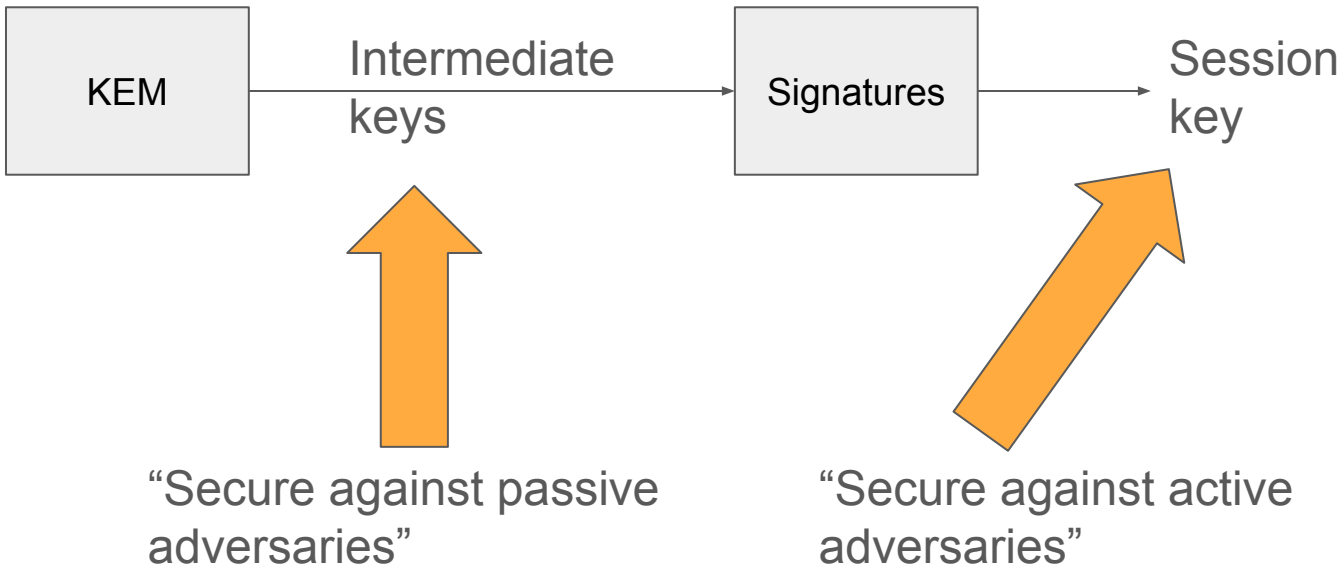
Extended ~~Key~~ FATT Update

Summary of FATT discussion on draft-ietf-tls-extended-key-update

Thom Wiggers

TLS 1.3 as viewed as a formal proofs person



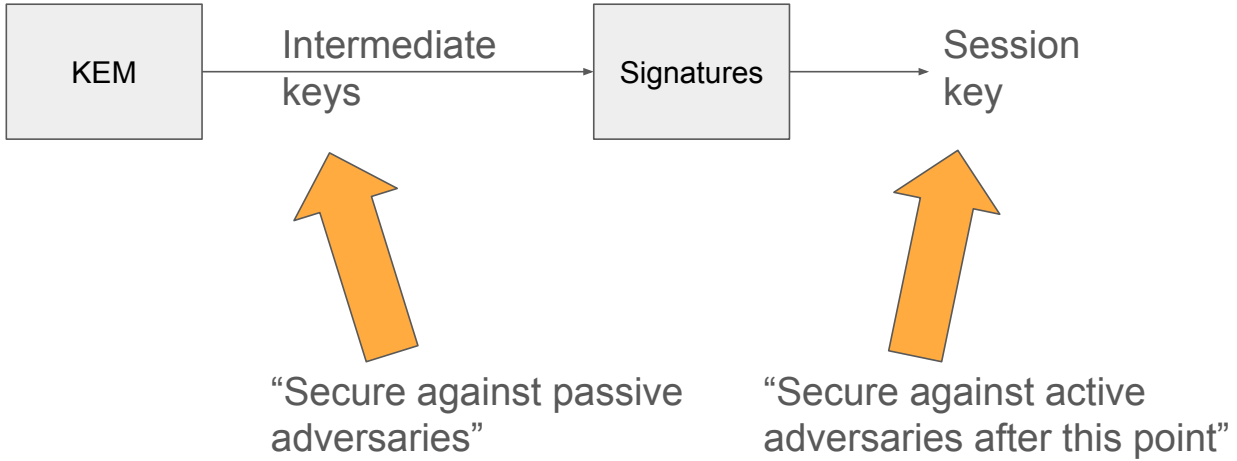
TLS 1.3 as viewed as a formal proofs person

Properties such as **Forward Secrecy** are defined using statements roughly like the following:

