Today

Security protocols

Protocol notation

Examples

- Needham—Schroeder and Kerberos
- Diffie-Hellman and TLS

Recall: Dolev-Yao attacker*

Communications should assume an attacker can:

1. observe all messages (**passive eavesdropper**),
2. send messages impersonating users (**active** attacker) and
3. intercept messages and drop or replace them (**middleperson**).

Be explicit about trust and communication: *security protocols*

---

* Dolev and Yao, "On the Security of Public Key Protocols", *IEEE Transactions on Information Theory* 29(2), 1983. DOI: 10.1109/TIT.1983.1056650

Security protocols

Rules for interaction among multiple parties

- notation and conventions
- primitives

Implicit: understandings of *belief*

- can be modeled formally
- not our focus

The protocol notation that we'll see in this class will describe things like messages that are sent from one party to another and cryptographic primitives that are used to protect them.

It's possible to model these beliefs formally. The BAN logic was an early example of this, with primitives such as:

$P \mid\equiv X$: P *believes* X

$P \triangleleft X$: P *sees* X

$P \Rightarrow X$: P *has jurisdiction* over X

It's also possible to model protocols in such a way that automated *theorem proving* can be applied to them, using tools like ProVerif.

Security protocol notation

Primitive	Notation	Meaning
Message	$A \to B : P$	Alice sends the password P to Bob
Hash	$A \to B : h(P)$	Alice sends the hash of P to Bob
Encryption	$A \to B : \{P\}_{K_B}$	P is encrypted under Bob's key
Shared key	$A \to B : \{P\}_{k_{AB}}$	k_{AB} is shared by Alice and Bob
Signature	$A \to B : \{P\}_{K_A^{-1}}$	Signature of P using K_A
Nonce	$A \to B : N_A$	A *number used once*
Timestamp	$A \to B : T$	Alice sends Bob the current time

Note: this protocol notation includes no specification of _____ cryptographic algorithms are used! Instead, cryptographic primitives are assumed to work correctly and provide their promised poperties. Proofs can then be done using the Standard Model, or more practically, in the Random Oracle Model that approximates cryptographic primitives with functions that produce random values.

Examples of common protocols

Needham–Schroeder and Kerberos

Diffie-Hellman key exchange

TLS: Transport Layer Security

OTR: Off-the-record messaging*, Signal and friends

---

* Borisov, Goldber and Brewer, "Off-the-record communication, or, why not to use PGP", in *WPES '04: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, 2004. DOI: 10.1145/1029179.1029200. }

Security protocol example

Needham–Schroeder protocol

$$A \to S : A, B, N_A$$

$$S \to A : \{N_A, k_{AB}, B, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$$

$$A \to B : \{k_{AB}, A\}_{k_{BS}}$$

$$B \to A : \{N_B\}_{k_{AB}}$$

$$A \to B : \{N_B - 1\}_{k_{AB}}$$

- Hi server, I'm Alice and I'd like to talk to Bob. Here's a random number.
- OK, Alice, if you really are Alice you'll be able to decrypt this information. Here's your random number back (to prove that **I know** k_{AB}), a key you can use to talk to Bob, Bob's identity and a **ticket for you to give Bob**.
- Hi Bob, here's something that the server asked me to give you.
- OK, if you really are Alice, you should be able to prove that **you know** k_{AB} by **decrypting and manipulating this nonce value**.

What's the flaw?

_____, which means that if an attacker _____, they can _____ _____. This has been fixed in Kerberos through the use of _____.

Diffie-Hellman key exchange

Alice chooses random number X_A, Bob chooses X_B

$$A \to B : \alpha^{X_A} \pmod{q}$$
$$B \to A : \alpha^{X_B} \pmod{q}$$

$$\begin{aligned} k_{AB} &= \alpha^{X_A X_B} && \pmod{q} \\ &= \left(\alpha^{X_A} \bmod q\right)^{X_B} && \pmod{q} \\ &= \left(\alpha^{X_B} \bmod q\right)^{X_A} && \pmod{q} \end{aligned}$$

But remember: **there is a weakness!**

We've described Diffie-Hellman key exhange before, using English prose, but now we'll see it written down with a more formal notation. Hopefully this will help us spot the _____ in the protocol!

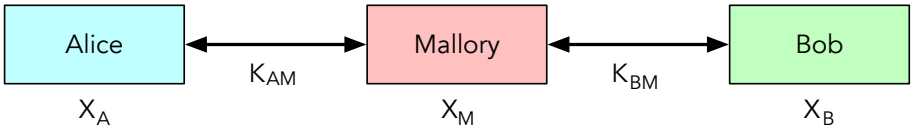
At this point, A and B share a key. But _____ A and B? What basis do Alice and Bob have to _____ that A is Alice and B is Bob?

Middleperson attack

$$A \rightarrow M : \alpha^{X_A} \pmod{q}$$
$$M \rightarrow A : \alpha^{X_{M_A}} \pmod{q}$$

$$B \rightarrow M : \alpha^{X_B} \pmod{q}$$
$$M \rightarrow B : \alpha^{X_{M_B}} \pmod{q}$$

$$k_{AM} = \alpha^{X_A X_{M_A}} \pmod{q}$$
$$k_{BM} = \alpha^{X_B X_{M_B}} \pmod{q}$$



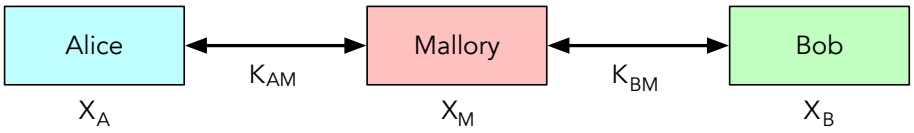
Suppose we have a _____ named Mallory (M) who acts as a Dolev-Yao attacker, sitting on the wire between Alice and Bob.

