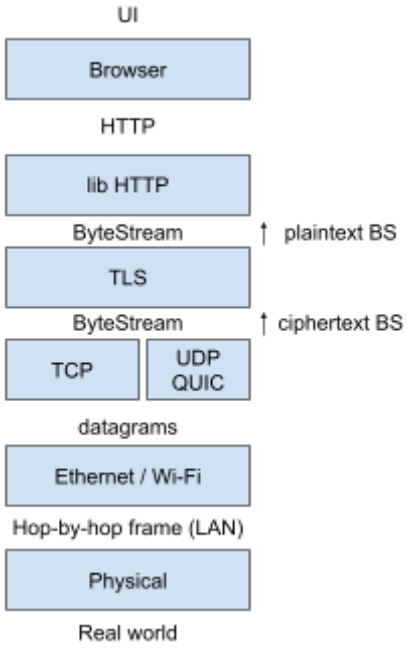
Today: Security



- Before discussing the system property, a common understanding of the threat model is necessary.

Threat Model	Mitigations / Techniques	System Property
Accidental corruption	- IP header checksum - TCP/UDP has header + payload checksum - Ethernet has header + payload FCS	**Integrity** - data received = data sent
Adversarial modification (Modify dst address / payload and modify the checksum)	- Secure hash with agreed hash value - Message Authentication Code	
Replay	Idempotence of messages	
	- AEAD - AKE	**Confidentiality** - only intended recipients can see the message
		Authenticity - parties are who the say they are

Cryptography Tools

- Secure hash algorithm: `hash`： X: arbitrary-length -> Y: 256 bits
  - `hash` is a one-way function. In other words, given `y`, it's hard to find the `x` such that `hash(x)=y`.

    - If two parties agree on the `y` before-hands, then the receiving party can verify whether the `x` is not corrupted by calculating `hash(x)`.
    - (But if someone corrupt the message for sending `y`, and change it to `y'`, which it get from `x'` such that `hash(x') = y'`, this may still be insecure, so that the process of sending `y` needs to be done in a 100% secure way: e.g. hand a physical piece of paper in person. And this needs to happen for every `x`).
- Trust On First Use (TOFU)
- Message Authentication Code (keyed hash)
    - `mac(x, key) -> tag`
    - `verify(x, tag, key) -> bool`
    - The adversarial party cannot generate a `tag` that passes the `verify` without knowing the `key`.
    - The key still needs to be sent in a secure way, but this only needs to be done once.
- Authenticated Encryption (AE(AD))
    - `box(plain text, key) -> (cipher text, tag)`
    - `unbox(cipher text, tag, key) -> optional<plain text>`
    - It's hard to generate a pair of `(cipher text, tag)` to pass the `unbox` function, and it's hard to `unbox` a cipher text without knowing the `key`.
    - But still , we have the pain of how to establish a shared secret.
- Public-key Cryptography / Authenticated Key Exchange (AKE)
    - Alice: (public key_1, private key_1) and
      Bob: (public key_2, private key_2)
    - So Alice know public_1, private_1 and public_2
    - Bob know public_2, private_2 and public_1
    - Alice sends some x_1 to Bob
    - Bob sends some x_2 to Alice
    - Adversarial parties can observe public_1, public_2, x_1, x_2
    - And, we have: AKE(x_1, x_2, private_1, public_2) = AKE(x_1, x_2, private_2, public_1) = *key* and this *key* is only known by Alice and Bob.

====

- Logistic: quiz on Wednesday

Last Time: Security

- Security Properties: Integrity, Confidentiality, Authenticity
    - Authenticity is necessary for Confidentiality

Threat Model	Mitigations / Techniques
Accidental corruption	- checksum/CRC

Adversarial modification	- Secure hash - Message Authentication Code (keyed hash)
Replay	Idempotence of messages
Eavesdropping	encryption (AE AD)
	Authenticated encryption requires a pre-established shared secret. For communication with strangers: - Trusted Third Party (Kerberos/Windows Active Directory): either relay the connection or Trent generates a new secret key and gives that to Alice and Bob - AKE

