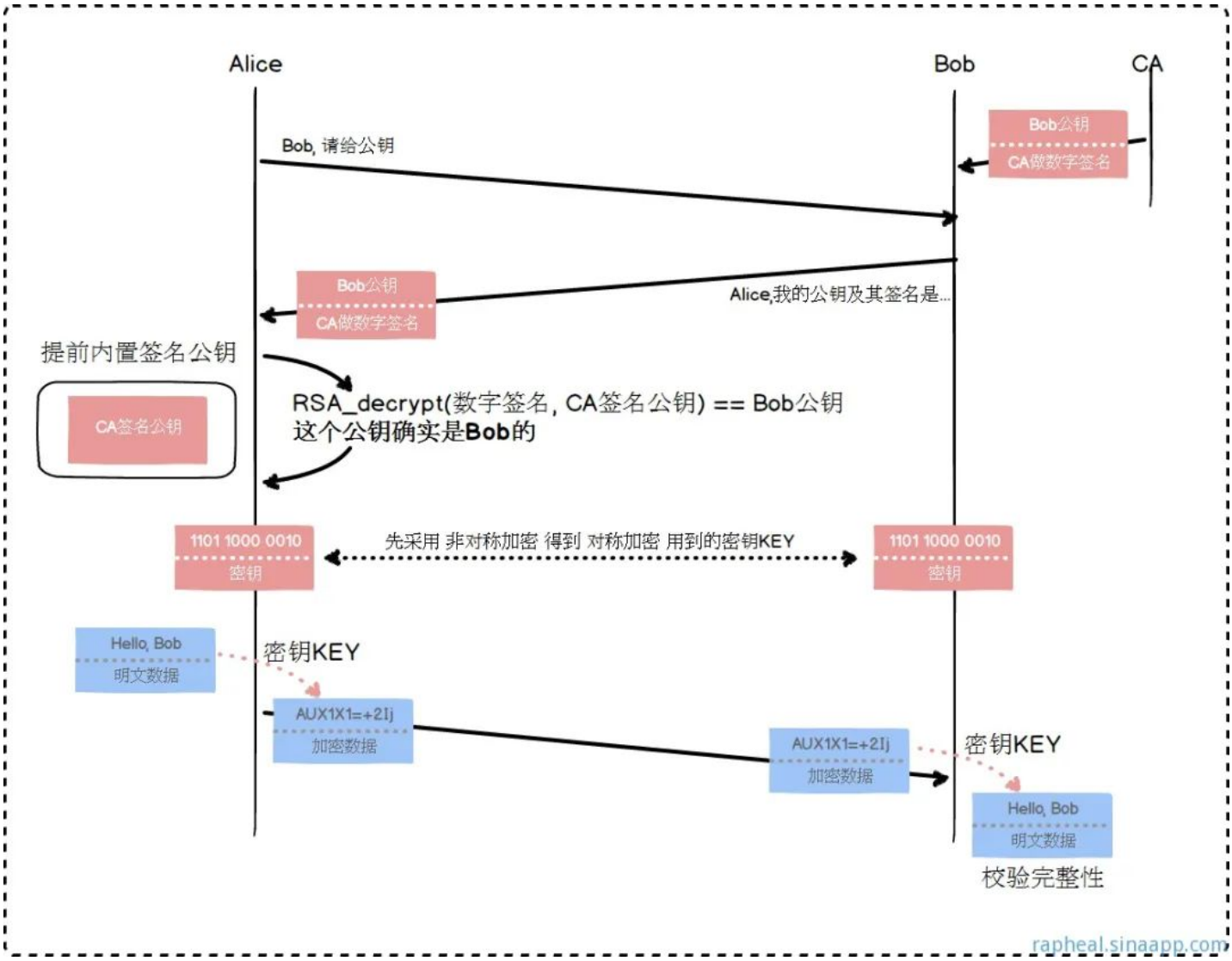


一次安全可靠的通信

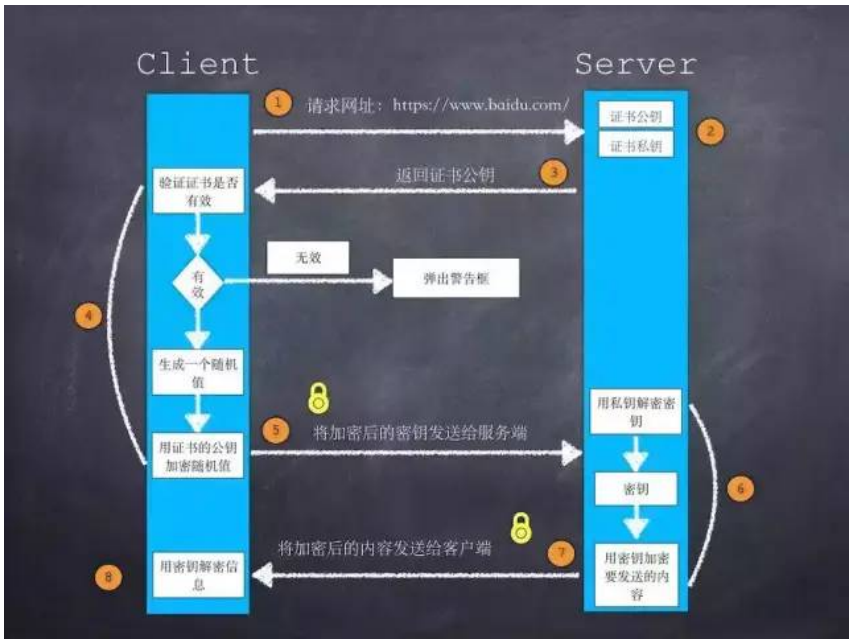
总结一下，安全可靠的保障：

1. 对称加密以及非对称加密来解决：保密性
2. 数字签名：认证、不可抵赖
3. 单向Hash算法：完整性

来一张完整的图：



https握手总结：



1. 客户端发起HTTPS请求

2. 服务端的配置

采用HTTPS协议的服务器必须要有一套数字证书，可以是自己制作或者CA证书。区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用CA证书则不会弹出提示页面。这套证书其实就是一对公钥和私钥。公钥给别人加密使用，私钥给自己解密使用。

3. 传送证书

这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构，过期时间等。

4. 客户端解析证书

这部分工作是有客户端的TLS来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等，如果发现异常，则会弹出一个警告框，提示证书存在问题。如果证书没有问题，那么就生成一个随即值，然后用证书对该随机值进行加密。

5. 传送加密信息

这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值来进行加密解密了。

6. 服务段解密信息

