Multipath TLS 1.3

Marc Fischlin, Sven-André Müller, Jean-Pierre Münch, and Lars Porth

Technische Universität Darmstadt, Darmstadt, Germany
`marc.fischlin@cryptoplexity.de`

Abstract. In a multipath key exchange protocol (Costea et al., CCS'18) the parties communicate over multiple connection lines, implemented for example with the multipath extension of TCP. Costea et al. show that, if one assumes that an adversary cannot attack all communication paths in an active and synchronized way, then one can securely establish a shared key under mild cryptographic assumptions. This holds even if classical authentication methods like certificate-based signatures fail. They show how to slightly modify TLS to achieve this security level.
Here we discuss that the multipath security can also be achieved for TLS 1.3 without having to modify the crypto part of protocol at all. To this end one runs a regular handshake over one communication path and then a key update (or resumption) over the other path. We show that this already provides the desired security guarantees. At the same time, if only a single communciation path is available, then one obtains the basic security properties of TLS 1.3 as a fall back guarantee.

1 Introduction

Secure connection establishment ultimately relies on the ability to authenticate the intended communication partner. Otherwise sensitive data may be transmitted to the wrong party, rendering any attempt to protect data-in-transit useless. Modern key establishment methods such as TLS therefore use various forms of authenticating the partner (unilaterally or mutually), ranging from shared secrets to the common certificate-based signatures.

However, the reliable binding of certified keys to identities is often hard to realize. These may be due to rogue certificates, issued to the wrong party such as in the Comodo and DigiNotar cases [20]. Another source of problems are misconfigured libraries which skip (parts of) the verification [14] or implementation errors as in Apple's `goto fail` [18]. Sometimes, connection proxies may also break up end-to-end connections and thereby weaken security, e.g., by insufficient certificate checks [4].

1.1 Multipath Key Exchange

Some solutions towards hedging against certificate misbinding have been proposed, including certificate pinning [10] to temporarily store known links, and certificate transparency [19] to log valid certificates. Recently, Costea et al. [5] discussed another possibility to enhance security by using the multipath extension of the TCP connection protocol (MPTCP) in [13]. Roughly, the multipath extension allows to establish further sub flows in a TCP connection to ensure reliable and possibly parallel data transmission over different communication channels (such as WiFi and mobile networks). While being primarily a tool for network efficiency, Costea et al. [5] point out that it can also be used to build *multipath key exchange* protocols.

In a multipath key exchange protocol the two parties send partial information of the key exchange protocol over different connections to create a shared key. One usually assumes that there are two connections available. The optimistic assumption is that an adversary can either be active on both connections but then cannot synchronize during the execution, called A/A adversary in [5]. This happens if the latency of the sub connections is small. Or, the adversary may be able to synchronize during the key establishment but then does not have means to actively attack both connections and thus only passively eavesdrop on one of the connections. This is called an $A{-}P$ attacker in [5].

Costea et al. [5] continue by designing a multipath key exchange scheme *SMKEX* based on the Diffie-Hellman problem. The protocol only requires a Diffie-Hellman exchange over one flow, and the exchange of nonces over the other flow, together with a hash confirmation value. No further authentication is required. They prove their protocol to be secure in a multipath variant of the Canetti-Krawcyzk (CK) model [3] in the random oracle model, against A/A and $A{-}P$ adversaries. In addition, they also comprehensively discuss the practical feasibility of the multipath approach, and how to modify the crypto part of TLS slightly to incorporate the enhanced security guarantees. The resulting protocol is called *MTLS*.

1.2 Our Contribution

We adopt the idea to relax the assumption about authentication guarantees by using multiple communication paths. We present here a TLS 1.3 compliant protocol [21] to enhance the security of the key establishment. The idea is to run a regular handshake execution over the MPTCP main flow, followed by the key update sub protocol of TLS 1.3 over the MPTCP sub flow. See Figure 1. The key update step renews the traffic secrets. Alternatively, one may run the resumption sub protocol of TLS 1.3 over the sub flow. The advantage of running the more expensive resumption step is that it updates all keys which TLS 1.3 established, including for example the resumption and exporter master secrets.

In comparison to the *SMKEX* and *MTLS* proposals in [5], our approach has some advantages:

- Our protocol works on top of existing TLS 1.3 implementation, without requiring any modifications of the cryptography. This is contrast to *SMKEX* which is built from scratch, and *MTLS* which modifies TLS slightly.
- Our protocol provides security against A/A and $A{-}P$ adversaries simultaneously, even if the TLS certificates are completely broken, relying on network

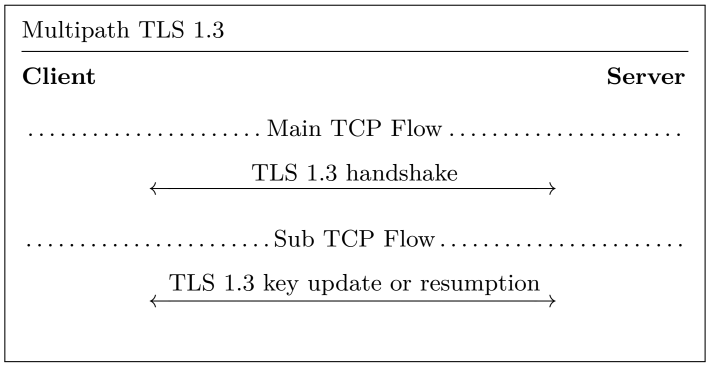


Fig. 1: MPTCP-TLS 1.3 Overview

assumptions instead. However, since it runs the basic TLS 1.3 mechanisms, even if the network assumptions turn out to be false, e.g., the parties exchange the information only over a single connection, then one still has the original TLS 1.3 security guarantees as fallback. *MTLS* in [5] is also considered to have this property.

– We discuss our approach concretely for TLS 1.3, but the idea of running the key exchange step over one flow, and then some form of key update or confirmation message over the other flow, should be applicable in general.

In terms of the security model, we introduce a multipath extension of the Bellare-Rogaway (BR) model [2,1]. The difference to the CK model [3] essentially is the latter allows for session-state reveals. But TLS 1.3 has not been designed to withstand such attacks and so far has been analyzed only in (multi-stage extensions [11] of) the BR model [8,9]. We note that we only consider security of the traffic secrets such that we restrict ourselves to a single-stage security model here. We also introduce some minor strengthenings compared to the model in [5].

We finally prove the TLS 1.3 (EC)DHE key exchange followed by a key update to be secure against A/A and $A{-}P$ adversaries in our security model. We do not rely on the random oracle assumption but need some standard assumptions about the Diffie-Hellman problem, the pseudorandomness of HKDF, and the integrity of the record protocol (which follows from the security of the AEAD schemes stipulated in TLS 1.3). In the $A{-}P$ case we also need a slightly stronger integrity assumption for the record protocol and discuss its plausibility.

2 Preliminaries

2.1 Multipath TCP

The MPTCP protocol [13] allows to establish multiple TCP subflows underneath an (MPTCP) connection. This allows for an improved and more reliable throughput. For establishing an MPTCP connection the initiator and responder start a regular TCP connection but use a special flag MP_CAPABLE, i.e., both sides agree on an MPTCP connection by setting the MP_CAPABLE flag in the TCP flow of SYN,SYN/ACK, and ACK messages. In the course of this the parties also pick random cryptographic keys and a locally unique 32-bit token, which are all transmitted (in clear) to the other side. The token is in fact a truncated hash value of the responder's key.

