Message Security

- So far: TLS which gives a channel to teh server
  - Big assumption: both parties are online at the same time which gives us fresh keys based on the handshake
- Problem: messaging is asynchronous
  - Want to send messages even if another if the other person is offline
- Most popular protocol: Signal protcol
  - WhatsApp
  - FB messenger
  - iMessage

Strawman

- Send from A to B using the public key of B
  - Also signed with the signing key of A so B can ensure it is A's
  - If the signature from A to B is sent separate from the ciphertext, this is bad
    - How do you know that the ciphertext was sent by A?
    - Anyone could sign the ciphertext and replace A's signature with it
    - This is a new problem in the asynchronous context b/c otherwise before you would know that the two messages are coming from the same source
  - There is a sophisticated way to include the signature in some way inside of the ciphertext

Authenticated Key Exchange

- Idea is we want to somehow do some key exchange asynchronously
  - After this, both A and B will have some shared stream of keys that they can use to decrypt messages
  - How we actually encrypt each message is not really talked about too much
  - More concerned with just generating these shared keyse

Forward Secrecy

- Provide some security even after someone is compromised
- All previous traffic should not be compromised
  - TLS solved this more easily with ephemeral session keys because both computers were alive during the connection

Signal's Approach:

- Both A and B have PK and SK for their identity
- When A wants to send a message to B, it generates a new ephemeral key
  - Combines ESK_A (ephemeral secret key) with PK_B to get K_1
  - Sends message encrypted with K_1
    - Also sends the the EPK_A so B can construct K_1 on their side
- Deleting the ephemeral key will not allow us to recover this
- Problem: compromising B compromises these messages
- Solution: B also needs ephemeral keys
  - When signing up for Signal app, they pregenerate a bunch of ephemeral keys and send them to the server
  - When A wants to talk, they ask the server for public key for B and one of B's ephemeral keys
    - They then combine B's ESK with SK_A to get K_2
    - K_1 and K_2 are then combined to get K
      - Using some hash and stuff (key combining function)

        - This is used to send message to B
    - B deletes ephemeral key after it receives a message using it
- We will see on Thursday how we can be transparent about public keys / A can ensure they have the correct public key for B
    - If the server is malicious and gives out fake ephemeral keys, they will be bogus
        - B will not be able to decrypt them
        - Server won't be able to decrypt either since the key will require B's secret key
    - There is a problem where what if the server does this and then later they break into B's server
        - Solution: medium term keys
        - Essentially there is some medium term key that is prepared by B that is signed
        - This is also used to encrypt the message to ensure B only knows about this
        - So multiple people will be sharing these medium term keys and B will swap them out every once in a while to ensure that a server cannot give out fake ephemeral keys and then later break into B's server years later
        - This generates a third key that is combined with the other two
- Problem: what if you compromise **both** A and B
    - Compute a fourth key K_4 that is combination of ephemeral keys from both sides
    - K can then be computed by concatenating all three of these
    - K1 ensures only A can read
    - K2 ensures only B can read
    - K3 ensures server can't read
    - K4 ensures forward secrecy
- Problem: what if the server runs out of ephemeral keys
    - You still have the medium term key used to sign so you just omit the K4
    - If B gets messages with the medium term key, it can maybe try deleting it

- So far: this has generated one key for a single message
    - We need a plan to derive more: ratchets

Multiple Messages

- If we just reuse the shared key we just derived then that's bad because then you have to keep this stored
- We can generate a chain of keys using a key generation function
    - Both sides will do this and generate the same keys
    - After a key is used, it can be deleted
    - You only need to store the top of your current chain
        - After sending a message, you can ratchet forwards and then skip
    - Its a one way hash function so you can't recover the previous ones

Post Compromise Security

- What signal tries to provide with asynchronous ratchet
    - We so far have told you about synchronous ratchet
- Problem: if someone breaks into phone, they can compromise all future keys
    - Just go forward with the chain
- Idea: if B is compromised they should have an opportunity to inject some randomness / entropy back in to get rid of the compromise
    - Build on top of the symmetric ratchet
    - A and B first agree on a key K
    - They then have an asymmetric ratchet
        - A generates a private / public key pair and sends public it to B
        - B generates a private / public key pair and then combines private with previous public from A and sends the new public to A
        - A and B can then use this with DH to generate a shared secret

        - They can continue generating new public / private key pairs and sending them to each other to form this asynchronous ratchet of shared secrets that are entirely ephemeral
    - They combine this shared secret from their asynchronous ratchet with the K to form a chain key
        - Each time their asynchronous ratchet shifts, they make a new chain key
        - These chain keys are used to encrypt a set of messages that a person sends without a response
- Therefore if a party is momentarily compromised and one of these chain keys gets leaked, on the next round trip it will be reset

Deniability

- We want the ability for anyone to be able to spoof a conversation transcript
    - We don't want a trace of this conversation to be leaked and then be proven that it actually happened
    - We can deny that a conversation happened
- Typically, a SK can sign, which is used as proof that someone sent something
    - This would be bad to use in a message
- Instead Signal uses MAC addresses to designate where a meessage came from and B verifies a message came from A based on decrypting and finding the MAC inside of it
    - MAC addresses are computable by anyone