Kerberos

Scribe: Jethro Beekman

October 31, 2012

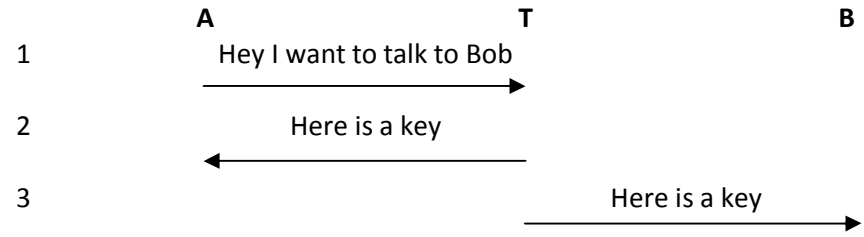
Kerberos

This is really about *Key Managemen*t. It sounds like they the way they designed Kerberos is really complicated. We'll try to answer the following question: "How could you have invented the Kerberos protocol?"

Last time we discussed how to setup secure channels; how to avoid replay attacks etc. When A and B have a shared key:

A->B: {A,B,ts,M}K
^ timestamp

Kerberos is about how to establish these shared keys between any number of nodes, using symmetric key crypto. You could have a different pre-shared key (PSK) between any two nodes, but this is inconvenient. What if all nodes have a PSK with a single node, the trusted authority? Alice sends $\{K_{AB}\}K_{AT}$ to T, T decrypts it and sends $\{K_{AB}\}K_{BT}$ along to B. A and B now have a shared key. This is the Needham-Schroeder[1] protocol. All K_{xT} are setup 'out of band', like in-person at the computer helpdesk. Now how to turns this into a Protocol?



Adding all the security goodies:

(1) A->T: $\{A,T,ts,B\}K_{AT}$

(2) T->A: $\{T,A,ts,A,B,K_{AB},t_{exp}\}K_{AT}$

(3) T->B: $\{T,B,ts,A,B,K_{AB},t_{exp}\}K_{BT}$

We can add the identity of A (unencrypted) to (1) so that T knows which key to use. This improves performance but not security.

[1] Needham, Roger; Schroeder, Michael (December 1978). "Using encryption for authentication in large networks of computers.". Communications of the ACM 21 (12): 993–999. doi:10.1145/359657.359659

In Kerberos, T never contacts B directly. How does that work with this scheme? (3) goes via A. Because we use cryptographically secure channels, this is the same from a crypto perspective. It might be better from a performance standpoint. For example, it reduces the connection state for T.

The Kerberos spec doesn't look exactly like this, there are some minor differences. For example, maybe nonces get used instead of timestamps. Or in our (2) and (3), A and B, respectively, appear twice in the message, this might not be the case in Kerberos.

You can see that when you try to turn something simple like this into a cryptographic protocol it gets complicated. You probably never have to design a crypto protocol. This stuff is hard. Review of it is hard. For example, there were two versions of the Needham-Schroeder protocol. One uses symmetric crypto, which was secure. The other uses public-key (PK) crypto, which had a subtle flaw that wasn't found until 17 years later. Today, you usually just use an OTS protocol that does what you need. There are also automated analysis tools. When you use these things, you should be aware that although your design might be secure, but your implementation might not be.

Student questions

Q: What about crypto versions? What are Kerberos v1,2,3 and SSLv1?
A: They were probably broken and were never used/released.

Q: Is Kerberos used today?
A: Not much, mostly within (academic) organizations. There are a lot of implementation differences between e.g. Windows and Linux implementations.

Q: Why doesn't Kerberos use PK crypto?
A: Not sure. Back in the day, people thought PK crypto was expensive.

Q: Kerberos uses password based keys, isn't that bad?
A: Yes, because the entropy of a password is much lower than that of a random key. Why is this worse than logging in with your password? You can capture traffic and brute-force it offline. Online you can do rate-limiting etc.

Q: How do we ensure time synchronization?
This is out of the scope of the protocol. Most time synchronization is done with NTP, which is not designed to be secure. You can do DoS or replay attacks when impersonating NTP servers. The latest NTP version has authentication. Basically: when you're using timestamps, be sure you're doing time sync right.

Q: What about key/ticket revocation?
A: This is included in Kerberos v5, tickets are single-use (not the ticket-granting service ticket).

Q: Any other ways to include revocation in your protocol?
A: Every time you use a key, first ask T if the key is still valid. This is like OCSP. Also in PK infrastructure (PKI) with Certificate Authorities (CA), certificates have expiration dates, which can be anywhere

between “an hour from now” and “10 years from now”. There is a trade-off between revocation expiration and certificate signing.

Q: In Kerberos, what happens when T goes down?
A: Existing tickets will still be valid, but those will expire eventually. You can set up T redundantly.