To open up a new subflow between addresses either party can start a new TCP connection, but this time include the MP_JOIN flag in the SYN,SYN/ACK,ACK flow. The link to the initial connection is via the token which is included in the MP_JOIN part. During the new establishment both parties exchange nonces, and authenticate both nonces via a (truncated) HMAC computation for the keys from the initial MPTCP connection. The nonces should prevent replay attacks.

While the deployment of MPTCP should be transparent for TCP-only connections, the sender of data over an MPTCP connection in principle has full control over the distribution of data through different sub flows. The routing can be set arbitrarily through the scheduler, albeit not all operating system may support arbitrary choices by default. The receiver may request to prioritize a sub flow via the MP_PRIO flag, and the sender should obey to this request. For our advanced security guarantees, however, we require that the second part of our key agreement protocol indeed runs over a fresh sub flow. If not then one falls back to the ordinary security of TLS 1.3 against active network attackers.

2.2 Transport Layer Security

We give a high-level overview of the Transport Layer Security (TLS) protocol, in particular version 1.3 [21]. Given that our focus in this work is on multipath connection security without authentication we omit the mechanisms for server and client authentication in the description here; our model and security proof still takes this part into account. Instead in the description here we focus on the main anonymous handshake, the record protocol, as well as the protocol to update the record layer keys for an existing connection. More details, which are especially relevant for the proof, appear in Appendix A.

The (EC)DHE handshake of TLS 1.3 runs a Diffie–Hellman-based key derivation. The client initiates the communication with its client hello message CH, including a nonce, and a client key share CKS carrying a Diffie-Hellman value. The server responds with its server hello SH message with its nonce, and its SKS part with a Diffie-Hellman value. The server computes the finished message SF, including a MAC under the derived key, and the client responds with its finished message CF.

For us, the most relevant part is key derivation. With a convoluted key derivation schedule based on the HKDF functions HKDF.Extract and HKDF.Expand, the parties compute (among others) a resumption master secret RMS, a client application traffic secret client_application_traffic_secret, and a server application traffic secret server_application_traffic_secret. The former key is used for the session resumption step only, and the latter keys are used to protect the

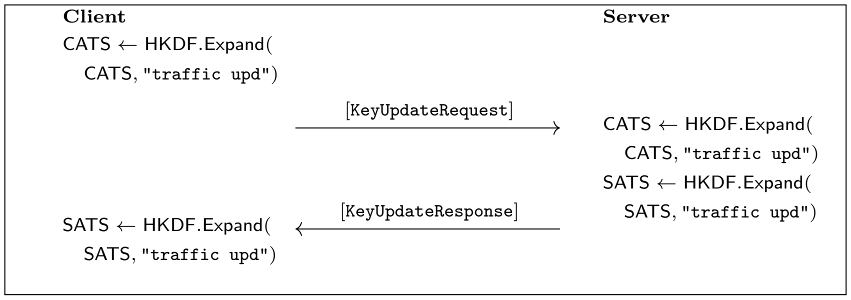

Client
Server
CATS ← HKDF.Expand(CATS, "traffic upd")
[KeyUpdateRequest]
CATS ← HKDF.Expand(CATS, "traffic upd")
SATS ← HKDF.Expand(SATS, "traffic upd")
[KeyUpdateResponse]
SATS ← HKDF.Expand(SATS, "traffic upd")

Fig. 2: The TLS 1.3 key update protocol. All messages are protected by the TLS record protocol using the current CATS and SATS, this is indicated by the square brackets.

communication via the record layer protocol (with an individual key for each sending party). We usually abbreviate the latter keys as CATS and SATS.

We omit the details about session resumption here and instead focus on the key update step. Figure 2 depicts the sub protocol to update the application traffic secrets, as well as the associated computations [21, Section 7.2]. In essence the initiator sends a fixed message requesting a key update and updates their sender secret which the responder is required to respond to with a fixed message, repeating the key updates as well as updating their keys.

Finally, the record layer protocol for TLS 1.3 enforces the use of an authenticated encryption with associated data (AEAD) scheme. It uses a secret IV as the initial nonce, derived from CATS or SATS, depending on whether the sender is the client or the server. The IV is derived as $\mathsf{HKDF.Expand}(\mathsf{CATS}, \texttt{"iv"})$ for the client, and analogously for the server and its traffic secret. The keys are derived similarly as $\mathsf{HKDF.Expand}(\mathsf{CATS}, \texttt{"key"})$ using a different label. The nonce is incremented with each sent message.

3 Security Model

3.1 Overview

We follow the description in [5] to motivate the different attacker models, especially $A{-}P$ and A/A. We always assume there are two communication paths between parties on which messages can be exchanged. The parties can choose the path for each message. On each path we assume there is one adversarial instance present, either active or passive. An *active* attacker may intercept and change messages. A *passive* attacker can monitor the communication between the two parties, but cannot modify it. Both types of adversaries can delay delivery of network messages at will.

We next distinguish between the communication between the different path attackers while a certain attacked execution is running. We let $X_1{-}X_2$ denote two path instances which can communicate arbitrarily during the execution, and X_1/X_2 to be two instances with restrictions. That is, let $X_1{-}X_2$ (resp.

X_1/X_2) denote a pair of *synchronized* (resp. *unsynchronized*) attackers, which can (resp. cannot) exchange information during the protocol execution. In both cases the attackers may exchange information before or after the protocol execution. The variable X_i can be either A for an *active* attacker (capable of altering messages) or P for a *passive* attacker.

Observe that we can disregard the scenario of $A{-}A$ where we have two synchronized active attackers. This scenario degenerates to a single attacker on a single path since the attackers can act as a single entity then. Also, according to our model $A{-}P$ and $P{-}A$ describe the same set of admissible adversaries. As pointed out by [5] it then suffices to consider types $A{-}P$ and A/A, synchronized adversaries with one passive party, and active but unsynchronized adversaries.

3.2 Security of Multi-Path Key Exchange

We next define security of multi-path key exchange by adopting the common game-based security models, similar to [5]. We assume that we have n parties $P_1, \ldots, P_n$ all running the key exchange protocol. Each party may receive a pair of public and secret keys. When executing the protocol both parties obtain a session key which can be used to secure subsequent data flow. In TLS 1.3 these session keys are actually pairs, consisting of the client_application_traffic_secret key (for the communication from client to server) and server_application_traffic_secret key (for the communication from server to client). It is usually assumed that in a genuine execution both parties derive identical session keys.

Attack Model. Neglecting the restrictions due to synchronization, the adversary against the key exchange protocol has full control over the network and can inject, modify, or drop network messages. It can interact with each party by initiating a session between parties P_i (as client) and P_j (as server) for administrative identifier id, and then send protocol messages to either of the two parties, receiving immediately the party's response. In addition the adversary can ask to reveal session keys, modeling leakage of session keys, e.g., if used in weak applications. The adversary can also corrupt parties in which case it receives the long-term secrets like the secret key or the PSK in TLS 1.3.

The goal of the adversary is to distinguish a genuine session key from a random string, significantly better than with the guessing probability of 1/2. For this the adversary can call a Test oracle which, initialized with a random bit $b \xleftarrow{\$} \{0,1\}$, returns either the session key (if $b = 0$) or an independently chosen random key (if $b = 1$), but answering queries consistently. There are usually some restrictions on testing a session, namely, that neither the session key of the tested session nor of its partner have been revealed. Here, partnering is usually defined by session identifiers sid.

