Key exchange with packet loss, delay, and misordering

Mick Seaman

This note discusses the performance and operation of key exchange protocols with reference to the functionality provided by KSP.

Basic protocol

Consider the protocol

$$S_A \rightarrow A, R_A \quad (1)$$
$$S_B \rightarrow B, R_B, A, R_A \quad (2)$$
$$S_{A:}\ K = R_A \oplus R_B \quad (3)^1$$
$$S_A \rightarrow A, R_A, B, R_B \quad (4)$$
$$S_{B:}\ K = R_B \oplus R_A \quad (5)$$

or one of its close relatives.

In this protocol, stations S_A and S_B exchange random numbers R_A and R_B to establish a common pair-wise key, K. If the messages are protected with a master key, and the key has only been entrusted to parties that can be trusted to operate the protocol correctly, then the protocol:

a) proves mutual possession of the master key

b) proves liveness, i.e. the stations possessing the key are operational

c) results in a shared key.

These goals ((a) thru (c)) are met with commendable economy[2]. The stations' identifiers, A and B, are not even required, unless they are used to identify frames as protected by K, and can clearly be omitted when a single key is to be used for a point-to-point link[3]. The real identities of the stations in the protocol are R_A and R_B — though the binding to A and B can facilitate management, S_A thinks of S_B as "the station that has chosen R_B". Replay of protocol messages, with old R_A,R_B values results in a little extra work but not the generation of a competing key, since neither station recognizes its old random value.

Packet loss and delay

R_A and R_B are often referred to as 'nonces', values that are only used once by the protocol, though such a characterization causes careful examination of the meaning of the word "used". If the key exchange protocol is supported by a reliable delivery mechanism, then clearly each protocol message could be transmitted a number of times. This does not violate the security of the protocol, which is just as well, otherwise a simple replay attack could be attempted.

Protection against packet loss is best provided by replaying the messages, rather than by picking new R_A,R_B values. If the latter is done then all three messages have to be successfully received. If the probability of single message loss is p, then the probability succeeding without any retransmission is $(1 - p)^3$. For $p = 0.3$, that is 0.34.

Worse, if new R_A,R_B values are chosen and the problem is excessive delay rather than loss, then the protocol may never succeed. S_A may retransmit step (1) of the protocol just before receiving from S_B at step (3) with the original values. S_A will have discarded its first nonce, R_{A1} say, in favor of R_{A2} so it knows to discard S_B's response, but it still has hope that the latter attempt experiences no loss. Retransmission using fresh nonces should therefore use a backoff for retransmission intervals[4], or conservative delay estimates. A reasonable measure of protocol performance is the expected time for a 99.99% probability of completing key exchange. Neither retransmission strategy approaches the performance achieved by more sophisticated protocols if nonces are changed once every n transmissions and $p^n \gg 0.01\%$.

Misordering

The effect of misordering in the presence of nonce changes is to both increase the time that can be taken for the protocol to succeed – since success can be undone by receipt of a previously transmitted message – and, when coupled with loss, to introduce the possibility that one station believes the protocol to have

1 The symbol ⊕ denotes 'exclusive-or'

2 The other side of the argument is that supporting a number of distinct functions with fewer protocol elements is just being 'clever', an activity that leads to obscurantism and design failure when there is the least extension of goals. However this is a cheap shot since there is no unique functional decomposition of any interesting problem.

3 And from step (1) of the protocol in any case.

4 It is apparent that we are slipping toward an explicit transport protocol here, and other issues will begin to detract from the simplicity of picking transaction identifiers that also generate the key.

been successfully completed but the other has a key based on the previous nonce.

In fact S_B may retain data for protocol execution separately for each peer nonce, in our example separately for R_{A1} and R_{A2}, but if both these are bound to identifier A then failure will have occurred.

Obviously if one nonce is changed and the other is not, an attacker can replay old messages to induce a denial of service attack[5].

Timeliness

The ability of either station to draw conclusions about when the other replied to a message, and thus to defend the key exchange and the use of the subsequently derived key against an attack that simply introduces a mischievous delay to compromise the operation of configuration protocols, is unfortunately lessened if the nonce are not changed frequently. Thus the protocols user's desires in this regard run contrary to his interests for timely success in the face of loss and misordering.

Enhanced Protocol

The basic protocol can be simply enhanced with a message number or counters. This is referred to as an "age" in descriptions of KSP, but the term count and the symbol C are used here to avoid any possibly adverse connotations of time, and to avoid overuse of "number" and "N" in context where other semantics for nonces may be appropriate.

The count allows messages to be sequenced, protecting against misordering, and also allows stations to measure timeliness without requiring the nonce R to be changed. R can now be chosen infrequently. The sense in which it is "used once" is that a new value is randomly chosen from a very large space whenever the complete history of values derived from a previous use is not known to the station choosing R, or a new R has to be picked for any other reason.

The protocol is[6]:

$$S_A \rightarrow A, R_A, C_A \qquad (1)$$

$$S_B \rightarrow B, R_B, C_B, A, R_A, C_A \qquad (2)$$

$$S_{A:}\ K = R_A \oplus R_B \qquad (3)$$

$$S_A \rightarrow A, R_A, C_{A+}, B, R_B, C_B \qquad (4)$$

$$S_{B:}\ K = R_B \oplus R_A \qquad (5)$$

where C_A and C_B are set to zero (or one) on their first use, and incremented for every subsequent message, and C_{A+} is a value greater than or equal to C_A.