服务端用私钥解密后，得到了客户端传过来的随机值(私钥)，然后把内容通过该值进行对称加密。所谓对称加密就是，将信息和私钥通过某种算法混合在一起，这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

7. 传输加密后的信息

这部分信息是服务段用私钥加密后的信息，可以在客户端被还原。

8. 客户端解密信息

客户端用之前生成的私钥解密服务段传过来的信息，于是获取了解密后的内容。

PS：整个握手过程第三方即使监听到了数据，也束手无策。

总结

为什么HTTPS是安全的?

在HTTPS握手的第四步中，如果站点的证书是不受信任的，会显示出现下面确认界面，确认了网站的真实性。另外第六和八步，使用客户端私钥加密解密，保证了数据传输的安全。

HTTPS和HTTP的区别

1. https协议需要到ca申请证书或自制证书。

2. http的信息是明文传输，https则是具有安全性的ssl加密。

3. http是直接与TCP进行数据传输，而https是经过一层SSL（OSI表示层），用的端口也不一样，前者是80（需要国内备案），后者是443。

4. http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

注意https加密是在传输层

https报文在被包装成tcp报文的时候完成加密的过程，无论是https的 header域 也好，body域也罢都是会被加密的。

当使用 tcpdump或者wireshark 之类的 tcp层工具抓包 ，获取是加密的内容，而如果用应用层抓包，使用 Charels(Mac)、Fildder(Windows) 抓包工具，那当然看到是明文的。