To capture the different communication paths we distinguish between main and sub connection of a session id. That is, id.main denotes the identity of the initial connection and id.sub of the joined sub flow. We restrict ourselves to a single sub flow here but the model can be easily extended to handle more sub flows. To deal with the different attack models we assume that we have two adversarial instances, $\mathcal{A}_{\text{main}}$ and $\mathcal{A}_{\text{sub}}$, both initialized with independent randomness. The main adversary $\mathcal{A}_{\text{main}}$ can initialize new sessions, test or reveal session keys, and corrupt users, and can communicate (only) with the main part of a session. In contrast, $\mathcal{A}_{\text{sub}}$ can only interact with sessions with identifier sub via Send queries. The two algorithms can interact via special Sync oracle, which allows to pass arbitrary information between the two algorithms, and $\mathsf{Relinquish}$ to go idle and hand over control to the other adversarial instance (but passing no further information).

Formally, we assume that the adversary $\mathcal{A}_{\text{main}}$ can make the following queries during the attack:

- $\mathsf{NewSession}(P_i, P_j, \mathit{role})$ creates a new session for party P_i with role $\mathit{role} \in \{\mathit{client}, \mathit{server}\}$, supposedly communicating with P_j, picks a fresh administrative identifier id with two sub identifiers $\mathsf{id.main}$ and $\mathsf{id.sub}$, and returns id. One also creates entries $\mathsf{id.user} \leftarrow P_i$, $\mathsf{id.partner} \leftarrow P_j$, $\mathsf{id.role} \leftarrow \mathit{role}$, and $\mathsf{id.key} \leftarrow \perp$ for the session key. It notes its status as $\mathsf{id.status} \leftarrow \mathit{running}$ and holds two other entries $\mathsf{id.main.sid}$ and $\mathsf{id.sub.sid}$ for the session identifiers of the two flows (where $\mathsf{id.sid} = (\mathsf{id.main.sid}, \mathsf{id.sub.sid})$).
- $\mathsf{Send}(m, \mathsf{id.main})$ sends the next protocol message to the session with identity $\mathsf{id.main}$ (resp. drops the request and returns $\perp$ if no session with identifier id has been initialized). The message m may be of the special form init if the party is supposed to start the communication. The session is invoked for this protocol message and may return a protocol message (which is forwarded to the adversary). In addition, the session may change its status $\mathsf{id.status}$ to *accepted* or *rejected*. In the former case it also sets the session key $\mathsf{id.key}$ to some bit string and the session identifier $\mathsf{id.sid}$ to be (parts of) the ordered sequence of incoming and outgoing messages for each flow; details are provided as part of the protocol description.
  Analogously, adversary $\mathcal{A}_{\text{sub}}$ may call $\mathsf{Send}(m, \mathsf{id.sub})$, which is processed as above.
- $\mathsf{Reveal}(\mathsf{id})$ ignores the request if $\mathsf{id.status} \neq \mathit{accepted}$, else returns $\mathsf{id.key}$ and sets the status to $\mathsf{id.status} \leftarrow \mathit{revealed}$.
- $\mathsf{Corrupt}(P)$ returns the long-term signing key of the party P. We keep this oracle here for sake of compatibility with ordinary models, but since we are interested in trading authentication for multiple paths we will later assume that the adversary immediately corrupts all parties anyway.
- $\mathsf{Test}_b(\mathsf{id})$ ignores the request if $\mathsf{id.status} \neq \mathit{accepted}$, else returns $\mathsf{id.key}$ for $b = 0$ resp. a random string from $\{0,1\}^{|\mathsf{id.key}|}$ if $b = 1$, and sets the status to $\mathsf{id.status} \leftarrow \mathit{tested}$. We assume that Test is only called once during the attack (by $\mathcal{A}_{\text{main}}$) and denote the corresponding identity by $\mathsf{id}_{\mathsf{Test}}$.

Both adversaries $\mathcal{A}_{\text{main}}$ and $\mathcal{A}_{\text{sub}}$ have access to two additional oracles:

- $\mathsf{Sync}(x)$ can be called by either adversary and forwards x to the other adversary. This is immediately followed by a $\mathsf{Relinquish}$ execution.

- $\mathsf{Relinquish}()$ lets the other adversary become active (and the calling adversary inactive). We assume that initially $\mathcal{A}_{\mathrm{main}}$ is active and $\mathcal{A}_{\mathrm{sub}}$ inactive, and that only the active adversary can make oracle calls.

At the end of the execution algorithm $\mathcal{A}_{\mathrm{main}}$ outputs a guess $a \in \{0,1\}$ for the hidden bit b. We declare the adversary to lose if it tests the session $\mathsf{id}_{\mathsf{Test}}$ (with $\mathsf{id}_{\mathsf{Test}}.\mathsf{status} = \textit{tested}$) and reveals the session key of an honest partner, that is, if there is a session $\mathsf{id}' \neq \mathsf{id}_{\mathsf{Test}}$ such that $\mathsf{id}'.\mathsf{sid} = \mathsf{id}_{\mathsf{Test}}.\mathsf{sid}$ and $\mathsf{id}'.\mathsf{status} = \textit{revealed}$. If this happens we automatically set a Boolean variable $\mathsf{lose} \leftarrow \mathtt{true}$ (which initially is $\mathtt{false}$).

In addition, we also declare the adversary to lose if it violates the A/A or $A{-}P$ properties for the test session. For the former we let the *lifetime* of the test session $\mathsf{id}_{\mathsf{Test}}$ cover all the actions of the adversaries between the $\mathsf{NewSession}$ call which returned $\mathsf{id}_{\mathsf{Test}}$ and the call which changes the status to $\mathsf{id}_{\mathsf{Test}}.\mathsf{status} \neq \textit{running}$. Let $\mathcal{I}$ be the set of session identities $\mathsf{id}' \neq \mathsf{id}_{\mathsf{Test}}$ which $\mathcal{A}_{\mathrm{main}}$ either calls an oracle about or has received from a $\mathsf{NewSession}$ call during the lifetime of $\mathsf{id}_{\mathsf{Test}}$. Then we set $\mathsf{lose} \leftarrow \mathtt{true}$ unless

- there is another session $\mathsf{id}' \neq \mathsf{id}_{\mathsf{Test}}$ to the tested session $\mathsf{id}_{\mathsf{Test}}$ which is partnered in one of the flows, i.e., such that $\mathsf{id}'.\mathsf{main}.\mathsf{sid} = \mathsf{id}_{\mathsf{Test}}.\mathsf{main}.\mathsf{sid}$ or $\mathsf{id}'.\mathsf{sub}.\mathsf{sid} = \mathsf{id}_{\mathsf{Test}}.\mathsf{sub}.\mathsf{sid}$ ($A{-}P$ property satisfied), or
- there is no Sync call and at most one $\mathsf{Relinquish}$ call during the lifetime of $\mathsf{id}_{\mathsf{Test}}$, and no Send call of $\mathcal{A}_{\mathrm{sub}}$ to some identity $\mathsf{id}' \in \mathcal{I}$ (A/A property satisfied).

