CS 181S – System Security Fall 2018

Lecture 9: Protocols

October 3, 2018 *Instructor: Eleanor Birrell*

1 Review of Symmetric Cryptography

Shared key cryptography (also known as symmetric key cryptography) involves functions Enc(encryption) and Dec(decryption):

$$\begin{aligned}\text{Enc} &: k \times m \rightarrow c\\ \text{Dec} &: k \times c \rightarrow m\end{aligned}$$

These functions are inverses, so that $m = \text{Dec}(\text{Enc}(m; k); k)$ holds. In these notes, we use the following notation:

We assume a strong attacker so that our protocols will resist whatever might be encountered in the "real world". The Dolev-Yao threat model has become a standard for this and formalizes the following attacks.

- An attacker can intercept messages transmitted between hosts.
- An attacker can parse any message comprising separately encrypted fields, and the attacker can extract the separate fields.
- An attacker can decrypt a message or field if and only if that attacker has previously obtained the appropriate encryption key.
- An attacker can construct and send new messages using information intercepted from old messages.
- The first three elements characterize passive attacks; the fourth element exemplifies an active attack.

2 Uses of Shared Key Cryptography

When cryptography is first mentioned, most people think about implementing confidentiality. If A and B share a key k, then one can send a message to the other, keeping the contents of that message secret from the adversary.

This basic approach also works for storing confidential information on a device, like a disk. Think of the device as B; only encrypted information is written to B. Note, the fact that information stored on B is encrypted might not suffice for keeping that information confidential. A text editor, for example, will typically work by copying any files it opens from long term storage (i.e., disk B) into CPU main memory; manipulation of file contents are then done in the CPU main memory. The file in CPU main memory is unencrypted (possibly providing an attacker with one access point). Moreover, to ensure that file updates are not lost, an editor is likely to store the current state of the object being edited in a temporary file (i.e., on disk B), and that means confidential information might temporarily reside unencrypted on the disk too, despite our best intentions. In fact, file deletion often leaves the file contents on a disk, so the confidential information might remain available in unencrypted form on the disk long after the editing session has ended.

3 Authentication using Shared Key Cryptography

An authentication protocol allows a principal receiving a message to determine which principal sent that message. Any afficionado of spy movies is doubtless familiar with one means by which principals might authenticate each other: The first principal asks an innocuous question — called the challenge — of the second ("Its a bright sunny day in New York."). And if the first principal knows the secret response to that challenge ("But I prefer rain.") then the first concludes that the second is a compatriot.

This authentication protocol has a significant weakness. Anyone overhearing the exchange is thereafter able to impersonate either side. The basic idea—using knowledge of a secret—is sound. The problem is that a principal must reveal the secret in order to prove knowledge of that secret. This is known as weak authentication, and it is subject to replay attacks whereby an adversary repeats fragments of a past protocol run, appearing to have knowledge of the secret, and thus passes the authentication test.

In a strong authentication protocol, knowledge of the secret is demonstrated without revealing the secret itself. If, for example, the secret is used in determining the answer for the challenge, then an adversary overhearing the challenge and response is not going to be able to feign knowledge of the secret when confronted with other, different challenges.

One way to implement strong authentication in a network starts from the assumption that principals share a secret key. Knowledge of that key is demonstrated by using the key to encrypt and decrypt challenges, but the key itself is never revealed. Here is a protocol that allows B to authenticate A. Assume that A and B are the only two principals with knowledge of secret key k.

1. B: select and store a new random value r.
2. B $\rightarrow$ A: B,r
3. A $\rightarrow$ B: Enc(r;k)
4. B: check whether $\text{Dec}(m_3) = r$, where m_3 is the message received in step 3 and r is the stored value from step 1.

Random value r selected in step 1 is called a nonce. It is (by design) new, so an attacker who has recorded previous versions of message 3 is unable to replay one of those messages in order to satisfy this new challenge issued by B. Inclusion of "B" in step 2 allows receiver A to select the correct shared key (i.e., the one shared with B) for generating message 3. An attacker seeing message 2 bearing the challenge does not know secret key k and thus should not be able to generate a response that satisfies B (in step 4) for the challenge (issued in step 2).