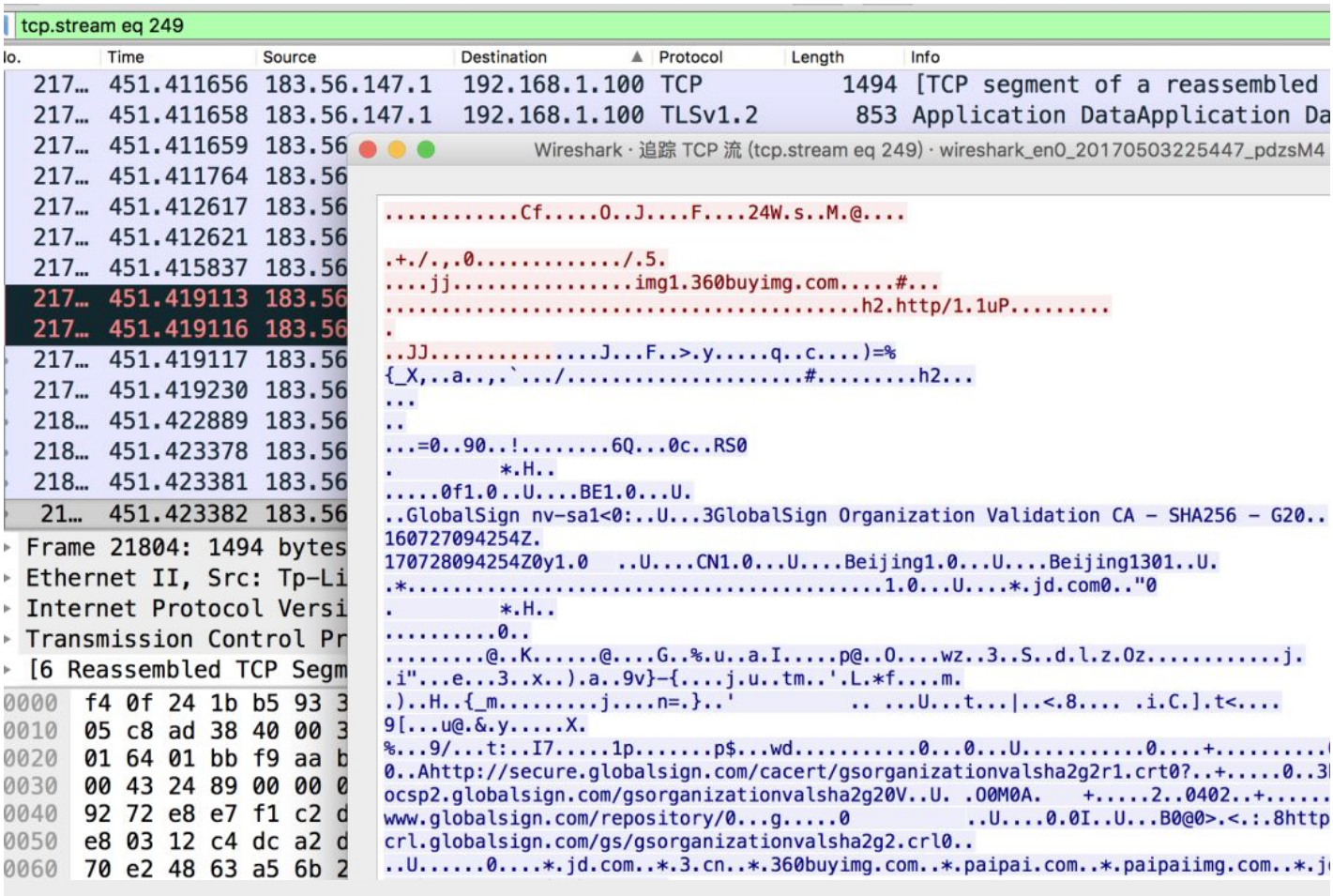
PS：HTTPS本身就是为了网络的传输安全。

例子，使用wireshark抓包：

http，可以看到抓到是明文的：

```
POST /weapi/pl/count?csrf_token=582766250be9b3971eeeb9735392802a HTTP/1.1
Host: music.163.com
Connection: keep-alive
Content-Length: 416
Origin: http://music.163.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/58.0.3029.81 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Referer: http://music.163.com/song?id=133998
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: _ntes_nnid=284a98c1456415dc60bf548361ea011b,1488526571070;
_ntes_nuid=284a98c1456415dc60bf548361ea011b; usertrack=c+5+hli6V0RZkShuCRA2Ag==;
UM_distinctid=15b3d43096a971-0af6f52cb82785-1d3b6853-13c680-15b3d43096b723;
vjuids=2c182bef3.15b3d430d26.0.2b50a5df42d93; vinfo_n_f_l_n3=00cef818465de70f.
1.1.1491381456297.1491884561290.1492263089519; Province=020; City=020;
__gads=ID=4b89a14ba1b8e308:T=1493015318:S=ALNI_Mbww2GCIrZYPZLdXr-j8xqfhH501w;
vjlast=1491381456.1493015316.11; _ga=GA1.2.1631582990.1488607045;
MUSIC_U=23bfc0fd0a003623237fc6ff0a01dd1b8d8b88f57e0ab57214b323f3723363fcd7048b9431c521e9ca35b808
432f07af31b299d667364ed3; __remember_me=true; __csrf=582766250be9b3971eeeb9735392802a;
__utma=94650624.1836862110.1488526571.1493802919.1493818157.119; __utmc=94650624;
__utmz=94650624.1492836565.99.3.utmcsr=baidu|utmccn=(organic)|utmcmd=organic; playerid=74237494;
JSESSIONID-WYYY=mrX%2Bx67QQ%5CmdskwS%5CB9Avo5Ns7gBlozSWCth8lT4Hl3g
%2FTGmudSWwOjzx2ddFWMiUDPQmR0Qk0V%2BGt6XF0ObQeonK%2BUaVFTB1synp4iFqQvBGtu4ADZ6F1rGA7uaIzz
%2FwrlrJwJrT4sQ3q9WyB93YdeB%5CtSku3gF4UCmEhQXVGdDUzsD%3A1493825176657; _iuqxldmzr_=32
```