In the A/A case we forbid the adversary $\mathcal{A}_{\mathrm{sub}}$ to make any call to some "alive" session $\mathsf{id}' \in \mathcal{I}$. This prevents the adversary from communicating by, say, observing the behavior of other sessions. An example could be $\mathcal{A}_{\mathrm{main}}$ putting session id' into a certain state which triggers a certain response when $\mathcal{A}_{\mathrm{sub}}$ calls id' after the handover. This could allow $\mathcal{A}_{\mathrm{main}}$ to pass arbitrary bit strings to $\mathcal{A}_{\mathrm{sub}}$ while the test session is still active, thus violating the A/A property.

Note also that we do not make any stipulations about corruptions of party. The idea of the multipath extension of TLS is exactly to withstand attacks where no authentication happens, or where the adversary controls the long-term signing key used for authentications.

We have defined security with respect to a single-test setting, i.e., where the adversary can only test a single session during the attack. This simplifies the definition compared to a multi-test scenario where the same secret bit b is used in multiple Test calls of the adversary. In the latter case one would need to make the above stipulation for each test session, preventing the adversary from communicating for any of the tested sessions. Depending on the scheme it may then be possible to show via a hybrid argument that the multi-test case can be reduced to the single-test case.

Security Definitions. We always assume that two accepting sessions with the same session identifier also derive the same session key. This is always the case in TLS 1.3 and in the following we do not discuss this further. We also note that, because of the freshly chosen nonces, the probability of three honest parties deriving the same (sub) session identifiers is negligible. This is called Match-security:

Definition 1 (Match-Security). *For a multi-path key exchange protocol* KE *and adversary pair* $\mathcal{A} = (\mathcal{A}_{main}, \mathcal{A}_{sub})$ *in the experiment above let* $\mathbf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}}$ *be the probability that* $\mathcal{A}$ *manages to make three sessions have the same session identifier,* $\mathsf{id}, \mathsf{id}', \mathsf{id}''$ *with* $\mathsf{id.sid} = \mathsf{id}'.\mathsf{sid} = \mathsf{id}''.\mathsf{sid}$*, or that two sessions have the same session identifier but different keys,* $\mathsf{id.sid} = \mathsf{id}'.\mathsf{sid}$ *but* $\mathsf{id.key} \neq \mathsf{id}'.\mathsf{key}$*. The protocol is* Match*-secure if for any efficient adversary pair* $\mathcal{A} = (\mathcal{A}_{main}, \mathcal{A}_{sub})$ *the advantage is negligible.*

Next we define key secrecy for simultaneous A/A and $A{-}P$ attacks:

Definition 2 (Key Secrecy). *A multi-path key exchange protocol* KE *is simultaneously key-secret against* A/A *and* $A{-}P$ *adversaries if for any efficient adversary pair* $\mathcal{A} = (\mathcal{A}_{main}, \mathcal{A}_{sub})$ *in the experiment above*

$$\mathbf{Adv}^{A-P\&A/A\text{-}secrecy}_{\mathsf{KE},\mathcal{A}} := \mathrm{Prob}\left[\, a = b \wedge \neg\mathsf{lose}\right] - \tfrac{1}{2} \leq negl.$$

In comparison to previous models we make the following changes:

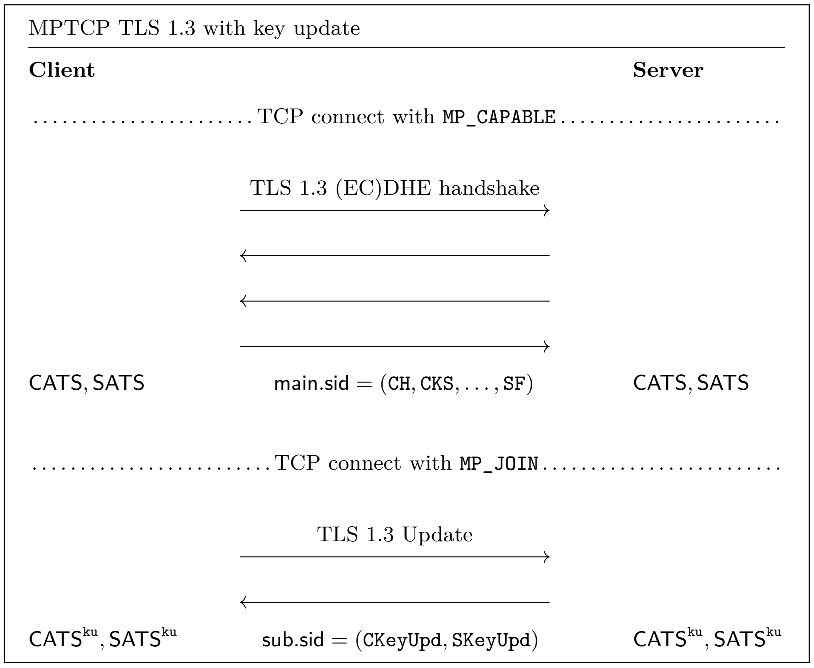
– The work by Costea et al. [5] models A/A adversaries by splitting the adversary when communicating with the test session. We split the adversary from the beginning but allow for an explicit information transfer through Sync queries (and disallow such queries when attacking the test session).
– Unlike [5] we do not enable session state reveals where the adversary receives the ephemeral randomness of the protocol participant. The reason is that TLS 1.3 does not account for such attacks.
– We account for security against $A{-}P$ and A/A simultaneously. That is, the adversary can decide during the attack on the type of attempt.
– Costea et al. [5] in the $A{-}P$ case explicitly consider adversaries which are passive on one of the two flows for the attacked session. Here we only demand that there exists a sub flow with some honest session, not necessarily in the same attacked session, where the adversary remains passive. Our model hence also captures cross-over attacks for different sessions.
– We do not consider multi-stage security of the TLS 1.3 session keys [11]. This notion is useful when one argues security of the intermediate keys derived during the handshake protocol, but we aim to protect the actual session keys $\mathsf{client\_application\_traffic\_secret}$ and $\mathsf{server\_application\_traffic\_secret}$ which are only derived at the very end.

4 Multipath Extension for TLS 1.3

4.1 Protocol

We present the MPTCP extension of the TLS 1.3 protocol in Figure 3. The client and server first execute a regular (EC)DHE handshake to derive application traffic secrets $\mathsf{CATS}, \mathsf{SATS}$. Then they run a key update on the added sub flow to derive the new keys $\mathsf{CATS}^{\mathsf{ku}}, \mathsf{SATS}^{\mathsf{ku}}$. Note that the protocol messages in the update step on the sub flow are still secured under the current keys, namely $\mathsf{client\_write\_iv} \leftarrow \mathsf{HKDF.Expand}(\mathsf{CATS}, \texttt{"iv"})$ for the initialization vector and $\mathsf{client\_write\_key} \leftarrow \mathsf{HKDF.Expand}(\mathsf{CATS}, \texttt{"key"})$ for the key for the client, and analogously for the server.

We view the multipath protocol execution as consisting of both flows. The protocol session accepts only after a successful key update, and only then status changes from *running* to *accepted*. The session key pair, which is subsequently used to protect communication, is the updated pair $\mathsf{CATS}^{\mathsf{ku}}, \mathsf{SATS}^{\mathsf{ku}}$. In particular, this means that Reveal queries of the adversary in the attack only make sense after completion of the sub flow. The adversary then receives the key pair $\mathsf{CATS}^{\mathsf{ku}}, \mathsf{SATS}^{\mathsf{ku}}$ but still does not have access to the intermediate key pair $\mathsf{CATS}, \mathsf{SATS}$.