Reflection Attacks. An attacker M can compromise the previous protocol, because A and B are each unwittingly running an "encryption service". In particular, A assumes any challenge it receives (message 2) is from an honest principal, and A always encrypts such challenges to produce responses. With a reflection attack, an intruder sends information from an on-going protocol execution back to the originator of that information. For example, an intruder might exploit an "encryption service" being run by both A and B and fool one of the participants into generating responses for its own challenges. To fool a participant, the intruder runs one or more concurrent instances of the protocol and interleaves them with the original. Because participants in protocol instances execute each protocol instance independently (and therefore no protocol instance inspects the state of any other concurrent instances, since doing that correlation would be expensive), a participant won't realize that it is generating responses for its own challenges.

Here's an example of a reflection attack. Indentation (and roman-numeral steps) indicate the concurrent instance of the protocol that M is running.

1. B: select and store a new random value r.
2. B $\rightarrow$ M: B,r
    - i. M $\rightarrow$ B: A,r
    - ii. B $\rightarrow$ M: Enc(r;k)
3. M $\rightarrow$ B: Enc(r;k)
4. B: check whether Dec(m_3) $= r$

In step 2, M has intercepted the message to A and learned the challenge (r). Then, in step i. M starts a second, concurrent instance of the protocol, impersonating A trying to authenticate B. In this second instance of the protocol, B does what it always does upon receiving a challenge—it encrypts the challenge using the correct key and replies (in step ii). But that reply provides what M needs for responding to the challenge posed by B in message 2 in order to convince B that M knows the shared key k with A and thus to convince B that M is A. Reflection attacks can often be prevented by breaking protocol symmetry and having messages somehow distinguish the different roles participants play. In the attack given above, B serves in the first protocol instance as the initiator and in the second instance as the responder. Here are some alternatives for breaking the symmetry:

Have A and B share two keys: k_AB and k_BA. Key k_AB is used by B when A is the initiator; key k_BA is used by A when B is the initiator. A will never send a message encrypted under k_AB, and B will never send a message encrypted under k_BA; thus a message encrypted under k_AB must have been sent by B. Let's revisit the attack.

1. B: select and store a new random value r.

2. B → M: B,r

   i M → B: A,r

   ii. B → M: Enc(r; k_AB)

3. M → B: ???? M sending Enc(r;k_AB) doesn't suffice, since Enc(r;k_BA) is awaited by B

OR

Insist that each response includes the identity of the responder, so the authentication protocol becomes:

1. B: select and store a new random value r.

2. B → A: B,r

3. A → B: Enc(A,r; k)

4. B: check whether D(Enc(A,r;k);k) equals "A,v" where v is the stored value r from step 1.

And the reflection attack is again foiled:

1. B: select and store a new random value r.

2. B → M: B,r

   i M → B: A,r

   ii. B → M: Enc(B,r; k)

3. M → B: ????? Sending Enc(B,r;k) doesn't suffice, since Enc(A,r;k) is awaited by B.

Man-in-the-Middle Attacks. Nothing restricts an attacker M to using the "encryption service" that is provided by only a single protocol participant. Indeed, even with the defenses just outlined, an attacker M could run two instances of the protocol: one with A and the other with B, engaging as needed whichever of these principals would produce the value being needed by M to perpetuate its deception.

Mhe general form of this attack is simple to understand. Each protocol step i of the form

i. X $\rightarrow$ Y: m

is replaced by two steps

i. X $\rightarrow$ M: m

i' M $\rightarrow$ Y: m

Perhaps, in retrospect, it is not surprising that such an attack is always possible. After all, M is indistinguishable from a wire or a network channel (which itself might involve multiple store-and-forward routers).

We cannot eliminate man-in-the middle attacks, but we can blunt their effectiveness. If all traffic is encrypted using a key shared only by the endpoints, then an intruder in the middle cannot read or alter messages. There is now little to be gained from the attack. We now have two compelling reasons for principals to share keys. First, shared keys can be used to implement string authentication. Second, shared keys help in defending against man-in-the-middle attacks. The obvious question, then, is how do hosts come to share those keys.

4 Key Distribution Protocols

With N hosts, O(N**2) shared keys are needed for each pair of hosts to have a distinct shared key. This is a large number of keys if N is large. Moreover, relatively few of those shared keys would ever actually be used, since any given host is likely to communicate only with a small subset of the hosts. So, instead, a mediated key exchange protocol is often used. Each host shares a key with some trusted host KDC (for Key Distribution Center), and KDC generates keys, on demand, for pairs of hosts that must communicate. Thus, starting from a relatively small number of shared keys we generate all the rest.

