Analysis of Protocols

Sophie Stevens

September 27, 2024

1 Introduction

In this lecture we're going to look at how to begin analysing key exchange protocols. A lot of the cryptography that you've already looked at is about cryptographic primitives, for example RSA, Diffie-Hellman exchange, AES, and their security. Protocol analysis is about making sure that we use these primitives properly to set up secure communications, by which we really mean key establishment. Once the key has been established, it is used to create a secure communications channel. We'll typically look at two-party key establishment, where two users set up a secure channel, possibly with the use of a third party.

At the end of this lecture, I'd like you to have some idea of how protocols are designed, what security goals they might aim to achieve, and what framework we might use to analyse protocols. In particular, we'll be looking at authenticated key establishment protocols, where the users not only establish a key, but they also want to be convinced that the user with whom they've agreed a key is actually the user that they expect.

2 Ongoing example

In our ongoing example we're going to evolve a protocol design. Our goal is for our two canonical users, Alice and Bob, to established a shared secret key using a server that they both trust. They want this key to be known only to them (and maybe the server) and they want this key to be fresh. *Fresh* in this context means that the key was first established on that run of the protocol – in particular, it's not a key that's been reused from a previous run. There are a few minimal assumptions that we have to make: firstly, that all parties know how the protocol should work – this means that they expect to receive and send messages at the same time; if for example, they receive no message when they are expecting one, they'll assume that the adversary has blocked the message, and will abort the process. Secondly, we're generally not going be that interested in the workings of the primitives themselves, and we'll assume they work perfectly. For example, we'll assume that an encryption algorithm is unbreakable unless you know the key. In reality of course this isn't true - we can always guess the key by brute force, and many encryption algorithms admit

better algebraic attacks (that are of course impractical in practice). Within this assumption, we also ignore side channel attacks.

2.1 Attempt 1

Let's begin: Alice (A) is going to tell the server (S) her own ID, and Bob's ID (ID_A, ID_B); the server will generate a key (K_AB) and sends it to Alice; Alice sends the key to Bob (B) as well as her own ID, so that he knows who this key is to be shared with.

We will summarise this exchange as follows:

```
Key Exchange v1:
    1. A --> S: ID_A, ID_B
    2. S --> A: K_AB
    3. A --> B: K_AB, ID_A
```

Now we ask the question: is this a good protocol? Although it does achieve the aim that Alice and Bob share a key, this key has been sent in the clear, which means that any adversary also knows the key. Conclusion: not a good protocol.

Implicitly, we've just given the adversary the power to read any message that is transmitted. In reality, this is a reasonable assumption to make - for example, Wireshark is a free open-source software that captures network traffic.

2.2 Attempt 2

We'll improve our protocol now by including encryption. We need to make some assumptions here, namely that Alice and Bob each share a key with the server and that this key is secret from the adversary. We denote the key shared between Alice and the server as K_AS, and the key between Bob and the server as K_BS; implicitly we assume that these are good keys: the adversary does not know them and they are different.

We write the encryption of a message m by key K as $\{m\}_K$.

Our new protocol is now:

Key Exchange v2:

1. A --> S: ID_A, ID_B
2. S --> A: $\{K\_AB\}_{K\_AS}$, $\{K\_AB\}_{K\_BS}$
3. A --> B: $\{K\_AB\}_{K\_BS}$, ID_A

Again, we ask the question: is this a good protocol? Certainly it's better than before, because the adversary no loner knows K_AB, by dint of the key being sent in an encrypted format and our assumption of 'perfect cryptographic primitives'. However, this doesn't mean that the adversary is powerless.

One modification that the adversary can do is to impersonate Alice in Step 3 and send Bob instead: $\{K\_AB\}_{K\_BS}$, ID_D. Consequently, Bob believes that his key is to be shared with User D, rendering the key K_AB useless.

However there's a more dangerous attack that can occur:

```
Attack on Key Exchange v2:
    1. C --> S: ID_C, ID_A
    2. S --> C: {K_AC}_K_AS, {K_AC}_K_CS

    1'. A --> S: ID_A, ID_B
    2'. C --> A: {K_AC}_K_AS, {K_AC}_K_CS
```

Firstly, the adversary, C, is a legitimate user of the protocol and so can send a key generation request to the server, who will reply as expected. Then, when A sends an initiation request, User C impersonates the server, returning a key that C knows, and also knows the encryption of under the key K_AS. Thus A has a key that she believes she shares with B, but she actually shares with C; consequently C can read any message intended for B.

This attack also violates the demand for freshness of a key – a key from one run of the protocol (namely Steps 1 and 2) is being reused in another run of the protocol (Steps 1' and 2'): there is nothing to bind the key to the identity or to the protocol run.

What wider lessons can we learn here? Firstly, we let the adversary modify messages at will; these can be partial or entire messages. We need to consider not only the confidentiality of messages, but also the message integrity. Secondly, the adversary can be a legitimate user of the protocol.

2.3 Attempt 3

Let's fix the issue of the key not being bound to the identity:

Our new protocol is now:

```
Key Exchange v3:
    1. A --> S: ID_A, ID_B
    2. S --> A: {K_AB, ID_B}_K_AS, {K_AB, ID_A}_K_BS
    3. A --> B: {K_AB, ID_A}_K_BS
```

Note that we are assuming integrity of the encryption algorithm - that encrypted data cannot be manipulated.

Again, we ask the question: is this a good protocol?