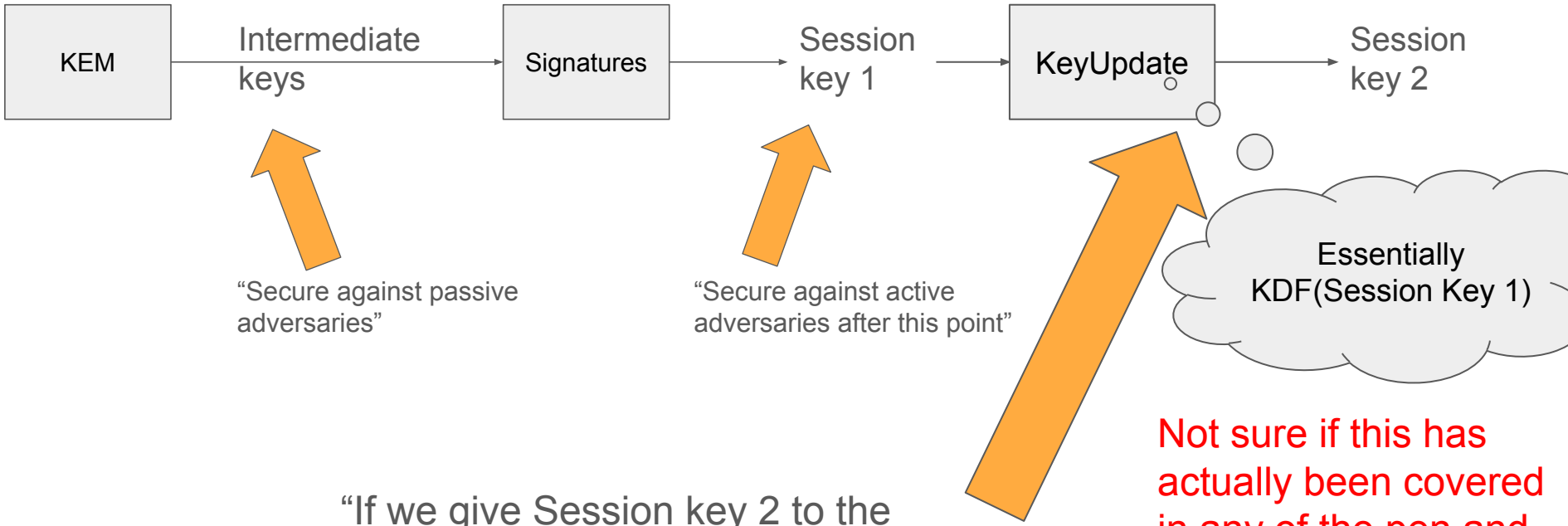
“The session key is hidden from the adversary if they don’t know the ephemeral keys OR don’t know the signature keys **when the session key is established**”

Which implies

“Even if we give the adversary the signing keys, if we do that **after** the session key is established it remains hidden”



TLS 1.3 + RFC8446 KeyUpdate as a formal proofs person



“If we give Session key 2 to the attacker, Session key 1 remains hidden from the adversary” (Forward Secrecy)

Not sure if this has actually been covered in any of the pen and paper proofs! (yet?)

How would I prove RFC 8664 Key Update

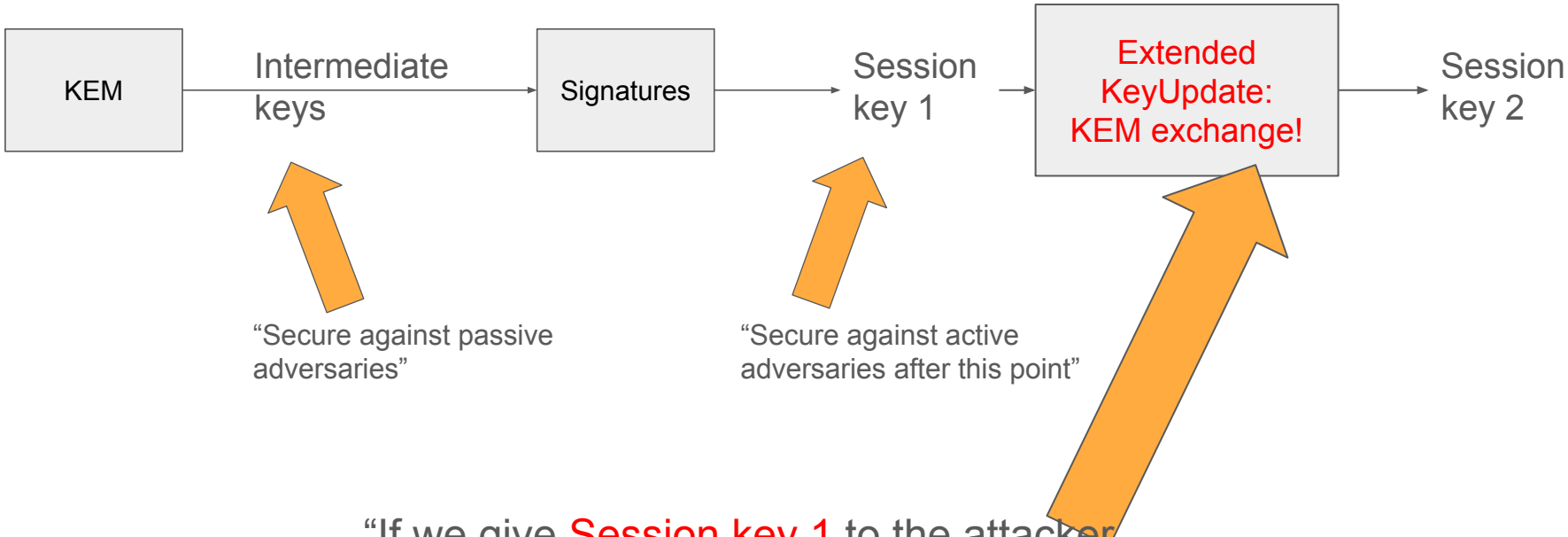
Because of the proof of security for session key 1 and because I won't reveal Session Key 1, I know that there is no adversary present when KeyUpdate happens.