Assume each principal P shares key K_P with KDC. An obvious protocol for principal A to obtain a fresh secret key K_AB for communication with a principal B is:

1. A $\rightarrow$ KDC: A,B
2. KDC $\rightarrow$ A: A,B, Enc(K_AB;K_A)
3. KDC $\rightarrow$ B: A,B, Enc(K_AB;K_B)

However, this protocol is not without vulnerabilities. Also it has an engineering flaw: A does not know whether B has received the key, and it is possible that a K_AB-encrypted message from A would reach B before K_AB does. (Message 3 from KDC is likely to be traveling a different route than a K_AB-encrypted message from A would.) The engineering flaw is addressed by having A be responsible for forwarding K_AB to B rather than having KDC do this.

1. A $\rightarrow$ KDC: A,B
2. KDC $\rightarrow$ A: A,B, Enc(K_AB;K_A), Enc(K_AB;K_B)
3. A $\rightarrow$ B: A,B, Enc(K_AB;K_B)

We now turn to vulnerabilities in the protocol. An intruder M that can impersonate B, could obtain a key and position itself to read all traffic between A and B. This key is obtained by M using a man-in-the-middle attack at step 1 and starting a second, independent instance of the authentication after step 2.

1. A $\rightarrow$ M: A,B

1'. M $\rightarrow$ KDC: A,M

2. KDC $\rightarrow$ M: A,M, Enc(K_AM;K_A), Enc(K_AM;K_M)

1".M $\rightarrow$ KDC: M,B

2'.KDC $\rightarrow$ M: M,B Enc(K_MB;K_M), Enc(K_MB;K_B)

2" M $\rightarrow$ A: A,B, Enc(K_AM;K_A), Enc(K_MB;K_B)

3. A $\rightarrow$ B: A,B, Enc(K_MB;K_B)

This man-in-the-middle attack is possible because message 2 contains fields, so according to Dolev-Yao M can extract then mix-and-match in order to construct bogus message 2' that M ultimately sends to A. We can rule out such a parsing by M of message 2 simply by encrypting the entire message 2. The message must be intelligible by A, so it suffices to encrypt using K_A. Here is the revised protocol:

1. A $\rightarrow$ KDC: A,B
2. KDC $\rightarrow$ A: Enc(A,B, K_AB, Enc(K_AB;K_B) ;K_A)
3. A $\rightarrow$ B: A,B, Enc(K_AB;K_B)

The same vulnerability exists with message 3. Here is the attack: M impersonates A in message 3, by first contacting KDC (as itself, M) in order to get a key K_MB that can be foisted upon B as if it were K_AB.

1. M $\rightarrow$ KDC: M,B
2. KDC $\rightarrow$ M: Enc(M,B, K_MB, Enc(K_MB;K_B); K_M)
3. M $\rightarrow$ B: A,B, Enc(K_MB;K_B)

The defense here is to include the names A and B in the final field encrypted under K_A of message 2. This prevents message 3 (which contains that final field as its final field) from being misinterpreted by B as having come from A.

1. A $\rightarrow$ KDC: A,B
2. KDC $\rightarrow$ A: Enc(A,B, K_AB, Enc(A,B,K_AB;K_B); K_A)
3. A $\rightarrow$ B: A,B,Enc(A,B,K_AB; K_B)

The next vulnerability to note is that message 2 can be replayed by an attacker having intercepted message 1, forcing A and B to use an old key K_AB. Any attacker that had learned an old value of K_AB has the incentive to make this happen. We defend against A being fooled into using an old message 2 by including a nonce in message 1, which must be returned in message 2.

1. A $\rightarrow$ KDC: A,B,r where r is a new random value
2. KDC $\rightarrow$ A: Enc(A,B,r, K_AB, Enc(A,B,K_AB; K_B) ; K_A)
3. A $\rightarrow$ B: A,B, Enc(A,B,K_AB; K_B)

We might defend against B being fooled into using an old value of K_AB (resulting, for example, from a replay of an old message 3) by adding a challenge-response round to the end of the protocol, obtaining a protocol originally proposed by Roger Needham and Mike Schroeder (see "Using encryption for authentication in large networks of computers" Communications of the ACM 21 (1978), 993-999). It is still used today, and is the basis, for example, of the popular Kerberos system: Needham-Schroeder Protocol:

1. A $\rightarrow$ KDC: A,B,r where r is a new random value
2. KDC $\rightarrow$ A: Enc(A,B,r, K_AB, Enc(A,B,K_AB; K_B); K_A)
3. A $\rightarrow$ B: A,B, Enc(A,B,K_AB; K_B)
4. B $\rightarrow$ A: Enc(r'; K_AB) where r' is a new random value
5. A $\rightarrow$ B: Enc(r' + 1; K_AB)

However, the actual utility of messages 4 and 5 is somewhat subtle. There are two cases to consider.