At this point it's worth asking what the adversary can do. She can be any participant in the protocol, and can block and send messages at will. if she blocks a message, to avoid detection, she has to replace it with something. In particular, she can *replay* messages from a previous run, or even potentially from that same run. This is a dangerous ability if the adversary ends up replaying from a previous protocol from which she has learnt the keys – we don't assume that the previous session keys are secure.

Let's see an attack on version 3 of the protocol, where the adversary impersonates the server by replaying exchanges from a broken session.

```
Attack on Key Exchange v3:
    1. A --> C: ID_A, ID_B
```

2. C --> A: {K_AB', ID_B}$_{K\_AS}$, {K_AB', ID_A}$_{K\_BS}$
3. A --> B: {K_AB', ID_A}$_{K\_BS}$

In the attack, we assume that K_AB' is the key from the broken session that the adversary knows.

One way that A could thwart this attack is by keeping a database of all previous messages and making sure that she doesn't see the same message twice. This is however incredibly impractical, both from a time perspective (time to search) and a storage perspective. Instead, we want to introduce some sort of automated replay protection. Options here are time stamps, nonces or counters. Time stamps and counters are non-random values, so easy to generate; however, it requires both parties being in sync, which is a management burden. A nonce is a 'number used once' and is generated randomly; we'll use this as our replay protection.

2.4 Attempt 4

Key Exchange v4 (Needham-Schroeder)

1. A --> S: ID_A, ID_B, n_A
2. S --> A: {K_AB, ID_B, n_A, {K_AB, ID_A}$_{K\_BS}$ }$_{K\_AS}$
3. A --> B: {K_AB, ID_A}$_{K\_BS}$, ID_A
4. B --> A: {n_B}$_{K\_AB}$
5. A --> B: {n_B - 1}$_{K\_AB}$

Ok at this point, we've reached a named protocol - the Needham-Schroeder protocol from 1978. The first thing to notice is that we have nonces, from both A and B; the goal of this is to assure A that her exchange with the server is fresh, and the goal of B's nonce is to be assured that his exchange with A is fresh. i.e. Because A was convinced that K_AB was fresh, by engaging in Bob's nonce challenge, she'll persuade Bob that the key is fresh.

However, this protocol is still vulnerable to an attack! For example, suppose you have an insecure session key K_AB'; then the adversary C impersonates A to perform steps 3-5 with the vulnerable key. Importantly, B's nonce isn't really persuading him of freshness - it's being used as key confirmation here, telling him that A shares his key. It's a challenge-response mechanism.

2.5 Attempt 5

Key Exchange v5

1. B --> A: ID_B, n_B
2. A --> S: ID_A, ID_B, n_A, n_B
3. S --> A: {K_AB, ID_B, n_A, {K_AB, ID_A, n_B}$_{K\_BS}$ }$_{K\_AS}$
4. A --> B: {K_AB, ID_A, n_B}$_{K\_BS}$

This time, the structure of the protocol is different: B now initiates, although still only A interacts with the server. Also, unlike in v4, A and B cannot be sure that the other party shares the created key.

3 Lessons from the example

At this point we'll depart from our long-running example, and try and extract some general principles.

3.1 Adversary model

Firstly, we need to explicitly state what our threat model is – what is the power of our adversary. A common threat model is the Dolev-Yao model. In this threat model, we assume that every transmitted message is carried by the adversary. This means that

1. We assume that the adversary does not have the power to break cryptography - she can't decrypt messages unless she has the decryption key etc.
2. The adversary can read any message that is transmitted.
3. The adversary can modify or replay any message.

This is what we've been implicitly assuming throughout our model. Note that this is a very cautious model; for example, we could loosen this assumption by assuming that `A` and the server have a private channel that the adversary cannot eavesdrop on. We could consider only *passive* attacks, where the adversary can only listen in (as opposed to *active* attacks, where the adversary is allowed to modify messages). Alternatively, we could give the adversary even more power - we could give it certain cryptanalytic powers (e.g. saying that a certain primitive is broken).

3.2 Security goals

We touched on some security goals today: freshness of the session key, and confidentiality of the key.

Here are some more things that we could consider:

1. authentication (see presentations!)
2. key confirmation
3. forward secrecy (see coursework!)...
4. ...and post-compromise security (see coursework!)
5. anonymity
6. server doesn't know the key
7. resistance to downgrade attacks (e.g. when the protocol is negotiating the algorithms to be used)

3.3 Symbolic vs computational security

We've been looking at symbolic security of a protocol today. Typically we use online tools, such as ProVerif or Tamarin, to model the protocol and to probe it for desired security properties. The online tool efficiently searches the space of the allowed adversarial actions to test whether a property does or does not hold. It's a very binary analysis, where we think of primitives as either being secure or insecure.

In contrast, we have the computational security model. This is more similar to the security reductions that you saw in Cryptology.

Further Reading

[1] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for authentication and key establishment*. Vol. 1. Springer, 2003.

[2] Iliano Cervesato. "The Dolev-Yao intruder is the most powerful attacker". In: *16th Annual Symposium on Logic in Computer Science—LICS*. Vol. 1. Citeseer. 2001, pp. 1–2.

[3] Dorothy E. Denning and Giovanni Maria Sacco. "Timestamps in key distribution protocols". In: *Commun. ACM* 24.8 (Aug. 1981), pp. 533–536. ISSN: 0001-0782. DOI: 10.1145/358722.358740. URL: https://doi.org/10.1145/358722.358740.

[4] Gavin Lowe. "An attack on the Needham-Schroeder public-key authentication protocol". In: *Information Processing Letters* 56.3 (1995), pp. 131–133. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(95)00144-2. URL: https://www.sciencedirect.com/science/article/pii/0020019095001442.