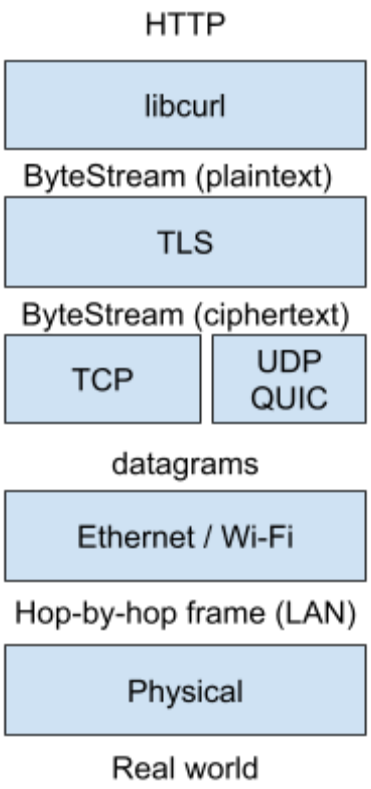
- peers: Alice + Bob
- eavesdropper: Eve
- adversarial modification: Mallory
- trusted third party: Trent
- AKE:
    1. Alice generates a key pair: $(public_{Alice}, private_{Alice})$
    2. Bob generates a key pair: $(public_{Bob}, private_{Bob})$
    3. Alice and Bob publishes their public keys
    4. Alice sends some x_1 to Bob, and Bob sends some x_2 to Alice
    5. Alice gets $AKE(x_1, x_2, public_{Bob}, private_{Alice}) = secret\ key$ and Bob gets $AKE(x_1, x_2, public_{Alice}, private_{Bob}) = secret\ key$, and the secret keys that Alice gets and Bob gets are the same.
    6. In addition, knowing $public_{Alice}$, $public_{Bob}$, x_1, x_2 does not reveal the secret key.
- But how do we know the public key of say Target?
    - Asking directly from Target does not work, since that message may be corrupted.
    - For a small number of entities, there could be a directory of public keys that were shared in a 100% secure way (e.g. an in-person meeting)
    - Or you could ask someone that you trust and you already know his/her public key
        - $sign(private_x, msg) \rightarrow signature$
        - $verify(public_x, msg, signature) \rightarrow bool$
- e.g. You are asking Keith for John's public key
    - $sign(private_{Keith}$, "John's public key is <x> according to Keith (expiring at time t)" $) \rightarrow signature_{Keith}$
    - $verify(public_{Keith}$, "John's public key is <x> according to Keith (expiring at time t)", $signature_{Keith}) \rightarrow true$. Then, this is a "certificate" that Keith verifies John's public key is <x>.

    - John can store this certificate, and show this to any person that trusts Keith to prove that John is actually John.
- Firefox —----------TCP---------- Target @ “target.com”
    - Firefox trusts a list of certification authorities (whose public keys are programmed into Firefox)
    - When Firefox connects to “target.com”, Target, to prove Target is actually Target, would provide:
        - “Hi, I’m target.com. My public key is <x>. Here is a certificate from a CA you trust”.
        - And a certificate: “Target come’s public key is <x> according to <CA>” + $signature_{CA}$ from $private_{CA}$.
    - Firefox:
        - $verify(public_{CA}$, “Target come’s public key is <x> according to <CA>”, $signature_{CA}) \rightarrow true$
    - Then Firefox and Target does AKE to get a shared secret key. This shared secret key is used to do $AEAD$ for all following communication in the current TCP connection.
    - These steps happen as part of the TLS layer. TLS translates between plaintext and ciphertext
- Q & A
    - A: This list of CAs is common across different browsers.
    - Q: How does a CA decide to give the certificate to a specific entity?
      A: CA would have an intensive verification process (back in the days), but over the time the standard has been lowered. Now it’s done via domain verification: if someone can put a provided verify.txt at URL/verify.txt within 5 minutes, a CA gives the certificate. This is indeed not secure, since DNS and routing are not secure.
    - Q: What if CAs are forced to grant a certificate?
      A: Certificate Transparency Log: a log of all certificates granted by CAs. Big companies monitor this
- The shift from HTTP and HTTPS was triggered by the fact governments were monitoring all the traffics (refer to the slides for more information).

===

Last Time:

- Security Properties: Integrity, Confidentiality, Authenticity
    - ,and authenticity is necessary for confidentiality
- Certificate provides authenticity
    - (Opportunistic encryption: even if you are not sure who you are talking to, still do encryption. This kind of confidentiality makes it harder for third parties (e.g. governments) to learn the content of the traffic, though it does not require authenticity.)



- Does this layer of TLS solve everything?
- **Issue 1:** Certificate authorities may be corrupted / intentionally issued not correct certificates. The transparency log helps to mitigate this, but it is not a 100% solution.
    - Big companies would monitor the transparency log
    - Browsers may require the certificate to also contain a proof that the certificate is from the transparency log
- **Issue 2:** Even if the payload of Internet datagrams is encrypted, you could still tell who is talking with who by the src/dst in the IP header. **"Metadata privacy"**.
    - VPN or through one relay server: governments can't tell who is talking with whom, but they can still get the info by threatening the relay server
    - How about more than one relay server? Essentially, any single relay server cannot see the full picture of the connection.
        - Onion routing: layers of encryption and each relay server can only take out one layer of the encryption. **The Onion Router (TOR).**
        - Each relay server only knows the hop before itself and the hop after itself
    - But: the timing still reveals something
- **Threats:** eavesdropper timing attack correlation
    - Something is being sent from TOR to Netflix at 1:33 a.m.
    - And if there is a short list of people that were using TOR at that time
    - It may not be that difficult to tell who is actually sending to Netflix
    - Q: How do third parties (say governments) tell people that are using TOR?
      A: TOR relay servers are public and TOR traffic may look different. Governments could figure out TOR relay servers IP addresses and block them. (To fight against

this, TOR relay servers IP addresses are released slowly, and they are trying to make TOR traffic look as innocent as normal traffic).

- **Threats**: Sybil attack
    - TOR works only if the relay servers are not colluding. If a certain entity has a huge number of TOR relay servers, then there is a high probability that a whole sequence of relay servers belong to this entity, then the entity can learn about the connection.
- TOR hidden service
    - Allow publishers and users of services to hide their identity.
    - Normally contains 6 hops of relay, 3 picked by the publisher, 3 picked by the client.
    - By exploiting security holes, governments can still reveal who was visiting/posting on TOR web servers. (Put the security hole exploit on the web servers of these hidden services, then whoever downloads the content would also download the security hole exploit).