Suppose the attacker replays an (old) message 3—a message containing an old value of K_AB known to the attacker. But in this case, the attacker would be able to respond to challenge 4, so B is fooled despite adding the challenge-response protocol (messages 4 and 5). (In Kerberos this problem is minimized by including a timestamp in the final field of message 2. B rejects a message 3 that contains a sufficiently old timestamp.) What if the attacker does not know the old value of K_AB that is contained in the old message 3 being replayed? Here, the challenge will not succeed, but the attacker has caused delay and network traffic. On a positive note, B does learn that a shared key has not been established. Reality Intrudes: Keys do get compromised

Keys are sometimes compromised. One would hope that once such a compromise is detected, new keys could be selected and subsequent mischief derailed. Unfortunately, it isn't that simple. To start, consider the case where K_AB becomes known to attacker M. Suppose further, that M had the forethought to have saved message 3

3. A $\rightarrow$ B: A,B, Enc(A,B,K_AB; K_B)

from the Needham-Schroeder run in which B was informed of this K_AB value. M can now force B to use K_AB again, as follows: At the next execution of Needham-Schroeder, M blocks transmission of message 3 to B and instead forwards the saved version of that message (i.e., the message containing the old K_AB). M then intercepts message 4 and impersonates A in step 5 (using this old, compromised K_AB to generate Enc(r' + 1; K_AB)). So, the result is that B will share an old K_AB with a process (M) that it believes is A.

The Otway-Rees authentication protocol avoids this vulnerability by extending Needham-Schroeder with a nonce n for the entire run. The protocol appears below. Notice:

The communications topology is a bit different than in Needham-Schroeder. With Otway-Rees, there is a nested structure (A; B; KDC; B; A) whereas with Needham-Schroeder, A is more of a hub, communicating separately with KDC and with B. KDC can check message 2 to ensure that nonces r1 and r2 are being associated with the same protocol run nonce n, and (because KDC is trusted) send message 3 only if that equality holds.

1. A $\rightarrow$ B: n,A,B, Enc(r1,n,A,B; K_A)

2. B $\rightarrow$ KDC: n,A,B, Enc(r1,n,A,B; K_A), Enc(r2,n,A,B; K_B)

3. KDC $\rightarrow$ B: n,Enc(r1,K_AB;K_A), Enc(r2,K_AB; K_B),

4. B $\rightarrow$ A: n,Enc(r1,K_AB; K_A)

Both A and B can be certain that shared key K_AB is new, because the response each receives contains a nonce that was included in that principal's most recent request. This works since KDC checks that the request from A (Enc(r1,n,A,B;K_A)) and the request from B (Enc(r2,n,A,B;K_B)) are for the same protocol run by checking that the same value for n appears in the two requests.

5 Type Attacks

A somewhat disturbing attack is possible against a protocol implementation unless care is taken with exactly how values are sent. Messages are just strings of bits, with each field in a message some substring of the whole. Moreover, Dolev-Yao allows an attacker to replace one field by the value found in another. So consider the following attack against an Otway-Rees protocol implementation. It assumes decryption just happens to produce outputs with particular lengths and correspondences. Here is the attack.

- M intercepts Otway-Rees message 1, and extracts substrings ”n,A,B” and ”Enc(r1,n,A,B; K_A)”.
- M blocks Otway-Rees protocol message 2 (which means message 3 won't be sent either).
- M then sends back to A as Otway-Rees protocol message 4 the following (which is constructed from information learned by M in step i of this attack): 4. M $\rightarrow$ A: n,Enc(r1,n,A,B; K_A)
- A will decrypt Enc(r1,n,A,B;K_A) believing this bitstring to be Enc(r1,K_AB; K_A). According to the correspondence we assumed above, A will conclude that K_AB equals ”n,A,B”.