Q: What are the main differences between PKI and Kerberos?
A: In PKI you can use the same certificate for many different connections, load on CA is low. Kerberos also has access control built-in. What happens if T/CA gets compromised? Compromising the CA doesn’t give you the private keys of A/B. In Kerberos you can decrypt all traffic you may have passively captured in the past or during the compromise. In PKI, the only thing you can do is a MITM attack while you control the CA. There is also the privacy issue that in Kerberos, T knows who you’re talking to. This also happens when you use OCSP.

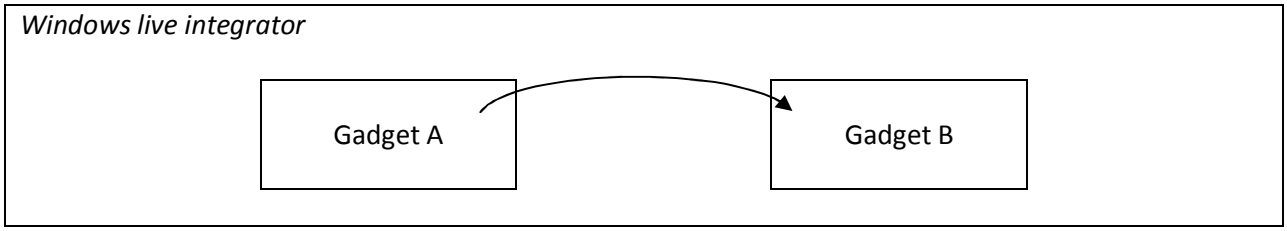
Closing remarks

Kerberos doesn’t tell you what to do with the data. MAC’s are required, but encryption is optional. Also, Kerberos has had a lot of security issues in the past, for example, the PRNG was bad.

One-line summary

If you have a trusted third-party who everyone is willing to trust, you can setup secure channels between anyone.

Windows Live gadget cross-frame communication



Windows live is an environment for mash-ups. The same origin policy prevents communication between gadgets A and B. There is a loop-hole which is called the fragment ID channel. A frame can set (not read) the URL on a different (target) frame. By setting the URL fragment part (part after the #), the page doesn’t get reloaded. Now the target frame can read the fragment, et voila, communication channel. This is like a mailbox, anyone can drop off mail, but only the person with the key can read.

To setup an authenticated channel, you can do something like this:

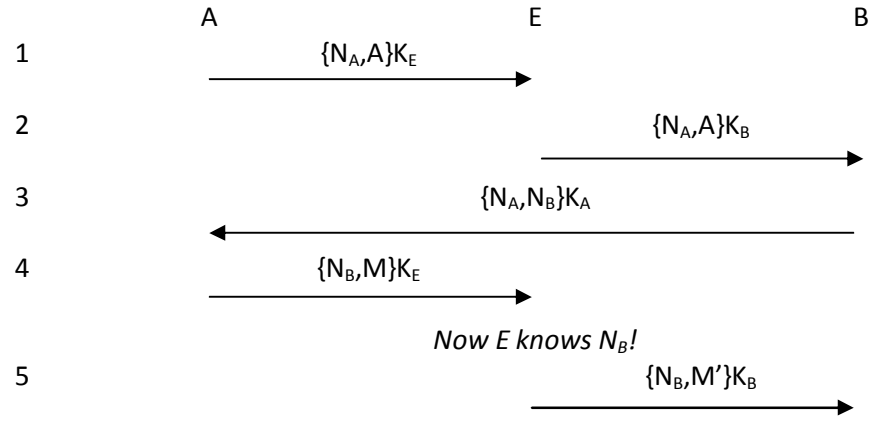
A->B: #N_A, FrameID[A]
B->A: #N_A, N_B
A->B: #N_B, M

This is similar to the TCP handshake. Also, it is very much like the Needham-Schroeder PK protocol:

A->B: $\{N_A,A\}K_B$
B->A: $\{N_A, N_B\}K_A$
A->B: $\{N_B,M\}K_B$

Remember that anyone can sign something with a public key, but only the owner of the private key can decrypt it. You can see the similarities with the mailbox and fragment ID channel.

There is a MITM attack on this protocol, which can also be done on Windows Live. E tricks A into talking to E instead of B.



This flaw was found using a protocol verifier by Lowe[2]. The fix is to include the source in message 3:

(3) $\{N_A,N_B,B\}K_A$

Now A sees that she got a response from B while expecting one from E.

Closing remarks

It is hard to see while some fields are necessary in a crypto protocol. Don't over-optimize your protocol if you don't know what you're doing, because you might remove some innocuous looking stuff that actually adds a great deal of security!

[2] Lowe, Gavin (November 1995). "An attack on the Needham-Schroeder public key authentication protocol.". Information Processing Letters 56 (3): 131–136. doi:10.1016/0020-0190(95)00144-2