[5] While no protocol can prevent denial of service by an attacker who has full control over the transmission medium, that does not mean to say that attacks that involve loss, replay, or misordering of a few messages and that have an indeterminately persistent effect should be admitted.

[6] Note again that each message is protected using a master key

The count values, C_A and C_B, do not wrap but a fresh R is chosen by the respective station when necessary. For the modest message rates[7] of a key exchange protocol this will be very infrequent, even for a modest sized field, so does not much affect expected performance.

Since the values C_A and C_B are parroted back to their sources (S_A and S_B) the latter can use them to ensure timeliness. Their inclusion does not affect the correctness of the basic protocol, when it succeeds, since their use simply results in the discard of aged and out of order messages — and is thus equivalent to a source of packet loss. Thus C_A and C_B do not have to be incremented for literally every message, but only as timeliness guarantees require — once every half second for example. Alternately they can reflect the value of a local timer, ticking perhaps in milliseconds, just so long as the timer is only reset when a new R is chosen.

Key Transport

As described above, the values R and their exchange in messages encrypted under a master key actually serve three purposes:

a) mutual authentication, or rather proof of the result of a prior authentication process, through proof of mutual possession of the master key

b) proof of liveness, i.e. the stations possessing the key are operational and engaging in the exchange

c) establishment of shared key.

The addition of the count (C) values allows the liveness guarantees provided in (b) above to be enhanced without frequently changing R.

Since both parties contribute to the shared key in (c), and do so using nonces, the accidental repetition of a derived key K is protected against. However the nonce value used for (c) does not necessarily have to be the same as that used for (a) and (b), and in that respect the functionality of R is overloaded, even if the overloading is convenient from the point of view of proving security.

As a first step in separating functions without invalidating prior proofs consider each nonce to be composed of two separate parts, R and K, generated and disposed of together. The R part supports functions (a) and (b), and the K supports (c).

To make the derivation and use of keys clear in the following protocol descriptions, a key K derived from combining values R_A and R_B is written as K_{RARB}, and the notation $\{...\}_{KRARB}$ is used to denote encryption and integrity protection of a message ... using that key with a suitable cryptographic mode and a random initial value/nonce.

[7] A few messages per second.

The original basic protocol can be written as:

$S_A \rightarrow \{A, R_A\}_M$ (1)

$S_B \rightarrow \{B, R_B, A, R_A\}_M$ (2)

$S_{A:}\ K_{RARB} = R_A \oplus R_B$ (3)

$S_A \rightarrow \{A, R_A, B, R_B\}_M$ (4)

$S_{B:}\ K_{RARB} = R_B \oplus R_A$ (5)

Separating out the R and K parts, as described above, we have:

$S_A \rightarrow \{A, R_A, K_A\}_M$ (1)

$S_B \rightarrow \{B, R_B, K_B, A, R_A, K_A\}_M$ (2)

$S_{A:}\ K_{KAKB} = K_A \oplus K_B$ (3)

$S_A \rightarrow \{A, R_A, K_A, B, R_B, K_B\}_M$ (4)

$S_{B:}\ K_{KAKB} = K_B \oplus K_A$ (5)

Of course an equally valid way to achieve the result shown by this separation of concerns would be first to generate the key K_{RARB} using either the basic protocol or the enhanced version (with C_A and C_B) described above, and then use the resulting secure channel to exchange K_A and K_B :

$S_B \rightarrow \{B, K_B, A, K_A\}_{KRARB}$ (6)

$S_{A:}\ K_{KAKB} = K_A \oplus K_B$ (7)

$S_A \rightarrow \{A, K_A, B, K_B\}_{KRARB}$ (8)

$S_{B:}\ K_{KAKB} = K_B \oplus K_A$ (9)

Which makes it apparent that although K_A and K_B are nonces, just as R_A and R_B are, that the Ks can be chosen independently of the Rs. It also makes it clear that the messages

$\{A, R_A, K_A, B, R_B, K_B\}_M$ (i)

and

$\{A, K_A, B, K_B\}_{KRARB}$ (ii)

and indeed

$\{R_A, R_{AB}, \ldots\}_M$ (iii)

and

$\{\ldots\}_{KRARB}$ (iv)

are equivalent, up to the point that there is a risk of key M having been used too many times, so either form ((iii) or (iv)) can be used to transport keying material for a data connection, with (iii) having the advantage that it is no necessary to calculate tables and setup the use of the "intermediate" key K_{RARB}. Moreover use of this form allows a single set of C values to provide ordering and timeliness protection for exchange of both R and K values.

The final form of the protocol is thus:

$S_A \rightarrow \{A, R_A, C_A, K_A\}_M$ (1)

$S_B \rightarrow \{B, R_B, C_B, K_B, A, R_A, K_A\}_M$ (2)

$S_{A:}\ K_{KAKB} = K_A \oplus K_B$ (3)

$S_A \rightarrow \{A, R_A, C_{A+}, K_A, B, R_B, C_B, K_B\}_M$ (4)

$S_{B:}\ K_{KAKB} = K_B \oplus K_A$ (5)

with discussion of alternative forms of keying material (K values) and their use being separable from the basic mechanism that transports those values.