https，可以看到抓到是密文的：



附录

HTTPS一般使用的加密与HASH算法如下：

非对称加密算法：RSA，DSA/DSS

对称加密算法：AES，RC4，3DES

HASH算法：MD5，SHA1，SHA256

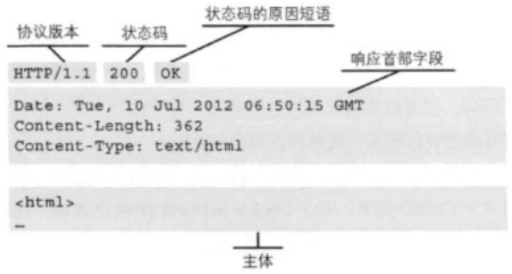
http三次握手：

现在这个社会，我们都离不开网络，那么网络的工作流程是怎么样的呢？从http发起请求到完成请求，网络到底给我们做了什么事情？
今天我们主要来分析下http请求的过程：
在Http工作之前，Web浏览器通过网络和Web服务器建立链连接，该连接是通过Tcp来完成的，该协议和Ip共同组成了Internet，即著名的Tcp/Ip协议族，因此Internet也被称为Tcp/Ip网络，Http是比Tcp更高的应用层协议，一般Tcp接口的端口好是80。

Web浏览器想Web服务器发送请求命令，这个命令中包含：

图：请求报文的构成

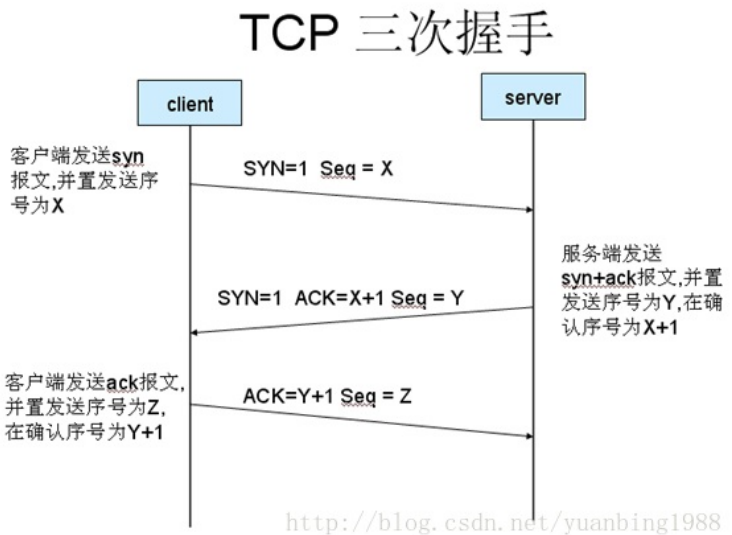
Web服务器发送响应数据给Web浏览器，这个包含：



图：响应报文的构成

然后Web服务器关闭连接。

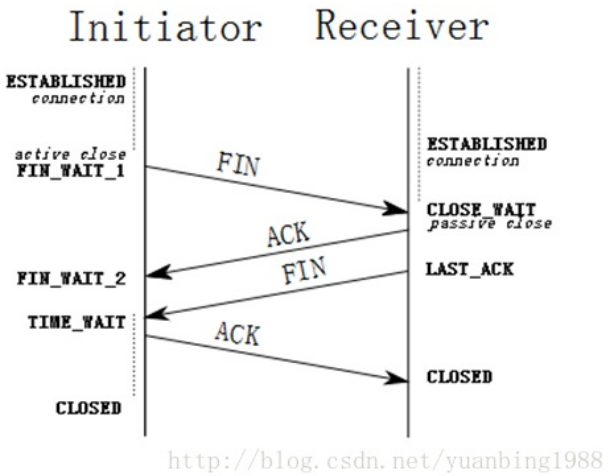
以上就是基本的http请求。
在这个过程中，http建立连接，Tcp经过了3次握手，下面我们来讲讲具体的3次握手的过程，首先我们来看一张图：