But M knows ”n,A,B” from step i of the attack, so M has forced A to accept as a key K_AB shared with B what in fact is a key shared with M Couple this type attack with a reflection attack, and it now becomes possible for M to induce both A and B to use ”n,A,B” as their shared key K_AB. Here's that attack.

- M records Otway-Rees message 1, and extracts substrings ”n,A,B” and ”Enc(r1,n,A,B;K_A)”.
- M intercepts and blocks Otway-Rees message 2. M records ”Enc(r2,n,A,B;K_B)”
- For Otway-Rees message 3, M uses the two bit strings it has already copied: 3. M $\rightarrow$ B: n,Enc(r1,n,A,B; K_A), Enc(r2,n,A,B;K_B)
- Because of the correspondences we assume, B will decrypt final field ”Enc(r2,n,A,B; K_B)” assuming it was ”Enc(r2,K_AB; K_B)” and get as new key K_AB the bit string ”n,A,B” which is known to M. B will forward to A as Otway Rees message 4 the first field in the message it just received from M 4. B $\rightarrow$ A: n,Enc(r1,n,A,B; K_A) but, again, because of the correspondences we assume, A will decrypt ”Enc(r1,n,A,B; K_A)” assuming it was ”Enc(r1,K_AB; K_A)” and get as new key K_AB the bit string ”n,A,B”.

Obviously, type attacks become impossible when this sort of mix-and-match substitution of different kinds of values is impossible. Use of a programming notation in which messages contain typing information is one way to avoid the problem.

6 Back to Needham-Schroeder...

We explored above the consequences that a key K_AB might be compromised. This led from Needham-Schroeder to Otway-Rees, and with Otway-Rees we took a short detour to understand type attacks. Let us now return to Needham-Schroeder—this time to investigate the consequences of having the key K_A that a principal A shares with KDC compromised. Presumably the compromise of K_A is eventually detected and A adopts a new value, expecting everyone to live happily ever after. Unfortunately, an attacker M can still foist a bad key on an unsuspecting principal B with which A is attempting to establish a shared key. Here's how that attack works. (You will find it helpful to refer back to the final version above for Needham-Schroeder.)

Suppose M has intercepted message 2 from an earlier run of Needham-Schroeder where intercepted key K_A was in use. This means that M can extract the value of shared key K_AB being proposed in that message and also can extract "Enc(A,B,K_AB; K_B)". M now waits for A to start a new execution of Needham-Schroeder to establish a shared key with B. M replaces message 3 in that execution with a message it constructs using "Enc(A,B,K_AB; K_B)" that M extracted in step i and that includes old key K_AB. Because M knows this old value of K_AB (from step i), M can respond correctly to the challenge that B then sends in Needham-Schroeder message 4. So B will be convinced that this old value of K_AB is the new key it shares with A. The vulnerability actually permits a less passive attack: M need not have intercepted a previous message 2, instead forging a message 1 to receive a message 2 for every principal P). This leaves M free to then initiate a connection to any principal P, impersonating A, until such time that every other principal's key has been changed. So a single compromised key ultimately would require changing all KDC keys!

The defense against this attack is to have B create a nonce r" so that the step ii replay of the old Needham-Schroeder message 3 can be detected by B. This is achieved by prepending to Needham-Schroeder step 1 a protocol (labeled steps a and b) whereby B selects r" and sends it to A for forwarding to KDC, where it is used in forming message 2 (the contents of which are later used in building message 3).

a. A $\rightarrow$ B: A,B

b. B $\rightarrow$ A: A,B,r"

1. A $\rightarrow$ KDC: A,B,r,r"

2. KDC $\rightarrow$ A: Enc(A,B,r, K_AB, Enc(A,B,K_AB,r"; K_B); K_A)

3. A $\rightarrow$ B: A,B, Enc(A,B,K_AB,r"; K_B)

4. B $\rightarrow$ A: Enc(r'; K_AB) where r' is a new random value
5. A $\rightarrow$ B: Enc(r' + 1; K_AB)

B can now reject an old message 3, because the nonce r" contained in an old message 3 will not be the same as the nonce it generated in step b of this protocol run.

Acknowledgements

Based on a lecture by Fred Schneider and on lecture notes by Benyah Shaparenko, Gregory Roth, and Fred Schneider.