When Alice sends her α^{X_A} message to Bob, Mallory can intercept it. Mallory can then impersonate Bob, so Alice and Mallory act out the first step of the protocol where Alice is A and Mallory is B.

The same then happens to Bob: Bob thinks he's talking to Alice (Alice is A and Bob is B), but in fact he's talking to Mallory. Thus, from Bob's perspective, Mallory is A and Bob is B.

After running the protocol with Mallory in the middle, Alice and Bob have established "secure" communications using symmetric-key cryptography, but only in a technical sense, not a meaningful sense!

Middleperson detection



- "Wait a minute, that's not the same as my k_{BM} !"
- **Q:** how do we check K_A ?
- **A:** TLS *certificate authorities*

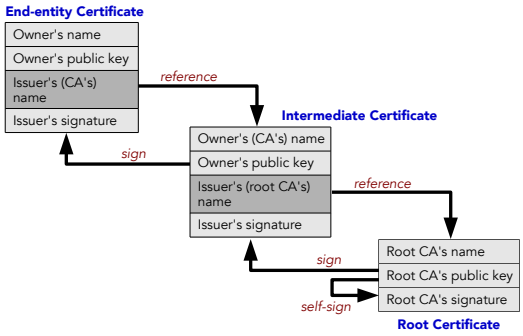
$$A \to M : \alpha^{X_A} \pmod{q}$$
$$M \to A : \alpha^{X_M} \pmod{q}$$
$$B \to M : \alpha^{X_B} \pmod{q}$$
$$M \to B : \alpha^{X_M} \pmod{q}$$
$$A \to B : \{k_{AM}\}_{K_A^{-1}}$$

This works well as long as _____. That's fine for point-to-point communications, but what about Web browsing, in which I'm connecting to — and needing to trust — services like my bank and my email provider?

Certificate authorities

Recall:

- Root CA signs (intermediate CA signs)* entity certificate
- Used for code signing, started with SSL/TLS



Source: Wikimedia Commons

Question:

What if a certificate authority were compromised or malicious?

Search terms: DigiNotar, TURKTRUST, **Certificate Transparency**

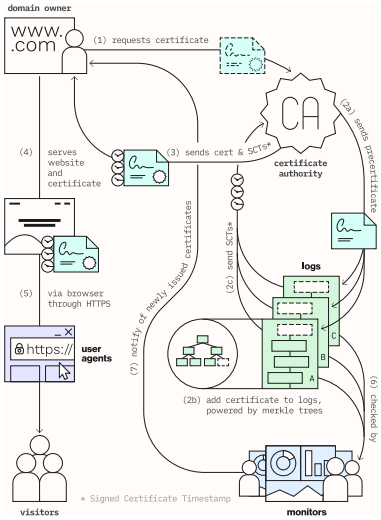
This is not a theoretical consideration! TLS puts a lot of trust in certificate authorities, so mistakes made at a CA can have wide-ranging effects. CAs can also be high-value targets for attackers seeking to break the confidentiality and integrity properties of TLS for large numbers of users or, more worrying, small numbers of users (even targeted individuals).

Certificate transparency

Certificate metadata:

- certificate issuance
- certificate auditing
- cross-signing

Now *required* by Chrome and Safari (though not by Firefox)



Source: certificate-transparency.org

Before issuing a cerrtificate, a CA will first ensure that the certificate has been logged in a public append-only record (a kind of blockchain!). This gives them a Signed Certificate Timestamp, which indicates when the certificate should be merged to the public log. This SCT can then be included with certificate chains to say, "not only has my certificate been signed by a chain that goes back to this CA, the CA has publically committed to having issued that certificate." This means that, even if a CA goes rogue, they can't issue a certificate that my CT-expecting browser will accept unless they _____.

Website owners can then work with *monitors* who watch the public certificate logs and look for funny business. For example, if a monitor notices that a new certificate was issued for `google.com` but Google doesn't know anything about it, they can raise the alarm that someone is trying to impersonate Google and they can _____ _____. This dramatically increases the risk for a rogue CA of issueing a fraudulent certificate.

TLS: Transport Layer Security

Most important protocol on the Internet

- Most common example: HTTPS
- Slow-moving train wreck from SSL 1.0 ⇒ successful TLS 1.3
- Amazing example of difficulty in herding cats

Shockingly difficult to use correctly*

---

* Georgiev et al., "The most dangerous code in the world: validating SSL certificates in non-browser software", in *Proceedings of the 2012 ACM conference on Computer and communications security (CCS '12)*, pp. 38–49, 2012. DOI: 10.1145/2382196.2382204.

There has been a steady parade of attacks against SSL 1.0 through SSL 3.0 and then TLS 1.0 through TLS 1.2. These have exploited things has subtle as improper ordering of primitives (encrypt-then-MAC vs MAC-then-encrypt) and incorrect padding within encryption. Weak Diffie-Hellman parameters, lack of forward secrecy... there's so much that we could go into if this were a course on the history of TLS vulnerabilities.

Aside from the difficulties of getting the protocol right, it's difficult to get people to use TLS APIs correctly! APIs that expose complex and subtle behaviours to experts are — surprise! — not always used correctly by non-experts who just want their app to "work".

> *Do not meddle in the affairs of wizards, for they are subtle and quick to anger.*

— J.R.R. Tolkien, *The Fellowship of the Ring*

Summary

Security protocols

Protocol notation

Examples

- Needham—Schroeder and Kerberos
- Diffie-Hellman and TLS