1：客户端发送了一个带有SYN的Tcp报文到服务器，这个三次握手中的开始。表示客户端想要和服务端建立连接。
2：服务端接收到客户端的请求，返回客户端报文，这个报文带有SYN和ACK标志，询问客户端是否准备好。
3：客户端再次响应服务端一个ACK，表示我已经准备好。

那么为什么要三次握手呢，有人说两次握手就好了。的确，为什么呢，这个可以从计算机网络中得到答案，举一个例子：已失效的连接请求报文段，client发送了第一个连接的请求报文，但是由于网络不好，这个请求没有立即到达服务端，而是在某个网络节点中滞留了，知道某个时间才到达server，本来这已经是一个失效的报文，但是server端接收到这个请求报文后，还是会想client发出确认的报文，表示同意连接。假如不采用三次握手，那么只要server发出确认，新的建立就连接了，但其实这个请求是失效的请求，client是不会理睬server的确认信息，也不会向服务端发送确认的请求，但是server认为新的连接已经建立起来了，并一直等待client发来数据，这样，server的很多资源就没白白浪费掉了，采用三次握手就是为了防止这种情况的发生，server会因为收不到确认的报文，就知道client并没有建立连接。这就是三次握手的作用。

当终止协议的时候，tcp进行了4次握手，那这4次握手有是怎么回事呢?



由于Tcp连接是进行全双工工作的，因此每个方向都必须单独进行关闭，这个原则是当一方完成他的数据发送的时候就发送一个FIN来终止这个方向的连接，收到这个FIN意味着这个方向上没有数据的流动，一个TCP连接在收到这个FIN之后还能发送消息，首先执行关闭的一方进行主动的关闭，而另一方进行被动的关闭。
1：TCP发送一个FIN，用来关闭客户到服务端的连接。
2：服务端收到这个FIN，他发回一个ACK，确认收到序号为收到序号+1，和SYN一样，一个FIN将占用一个序号。
3：服务端发送一个FIN到客户端，服务端关闭客户端的连接。
4：客户端发送ACK报文确认，并将确认的序号+1，这样关闭完成。

那么为什么是4次挥手呢?
可能有人会有疑问，tcp我握手的时候为何ACK和SYN是一起发送。挥手的时候为什么是分开的时候发送呢，原因是TCP的全双工模式，接收到FIN意味着没有数据发送过来了，但是还可以继续发送数据。

3次握手过程的状态：
listener:这个很好理解，就是服务端的某个socket处于监听状态，可以接收连接了。

syn_send:当某个socket执行connect的时候，首先发送SYN报文，因此也进入了SYN_SEND状态，并等待服务端发送过来的报文，syn_send表示客户端已发送SYN报文。
syn_rcvd:这个状态与SYN_SEND状态差不多，表示接收了SYN报文，这个状态是服务器端的socket在建立tcp连接是的三次握手中的一个中间状态，很短暂，当客户端收到ACK报文的时候，表示连接确立，进入established状态。

4次挥手的状态：
FIN_WAIT_1: 这个状态要好好解释一下，其实FIN_WAIT_1和FIN_WAIT_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别是：FIN_WAIT_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入到FIN_WAIT_1状态。而当对方回应ACK报文后，则进入到FIN_WAIT_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN_WAIT_1状态一般是比较难见到的，而FIN_WAIT_2状态还有时常常可以用netstat看到。（主动方）

FIN_WAIT_2：上面已经详细解释了这种状态，实际上FIN_WAIT_2状态下的SOCKET，表示半连接，也即有一方要求close连接，但另外还告诉对方，我暂时还有点数据需要传送给你(ACK信息)，稍后再关闭连接。（主动方）

TIME_WAIT: 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。如果FIN_WAIT_1状态下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME_WAIT状态，而无须经过FIN_WAIT_2状态。（主动方）

CLOSING（比较少见）: 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送FIN报文后，按理来说是应该先收到（或同时收到）对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况呢？其实细想一下，也不难得出结论：那就是如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。

CLOSE_WAIT: 这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方close一个SOCKET后发送FIN报文给自己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE_WAIT状态。接下来呢，实际上你真正需要考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以close这个
SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE_WAIT状态下，需要完成的事情是等待你去关闭连接。（被动方）

LAST_ACK: 这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也即可以进入到CLOSED可用状态了。（被动方）

CLOSED: 表示连接

##

##

HTTP与TCP/IP区别?