MPTCP TLS 1.3 with key update
Client
Server
TCP connect with MP_CAPABLE
TLS 1.3 (EC)DHE handshake
CATS, SATS
main.sid = (CH, CKS, ..., SF)
CATS, SATS
TCP connect with MP_JOIN
TLS 1.3 Update
CATS^ku, SATS^ku
sub.sid = (CKeyUpd, SKeyUpd)
CATS^ku, SATS^ku

Fig. 3: Protocol Overview over MPTCP-TLS 1.3 with key update. The final session key(s) are the application-traffic-secrets $\mathsf{CATS}^{\mathsf{ku}}, \mathsf{SATS}^{\mathsf{ku}}$ after the key update. `CKeyUpd` and `SKeyUpd` denote the (secured) record-layer messages for the key update.

We note that the cryptographic part of TLS 1.3 remains unaltered. Only the socket interface to MPTCP would need to be changed, possibly enabling TLS 1.3 to demand a path change for the key update or resumption. Still, if for some reason MPTCP only runs plain TCP in backward compatibility mode, the network communication of our protocol looks like a common TLS 1.3 execution. Furthermore, even if the TLS application was not aware that the connection runs plain TCP, we would still have the basic security guarantees of TLS 1.3.

4.2 Security Assumptions

To show security we need several assumptions about the cryptographic primitives. We define them briefly below. Some assumptions like collision resistance of the hash function H are standard and can be found also in text books like [15]. For assumptions about the authenticated encryption with associated data (AEAD) in the record layer see [22]. We also need some slightly non-standard assumptions which nonetheless appear to be highly reasonable.

We let $\mathbf{Adv}^{\mathrm{DDH}}_{\mathbb{G},\mathcal{D}}(\lambda)$ be the advantage of an algorithm $\mathcal{D}$ deciding Diffie-Hellman values in the group $\mathbb{G}$. That is, $\mathbf{Adv}^{\mathrm{DDH}}_{\mathbb{G},\mathcal{D}}(\lambda)$ denotes the absolute difference between the probabilities that $\mathcal{D}$, on input a description of the group $\mathbb{G}$ with generator g and three values g^x, g^y, g^{xy} resp. g^x, g^y, g^z for random x, y, z, outputs 1. In our case we assume that $\mathbb{G}$ is the weakest of the elliptic curve groups of TLS 1.3.

We assume that the hash function H for computing the transcript hash is collision resistant. In other words, letting $\mathbf{Adv}^{\mathrm{coll}}_{\mathsf{H},\mathcal{C}}$ be the probability that an algorithm $\mathcal{C}$ outputs a collision $x \neq x'$ with $\mathsf{H}(x) = \mathsf{H}(x')$ is small.

We also assume that $\mathsf{HKDF.Extract}$ and $\mathsf{HKDF.Expand}$ are pseudorandom functions (for random inputs in the second input for $\mathsf{Extract}$ and in the first input for Expand, distributed according to some distribution D). That is, let $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HKDF.Extract},D,\mathcal{D}}$ for an algorithm $\mathcal{D}$ be the absolute difference in outputting 1 when having oracle access to $\mathsf{HKDF.Extract}(\mathsf{key}, \cdot)$ for $\mathsf{key} \xleftarrow{\$} D$ resp. to a random function with the same input-output size. Define $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HKDF.Expand},D,\mathcal{D}}$ analogously for function $\mathsf{HKDF.Expand}(\cdot, \mathsf{key})$.

For A/A attacks we sometimes even consider pseudorandomness of HKDF for partially adversarial chosen distributions $\mathsf{key} \leftarrow D(x; r)$ where an adversary can choose the input x after learning the distribution's randomness r. The distinguisher $\mathcal{D}$, however, does not get to learn x, r (such that the key still has high entropy) but only gets oracle access to $\mathsf{HKDF.Expand}(\cdot, \mathsf{key})$ or a random function. In other words, we assume that $\mathsf{HKDF.Expand}$ is a good extractor for the adaptively biased source D. This seems to be very plausbile given that HKDF was designed to have this property [16,17].

Finally, for the record layer protocol we assume that the probability of sending a protocol message through the record layer which is not rejected is infeasible. That is, let $\mathbf{Adv}^{\mathrm{int}}_{\mathsf{AEAD},\mathcal{B}}$ be the probability that an algorithm $\mathcal{B}$ first outputs a message m, then a key key is generated for the AEAD scheme, and an initial random nonce N_0 according to the record layer is picked. Then the adversary receives $C \leftarrow \mathsf{AEAD.Enc}(\mathsf{key}, N_0, m)$ and is supposed to output $C^* \neq C$ such that $\mathsf{AEAD.Dec}(\mathsf{key}, N_0, C^*) \neq \bot$. The fact that the adversary's success probability is small is implied by the common authentication or integrity assumption for AEAD schemes [22].

We also need a stronger but still reasonable assumption about the ability to use a different key key' and nonce N' to generate a valid record which can be successfully decrypted under the original key key and nonce N. We define this "correlation" property in combination with HKDF because the alternative key key' cannot be chosen directly but needs to be generated by calling HKDF, making attacks less likely. That is, for any adversary $\mathcal{E}$ define $\mathbf{Adv}^{\mathrm{corr}}_{\mathsf{AEAD},\mathsf{HKDF},\mathcal{E}}$ to be the probability that $\mathcal{E}$ outputs $(MS, x) \neq (MS', x')$, and m such that the following holds: Let $\mathsf{CATS} \leftarrow \mathsf{HKDF.Expand}(MS, x)$, $\mathsf{key} \leftarrow \mathsf{HKDF.Expand}(\mathsf{CATS}, \texttt{"key"})$, $N \leftarrow \mathsf{HKDF.Expand}(\mathsf{CATS}, \texttt{"iv"})$, as well as $\mathsf{CATS}' \leftarrow \mathsf{HKDF.Expand}(MS', x')$, $\mathsf{key}' \leftarrow \mathsf{HKDF.Expand}(\mathsf{CATS}', \texttt{"key"})$, $N' \leftarrow \mathsf{HKDF.Expand}(\mathsf{CATS}', \texttt{"iv"})$, $C' \leftarrow \mathsf{AEAD.Enc}(\mathsf{key}', N', m)$, and check that $\mathsf{AEAD.Dec}(\mathsf{key}, N, C') \neq \perp$.

The assumption appears to hold for common AEAD schemes. If we assume that HKDF behaves like a random oracle then the different inputs $(MS, x) \neq (MS', x')$ yield independently distributed outputs. But then the probability that two random key-nonce combinations can be used to encrypt and successfully decrypt is unlikely. Otherwise one could attack the AEAD scheme by trying to decrypt with a fresh random key-nonce pair and succeed with high probability.

4.3 Security

We first show Match-security. Note that we count the number s of sessions via the $\mathsf{NewSession}$ calls of the (main) adversary, and the (full) session identifiers consist of both sub identifiers.

Proposition 1. *The MPTCP TLS 1.3 extension, (EC)DHE handshake with key update, is* Match*-secure. More precisely, for any adversary $\mathcal{A}$ initiating a maximum number s of sessions and for nonce length $|nonce| = 256$ we have* $\mathbf{Adv}^{\mathsf{Match}}_{\mathsf{KE},\mathcal{A}} \leq s^2 \cdot 2^{-|nonce|}$.

