2. 密钥导出

从原则上讲，MS 此时已由 Bob 和 Alice 共享，它能够用作所有后继加密和数据完整性检查的对称会话密钥。然而，对于 Alice 和 Bob 每人而言，使用不同的密码密钥，并且对于加密和完整性检查也使用不同的密钥，通常认为更为安全。因此，Alice 和 Bob 都使用 MS 生成 4 个密钥：

- E_B，用于从 Bob 发送到 Alice 的数据的会话加密密钥
- M_B，用于从 Bob 发送到 Alice 的数据的会话 MAC 密钥
- E_A，用于从 Alice 发送到 Bob 的数据的会话加密密钥
- M_A，用于从 Alice 发送到 Bob 的数据的会话 MAC 密钥

Alice 和 Bob 每人都从 MS 生成 4 个密钥。这能够通过直接将该 MS 分为 4 个密钥来实现。（但在真实的 SSL 中更为复杂一些，我们后面将会看到。）在密钥导出阶段结束时，Alice 和 Bob 都有了 4 个密钥。其中的两个加密密钥将用于加密数据；两个 MAC 密钥将用于验证数据的完整性。

3. 数据传输

既然 Alice 和 Bob 共享相同的 4 个会话密钥（E_B，M_B，E_A 和 M_A），他们就能够经 TCP 连接开始发送安全的数据。因为 TCP 是一种字节流协议，一种自然的方法是用 SSL 在传输中加密应用数据，然后将加密的数据在传输中传给 TCP。但是如果我们真的这样做，我们将用于完整性检查的 MAC 置于何处呢？我们无疑不希望等到 TCP 会话结束时才验证所有 Bob 数据的完整性，Bob 数据的发送要经历整个会话！为了解决这个问题，SSL 将数据流分割成记录，对每个记录附加一个 MAC 用于完整性检查，然后加密该“记录 + MAC”。为了产生这个 MAC，Bob 将数据连同密钥 M_B 放入一个散列函数中，如在 8.3 节所讨论。为了加密“记录 + MAC”这个包，Bob 使用他的会话加密密钥 E_B。然后这个加密的包将传递给 TCP 经因特网传输。

虽然这种方法几经周折，但它为整个报文流提供数据完整性时仍未达到无懈可击。特别是，假定 Trudy 是一名“中间人”，并且有在 Alice 和 Bob 之间发送的 TCP 报文段流中插入、删除和代替报文段的能力。例如，Trudy 能够俘获由 Bob 发送的两个报文段，颠倒这两个报文段的次序，调整 TCP 报文段的序号（这些未被加密），然后将这两个次序翻转的报文段发送给 Alice。假定每个 TCP 报文段正好封装了一个记录，我们现在看看 Alice 将如何处理这些报文段。

1）在 Alice 端运行的 TCP 将认为一切正常，将这两个记录传递给 SSL 子层。

2）在 Alice 端的 SSL 将解密这两个记录。

3）在 Alice 端的 SSL 将使用在每个记录中的 MAC 来验证这两个记录的数据完整性。

4）然后 SSL 将解密的两条记录的字节流传递给应用层；但是 Alice 收到的完整字节流由于记录的颠倒而次序不正确！

鼓励读者观察类似的场景，如当 Trudy 删除报文段或当 Trudy 重放报文段时。

对该问题的解决方案如你可能猜想的那样，那就是使用序号。SSL 采用如下的方式。Bob 维护一个序号计数器，计数器开始为 0，Bob 每发送的一个 SSL 记录它都增加 1。Bob 并不实际在记录中包括一个序号，但当他计算 MAC 时，他把该序号包括在 MAC 的计算中。所以，该 MAC 现在是数据加 MAC 密钥 M_B 加当前序号的散列。Alice 跟踪 Bob 的序