TPC/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据。WEB使用HTTP协议作应用层协议，以封装HTTP 文本信息，然后使用TCP/IP做传输层协议将它发到网络上。

下面的图表试图显示不同的TCP/IP和其他的协议在最初OSI（Open System Interconnect）模型中的位置：

7	**应用层**	HTTP、SMTP、SNMP、FTP、Telnet、SIP、SSH、NFS、RTSP、XMPP、Whois、ENRP
6	**表示层**	XDR、ASN.1、SMB、AFP、NCP
5	**会话层**	ASAP、TLS、SSH、ISO 8327 / CCITT X.225、RPC、NetBIOS、ASP、Winsock、BSD sockets
4	**传输层**	TCP、UDP、RTP、SCTP、SPX、ATP、IL
3	**网络层**	IP、ICMP、IGMP、IPX、BGP、OSPF、RIP、IGRP、EIGRP、ARP、RARP、 X.25
2	**数据链路层**	以太网、令牌环、HDLC、帧中继、ISDN、ATM、IEEE 802.11、FDDI、PPP
1	**物理层**	线路、无线电、光纤、信鸽

PS：表格来自网上资料

CA证书是什么?

CA（Certificate Authority）是负责管理和签发证书的第三方权威机构，是所有行业和公众都信任的、认可的。

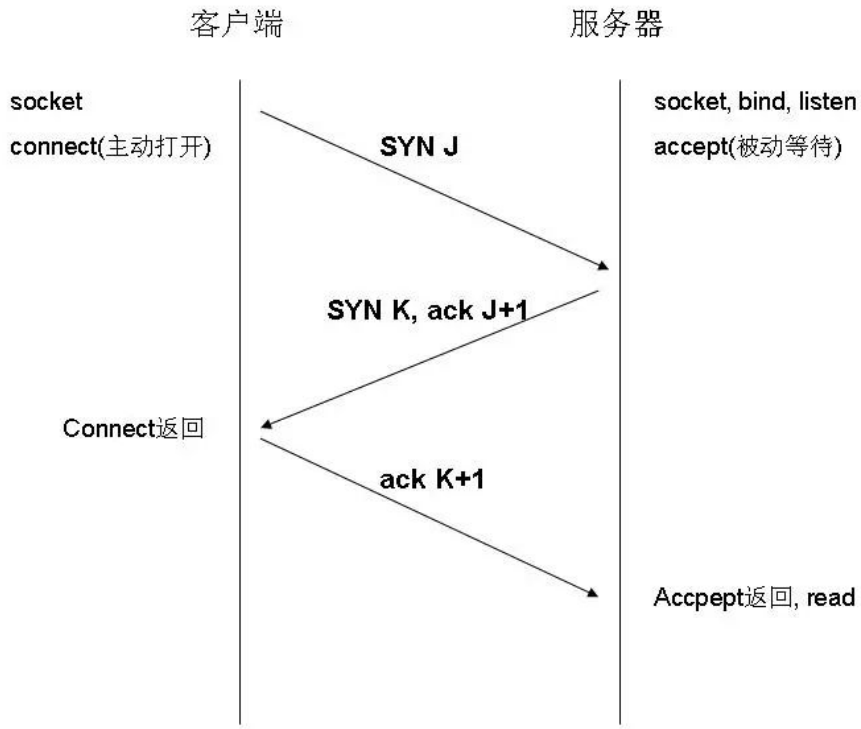
CA证书，就是CA颁发的证书，可用于验证网站是否可信（针对HTTPS）、验证某文件是否可信（是否被篡改）等，也可以用一个证书来证明另一个证书是真实可信，最顶级的证书称为根证书。除了根证书（自己证明自己是可靠），其它证书都要依靠上一级的证书，来证明自己。

HTTP三次握手

HTTP（HyperText Transfer Protocol)超文本传输协议是互联网上应用最为广泛的一种网络协议。由于信息是明文传输，所以被认为是不安全的。而关于HTTP的三次握手，其实就是使用三次TCP握手确认建立一个HTTP连接。

如下图所示，SYN（synchronous）是TCP/IP建立连接时使用的握手信号、Sequence number（序列号）、Acknowledge number（确认号码），三个箭头指向就代表三次握手，完成三次握手，客户端与服务器开始传送数据。

TCP/IP基础——TCP三次握手



PS：图片来自网上资料

第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。