Proof. The property follows as for regular TLS 1.3 in [8,9]. The probability that there are three sessions among the s sessions with the same sid is bounded from above by $s^2 \cdot 2^{-|\mathrm{nonce}|}$, since the probability that an honest party picks the same nonce as the (potentially partnered) other two sessions is given by the birthday bound. The fact that the same sid yields the same key follows straightforwardly, because the session identifier contains all information which enters the key derivation for CATS and SATS and if this key is identical, then then the same update messages $\texttt{CKeyUpd}, \texttt{SKeyUpd}$ also cause the same update step to $\mathsf{CATS}^{\mathsf{ku}}$ and $\mathsf{SATS}^{\mathsf{ku}}$. □

Theorem 1. *The MPTCP TLS 1.3 extension, (EC)DHE handshake with key update, is simultaneously key-secret against $A{-}P$ and A/A adversaries. More precisely, for any adversary $\mathcal{A} = (\mathcal{A}_{main}, \mathcal{A}_{sub})$ initiating at most s sessions we have that there are algorithms $\mathcal{D}, \mathcal{D}_1, \ldots, \mathcal{D}_{16}$, $\mathcal{D}'$, $\mathcal{B}$, $\mathcal{B}'$, $\mathcal{C}$, $\mathcal{E}$ and distributions*

$D_1, \ldots, D_{16}, D'$ with

$$\begin{aligned}&\mathbf{Adv}^{A-P\&A/A\text{-}secrecy}_{\mathsf{KE},\mathcal{A}}\\ &\quad\le s^2 \cdot \Big(\mathbf{Adv}^{DDH}_{\mathbb{G},\mathcal{D}} + \sum_{i=1}^{16} \mathbf{Adv}^{prf}_{\mathsf{HKDF.Extract/Expand},D_i,\mathcal{D}_i} + 2\cdot \mathbf{Adv}^{int}_{\mathsf{AEAD},\mathcal{B}}\\ &\qquad + \mathbf{Adv}^{coll}_{\mathsf{H},\mathcal{C}} + \mathbf{Adv}^{corr}_{\mathsf{AEAD},\mathsf{HKDF},\mathcal{E}}\\ &\qquad + \mathbf{Adv}^{prf}_{\mathsf{HKDF.Expand},D',\mathcal{D}'} + \mathbf{Adv}^{corr}_{\mathsf{AEAD},\mathsf{HKDF},\mathcal{E}} \Big).\end{aligned}$$

Here the other algorithms have roughly the same run time as $\mathcal{A}$ plus the time to execute the attack on the key exchange protocol.

Proof. Consider an adversary $\mathcal{A} = (\mathcal{A}_{\text{main}}, \mathcal{A}_{\text{sub}})$ against the key secrecy of the key exchange protocol. We discuss first the $A{-}P$ case. That is, the adversary may be active in one flow and passive in the other one. More formally, assume that for the test session $\mathsf{id}_{\mathsf{Test}}$ there exists another session id' with the same session identifier in either the main or sub flow. We assume that we know the right sessions $\mathsf{id}_{\mathsf{Test}}$ and id' in advance; this can be accomplished by guessing the sessions with probability at least $1/s^2$ among all s sessions.

Further note that we can make all $\mathsf{Corrupt}$ queries at the outset, such that the adversary immediately knows the signing keys. This is valid since key secrecy does not depend on authentication. Note that this in particular means that $\mathcal{A}$ could simulate all other sessions different from $\mathsf{id}_{\mathsf{Test}}$ and id' itself.

An important observation for the proof steps below is to note, once more, that Reveal queries of the adversary only make sense after the successful update step. Then the status changes to *accepted* and the Reveal queries returns the updated key pair $\mathsf{CATS}^{\mathsf{ku}}, \mathsf{SATS}^{\mathsf{ku}}$ (but not the keys $\mathsf{CATS}, \mathsf{SATS}$). We will take advantage of this observation multiple times below.

We next make a case distinction, depending on whether the main or sub flow of $\mathsf{id}_{\mathsf{Test}}$ and id' match:

Case A: Passive in main flow, $\mathsf{id}'.\mathsf{main.sid} = \mathsf{id}_{\mathsf{Test}}.\mathsf{main.sid}$. In this case we argue that the session key $\mathsf{CATS}, \mathsf{SATS}$ in the two sessions is secure. To see this we can perform a sequence of game hops, where we let $G_{A.i}$ denote the event that $\mathcal{A}$ wins in the corresponding game.

Game $G_{A.0}$. Is the original attack, with the simplifications about knowing $\mathsf{id}_{\mathsf{Test}}$ and id' at the beginning and corrupting all long-term keys at the outset.

Game $G_{A.1}$. Modify the game and replace the internally used Diffie-Hellman value g^{xy} in the two main executions of $\mathsf{id}_{\mathsf{Test}}$ and id' by a random value g^z. A simple reduction to the DDH problem shows that this cannot decrease the adversary's success probability in the key secrecy game $G_{A.1}$ by more than the advantage against the DDH problem:

$$\text{Prob}[G_{A.0}] \le \text{Prob}[G_{A.1}] + \mathbf{Adv}^{\text{DDH}}_{\mathbb{G},\mathcal{D}}.$$

The reduction $\mathcal{D}$ receives (g^x, g^y, g^z) as input, runs the entire key exchange attack with $\mathcal{A}$, also picking the test bit b, and inserts g^x and g^y into the test session $\mathsf{id}_{\mathsf{Test}}$ as well as the partner session id' on the honest parties side. But when both parties are supposed to compute g^{xy} reduction $\mathcal{D}$ uses g^z instead. Eventually, $\mathcal{D}$ checks if $\mathcal{A}$ succeeds in predicting b and does not lose. Algorithm $\mathcal{D}$ outputs 1 if this is the case. Note that if $g^z = g^{xy}$ we perfectly simulate $G_{A.0}$ whereas for a random g^z we perfectly simulate $G_{A.1}$. It follows that the difference in probabilities is bounded by the advantage against the DDH problem (for the admissible group $\mathbb{G}$ used in the execution).

Game $G_{A.2}$. Replace all output values in the HKDF evaluations (after, and including the computation of $HS \leftarrow \mathsf{HKDF.Extract}(xES, g^z)$) in the two main executions of sessions $\mathsf{id}_{\mathsf{Test}}$ and id' by random values. This includes the Expand calls to derive SS, $\mathsf{server\_finished\_key}$, CS, $\mathsf{client\_finished\_key}$, xHS, RMS and SATS as well as CATS, but also the $\mathsf{Extract}$ step to compute MS. Finally, we also replace the derived data $\mathsf{client\_write\_key}, \mathsf{client\_write\_iv}$, and $\mathsf{CATS}^{\mathsf{ku}}$ from CATS, and $\mathsf{server\_write\_key}, \mathsf{server\_write\_iv}$, and $\mathsf{SATS}^{\mathsf{ku}}$ from SATS by random values. Note that we can already replace the keys $\mathsf{CATS}^{\mathsf{ku}}$ and $\mathsf{SATS}^{\mathsf{ku}}$ as if they were computed, although we have not yet shown that they are actually derived; this will be shown below.