=> I don't need to worry about malicious KeyUpdate and what that means

=> I can continue to re-use what I've proven in prior stages because that makes sense in my security model

This makes life quite a bit easier

TLS 1.3 + draft-ietf-tls-extended-key-update



“If we give Session key 1 to the attacker, Session key 2 remains hidden from the adversary (who does not mess with EKU)”
(Post-Compromise Security)

What does this mean for proving security

Because PCS is about recovering security after compromise, we **do** want to leak our proven-confidential-and-authentic Session Key 1 to the adversary

=> I need to worry about malicious ExtendedKeyUpdate and what that means

Probably restrict the adversary to be passive-only **after** being active before

=> I can not necessarily continue to re-use what I've proven in prior stages because the adversary gets to play with my session key for a while

This makes life quite a bit harder when writing a proof

Prior/Related Work and Analyses

Mixing in new asymmetric keys during the handshake also happens in e.g. **Signal.** Crypto proofs people call this “Continuous Key Agreement” (CKA)

- Many proofs for Double Ratchet’s asymmetric ratchet exist
- CKA proofs often assume some magic secure key exchange happened first
- Composition between “key exchange” (Signal: PQXDH) and “CKA” often left as exercise to the reader
  - Theoretically, this is a gap

IKEv2 has a similar mechanism for SA rekeying of which I am not aware of analysis (extra complication: IKEv2 supports full renegotiation of everything)

Where are we on draft-ietf-tls-extended-key-update

- The EKU CKA mechanism is mostly very simple
  - No negotiation
- Confusion on earlier versions of the draft between FS and PCS appears to have been mostly resolved
- Subtle issues in e.g. key computation and the many keys in TLS can still be tricky

The FATT acknowledges that the authors have put in a lot of honest effort on this!

“Simple” key computations can hide lots of subtlety

```
      Main Secret N
            |
            v
      Derive-Secret(., "derived", "")
            |
            v
(EC)DHE -> HKDF-Extract = Main Secret N+1
            |
            +-----> Derive-Secret(., "c ap traffic",
            |                EKU(key_update_request) ||
            |                EKU(key_update_response))
            |                = client_application_traffic_secret_N+1
            |
            +-----> Derive-Secret(., "s ap traffic",
            |                EKU(key_update_request) ||
            |                EKU(key_update_response))
            |                = server_application_traffic_secret_N+1
            |
            +-----> Derive-Secret(., "exp master",
            |                EKU(key_update_request) ||
            |                EKU(key_update_response))
            |                = exporter_secret_N+1
            |
            +-----> Derive-Secret(., "res master",
            |                EKU(key_update_request) ||
            |                EKU(key_update_response))
                             = resumption_main_secret_N+1
```

This key schedule from prior version -08 computed traffic secrets directly from the previous Main Secret.

- **Good:** the EKU messages are hashed into the traffic secrets
- **Bad:** Main Secret N did not include the prior transcript from the TLS handshake or earlier EKU, which were dropped on the floor.

This may make proving things harder! But maintaining the transcript also has implementation side-effects.

-09 starts maintaining a transcript (PR#95).
Discussion on list on the complicated side-effects…

(The RFC8446 computations for *`_application_traffic_secret`include ClientHello to Server Finished).

Assorted FATT Quotes

The discussion of the FATT (partially from a much earlier version) continued the theme of EKU requiring new considerations of many security properties of TLS 1.3 in subtle ways:

“Actually achieving this in practice will require more work around the edges than I think the authors realize. For example, **all past session tickets need to be expired during an extended key update**, otherwise an attacker who did not MiTM the extended key update, can reset the connection and resume with a session ticket in order to get back in.” — Unnamed FATT member

“...this will change the security properties of ‘normal’ TLS (not necessarily in a good or bad way, but it will change) and the security consideration section will need to be updated post-analysis to reflect proper guidance.” — Anonymous FATT member

FATT recommendation

This draft, which extends TLS 1.3 with a Continuous Key Agreement protocol, **requires analysis**

- Existing computational analyses of TLS 1.3 won't extend (easily) to the new extension's mechanisms
- Symbolic analysis may also be helpful

Reminder: the FATT gives advice on a best-effort basis. We have **not** done a "proper" security analysis or proof. We are not saying the work by the authors is not good enough. We are also **not** a gatekeeper for the TLS working group. We also acknowledge that "getting a proof done" is not easy.

The TLS working group now needs to form its opinion on what to do with this draft.