The proof replaces all the key values step wise, starting with computation of HS from the input source g^z, such that we can argue that the derivation of SS from the now random HS via Expand can be substituted by picking SS randomly etc. In each of the in total 16 steps we have an input distribution D_i and a distinguisher $\mathcal{D}_i$ such that we can show that the winning difference in each step is bounded by $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HKDF.Extract/Expand},\mathcal{D}_i}$. Altogether we thus have

$$\mathrm{Prob}[G_{A.1}] \leq \mathrm{Prob}[G_{A.2}] + \sum_{i=1}^{16} \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{HKDF.Extract/Expand},D_i,\mathcal{D}_i}.$$

In particular, we now have that CATS and SATS and the channel key-iv values derived from them, as well as $\mathsf{CATS}^{\mathsf{ku}}$ and $\mathsf{SATS}^{\mathsf{ku}}$, are random keys which are independent of the protocol messages between $\mathsf{id}_{\mathsf{Test}}$ and id'.

Game $G_{A.3}$. Declare the adversary to lose if it successfully executes the key update in the sub flow of session $\mathsf{id}_{\mathsf{Test}}$ or id' with $\mathsf{id}_{\mathsf{Test}}.\mathsf{sub.sid} \neq \mathsf{id}'.\mathsf{sub.sid}$. Note that the adversary can only make any of the two sessions accept if it sends a valid record layer message to the corresponding party, either under the now random channel key-nonce pair $\mathsf{client\_write\_key}, \mathsf{client\_write\_iv}$ or $\mathsf{server\_write\_key}$, $\mathsf{server\_write\_iv}$. The adversary may first receive a message under the other key from the honest client or server before producing a successful forgery against the other party's key. We can simulate this by a single query before creating the forgery which is admissible in the integrity game. But then we give a reduction $\mathcal{B}$ to the security of either of the two keys, such that

$$\mathrm{Prob}[G_{A.2}] \leq \mathrm{Prob}[G_{A.3}] + 2 \cdot \mathbf{Adv}^{\mathrm{int}}_{\mathsf{AEAD},\mathcal{B}}.$$

Here we use that Reveal queries do not reveal the intermediate keys and only give reasonable answers after completion of the entire execution.

We finally note that, in this game, sessions $\mathsf{id}_{\mathsf{Test}}$ and id' can only complete the sub flow execution if the adversary relays the communication between the two sessions which update the keys to $\mathsf{CATS}^{\mathbf{ku}}$ and $\mathsf{SATS}^{\mathbf{ku}}$. In particular, the adversary cannot Reveal the session key in session id' since it is partnered with the test session in both flows.

In the final game the adversary has no advantage to predict the secret bit b because this game does not depend on b anymore; the final session keys are independent random values in both cases. It follows that $\mathrm{Prob}[G_{A.3}] \leq \frac{1}{2}$.

Case B: Passive in sub flow, $\mathsf{id}'.\mathsf{sub}.\mathsf{sid} = \mathsf{id}_{\mathsf{Test}}.\mathsf{sub}.\mathsf{sid}$. Note that this stipulates that $\mathsf{id}'.\mathsf{main}.\mathsf{sid} \neq \mathsf{id}_{\mathsf{Test}}.\mathsf{sub}.\mathsf{sid}$ or else we are again in case A. But then, since the session identifier in the main flow contain exactly the data entering the transcript hash, we can conclude that key derivation uses different inputs, at least if we assume collision resistance of the hash function:

$G_{B.0}$. Is the simplified starting attack as above.

$G_{B.1}$. As game $G_{B.0}$ but declare the adversary to lose if $\mathsf{H}(\mathsf{id}_{\mathsf{Test}}.\mathsf{main}.\mathsf{sid}) = \mathsf{H}(\mathsf{id}'.\mathsf{main}.\mathsf{sid})$. This would immediately contradict the collision-resistance, i.e., we can give a reduction $\mathcal{C}$ such that

$$\mathrm{Prob}[G_{B.0}] \leq \mathrm{Prob}[G_{B.1}] + \mathbf{Adv}^{\mathrm{coll}}_{\mathsf{H},\mathcal{C}}.$$

$G_{B.2}$. As game $G_{B.1}$ but declare the adversary to lose if $\mathsf{id}_{\mathsf{Test}}$ or id' accept. We can again give a reduction $\mathcal{E}$ against the correlation intractability of the AEAD scheme (in combination with HKDF). Adversary $\mathcal{E}$ can impersonate the client resp. server in the sessions $\mathsf{id}_{\mathsf{Test}}$ and id' such that it knows the keys MS_{Test} and MS' on both sides. These keys may or may not match. But for sure the transcript hashes do not match by the previous game hop, such that we obtain key derivation inputs $(MS_{\mathsf{Test}}, x) \neq (MS', x')$ which the (relayed) sub flow would make both sides accept. This, however, would contradict the correlation integrity of the AEAD scheme:

$$\mathrm{Prob}[G_{B.1}] \leq \mathrm{Prob}[G_{B.2}] + \mathbf{Adv}^{\mathrm{corr}}_{\mathsf{AEAD},\mathsf{HKDF},\mathcal{E}}.$$

In the final game it follows that neither party $\mathsf{id}_{\mathsf{Test}}$ nor id' has accepted, such that the adversary cannot do any better than guessing the bit b:

$$\mathrm{Prob}[G_{B.2}] \leq \tfrac{1}{2}.$$

Case C: Active in both flows but acting independently. Finally, we need to argue that A/A attacker cannot predict the bit b significantly beyond guessing it. For this consider the test session $\mathsf{id}_{\mathsf{Test}}$ as before. We assume that there is no other flow with identical session identifier, neither in the main step nor in the sub flow. Else we would be already in cases A or B.

First note that, once the test session $\mathsf{id}_{\mathsf{Test}}$ has been started by $\mathcal{A}_{\text{main}}$, the sub adversaries cannot exchange information through Sync calls, nor via any other oracle calls during the lifetime of session $\mathsf{id}_{\mathsf{Test}}$. It follows that the main flow uses random inputs such as the party's nonce to compute the transcript hash $\mathsf{H}(\mathtt{CH}||\mathtt{SH}||\dots)$. We can therefore cast this input to $\mathsf{HKDF.Expand}$ via some distribution $D'(x;r)$ where the r part describes the honest party's contribution to the transcript, and x the contribution of $\mathcal{A}_{\text{main}}$, possibly chosen adaptively. Note that, while the transcript hash has no entropy from $\mathcal{A}_{\text{main}}$'s point of view, for $\mathcal{A}_{\text{sub}}$ it is still unknown, because no information flows from $\mathcal{A}_{\text{main}}$ to $\mathcal{A}_{\text{sub}}$.

We next define the following game hops:

$G_{C.0}$. Is the attack as above.

$G_{C.1}$. As game $G_{C.0}$ but declare $\mathcal{A}$ to lose if the sub flow in $\mathsf{id}_{\mathsf{Test}}$ accepts.

It follows from the pseudorandomness of $\mathsf{HKDF.Expand}$ for the transcript-hash input distribution D' above that we can replace CATS and SATS computed over the transcript in the moment when $\mathcal{A}_{\text{sub}}$ is active in $\mathsf{id}_{\mathsf{Test}}$ by random values. We argue that the probability that $\mathcal{A}_{\text{sub}}$ makes the sub flow accept cannot change significantly, else we derive a contradiction to the pseudorandomness of $\mathsf{HKDF.Expand}$ via some reduction $\mathcal{D}'$. Note that for this we merely need to wait for $\mathcal{A}_{\text{sub}}$ to make the sub flow accept or to hand over to $\mathcal{A}_{\text{main}}$ again (or abort the execution).

Once we have replaced the traffic application secrets by random values we immediately get a reduction $\mathcal{B}'$ to the integrity of AEAD, as in Case A. Hence,

$$\text{Prob}[G_{C.0}] \leq \text{Prob}[G_{C.1}] + \mathbf{Adv}^{\text{prf}}_{\mathsf{HKDF.Expand},D',\mathcal{D}'} + \mathbf{Adv}^{\text{corr}}_{\mathsf{AEAD},\mathsf{HKDF},\mathcal{B}'}.$$

In the final game the adversary $\mathcal{A}_{\text{sub}}$ does not make the sub flow accept. It follows that $\mathcal{A}$ does not learn any information about b from the Test query (since the session key is not set). It follows that the probability of predicting b is bounded by $\frac{1}{2}$.

Summing over all possibilities yields the claimed bound. $\square$

4.4 Sub Flow Resumption

Instead of using the simple key update procedure we may alternatively use resumption to update the keys over the sub flow. The advantage is that all TLS 1.3 keys are updated by this, not only the application traffic secrets. The security argument would be very similar to the one above, only that we need to take the key RMS into account.

A noteworthy point is that we can actually relax the correlation integrity condition on the AEAD scheme if we run resumption in the mode with the (EC)DHE step. In this case the sub flow would create a fresh Diffie-Hellman share in the sub flow as well, and we can argue that in Case B with relayed sub flow the adversary thus cannot distinguish the derived keys from random, just as we argue along the DDH assumption for passive adversaries in the main flow.

4.5 Practical Considerations

We discuss here some aspects when running the above multipath TLS 1.3 version. The first thing to note is that, in order to take advantage of the stronger security guarantees, the parties need to ensure that the communication of the update step is routed through the second communication channel. Luckily, even if the parties are not aware of this, or cannot ensure this, they can still rely on the basic security of TLS 1.3. Hence, from a security viewpoint the failure of using multiple communication lines does not make our protocol insecure.

Concerning efficiency observe that MPTCP in principle allows for parallel communication through the different channels. Our protocol, on the other hand, needs to complete the regular handshake execution before being able to run the update via the other communication line.

Next suppose that the key update step fails—if the handshake already fails then the protocol execution cannot be continued. At this point the two parties have already established a joint key via the handshake part. It may thus be tempting to still use that key. For security reasons and for compatibility it is nevertheless recommend to cautiously follow the TLS 1.3 specification [21] that the connection must be be closed.

5 Conclusions

We have shown that update steps in key exchange protocols can be used to provide multipath security. We have discussed this specifically for the case of TLS 1.3, assuming that one can reliably assign protocol messages to communication paths. An interesting question is to analyze what kind of security can still be achieved if some of the messages may be unexpectedly transmitted over the other path. Also, we have not investigated the possibility of 0RTT modes and the security of the intermediate session keys. Note that any cryptographic analysis of 0RTT modes must take into account the possibility of replay attacks [12].

Our security analysis provides a non-tight security bound with respect to the underlying cryptographic primitives, in the sense that the key secrecy bound depends quadratically on the number s of sessions. Furthermore, we work in the single-test setting, and a potential hybrid argument to extend this to the multi-test setting would incur another factor s. Recent efforts for TLS 1.3 [6,7], however, have shown that it is possible to derive tight security bounds. It would be interesting to see if this isapplicable here in the split adversary model as well.

Another interesting question is how smoothly one can use multipath connections. The work by Costea et al. [5] already provides a comprehensive set of experiments, indicating that it is doable in practice. It remains to investigate if this is also true for applications which rely on fast connection times of TLS 1.3, inciting the development of 0RTT modes.

Acknowledgments. We thank the anonymous reviewers for valuable comments. Marc Fischlin has been [co-]funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297.

References

1. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding. Lecture Notes in Computer Science, vol. 1807, pp. 139–155. Springer (2000)
2. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 773, pp. 232–249. Springer (1993)
3. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. Lecture Notes in Computer Science, vol. 2045, pp. 453–474. Springer (2001)
4. de Carné de Carnavalet, X., Mannan, M.: Killed by proxy: Analyzing client-end TLS interception software. In: 23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016. The Internet Society (2016)
5. Costea, S., Choudary, M.O., Gucea, D., Tackmann, B., Raiciu, C.: Secure opportunistic multipath key exchange. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 2077–2094. ACM (2018)
6. Davis, H., Günther, F.: Tighter proofs for the SIGMA and TLS 1.3 key exchange protocols. IACR Cryptol. ePrint Arch. **2020**, 1029 (2020), `https://eprint.iacr.org/2020/1029`
7. Diemert, D., Jager, T.: On the tight security of TLS 1.3: Theoretically-sound cryptographic parameters for real-world deployments. IACR Cryptol. ePrint Arch. **2020**, 726 (2020), `https://eprint.iacr.org/2020/726`
8. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 1197–1210. ACM (2015)
9. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the tls 1.3 handshake protocol. Cryptology ePrint Archive, Report 2020/1044 (2020), `https://eprint.iacr.org/2020/1044`
10. Evans, C., Palmer, C., Sleevi, R.: Public Key Pinning Extension for HTTP. RFC 7469 (Apr 2015), `https://rfc-editor.org/rfc/rfc7469.txt`
11. Fischlin, M., Günther, F.: Multi-stage key exchange and the case of google's QUIC protocol. In: Ahn, G., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. pp. 1193–1204. ACM (2014)
12. Fischlin, M., Günther, F.: Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. pp. 60–75. IEEE (2017). https://doi.org/10.1109/EuroSP.2017.18, `https://doi.org/10.1109/EuroSP.2017.18`

13. Ford, A., Raiciu, C., Handley, M.J., Bonaventure, O., Paasch, C.: TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684 (Mar 2020), `https://rfc-editor.org/rfc/rfc8684.txt`
14. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. pp. 38–49. ACM (2012)
15. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014)
16. Krawczyk, D.H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (May 2010), `https://rfc-editor.org/rfc/rfc5869.txt`
17. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6223, pp. 631–648. Springer (2010)
18. Langley, A.: Apple's ssl/tls bug. ImperialViolet (2014), `https://www.imperialviolet.org/2014/02/22/applebug.html`
19. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. RFC 6962 (Jun 2013), `https://rfc-editor.org/rfc/rfc6962.txt`
20. Menn, J.: E-mail breach in Iran raises surveillance fears. Financial Times, August 31 (2011)
21. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Aug 2018), `https://rfc-editor.org/rfc/rfc8446.txt`
22. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002. pp. 98–107. ACM (2002)

A Transport Layer Security

Figure 4 depicts the basic TLS 1.3 anonymous (EC)DHE handshakes including the essential steps of the Diffie–Hellman-based key derivation. The key update step has already been explained in Section 2.2. A session resumption is similar to the handshake but adds some additional steps. It requires the server to have issued a ticket to the client containing a nonce and identifying information which are used for the resumption handshake. The client uses an additional extension `ClientPreSharedKey` in the first message to indicate potential identifiers. The server acknowledges one in its `ServerPreSharedKey` extension with the second message. The parties then use the resumption secret RMS from before to compute a pre-shared key PSK, which this time enters the computation $ES \leftarrow \mathsf{HKDF.Extract}(\texttt{""}, PSK)$. They also derive a binder key BK which is used to verify the key. From there on the steps are identical to the one of a handshake execution. We note that resumption can be executed with and without the Diffie-Hellman step.

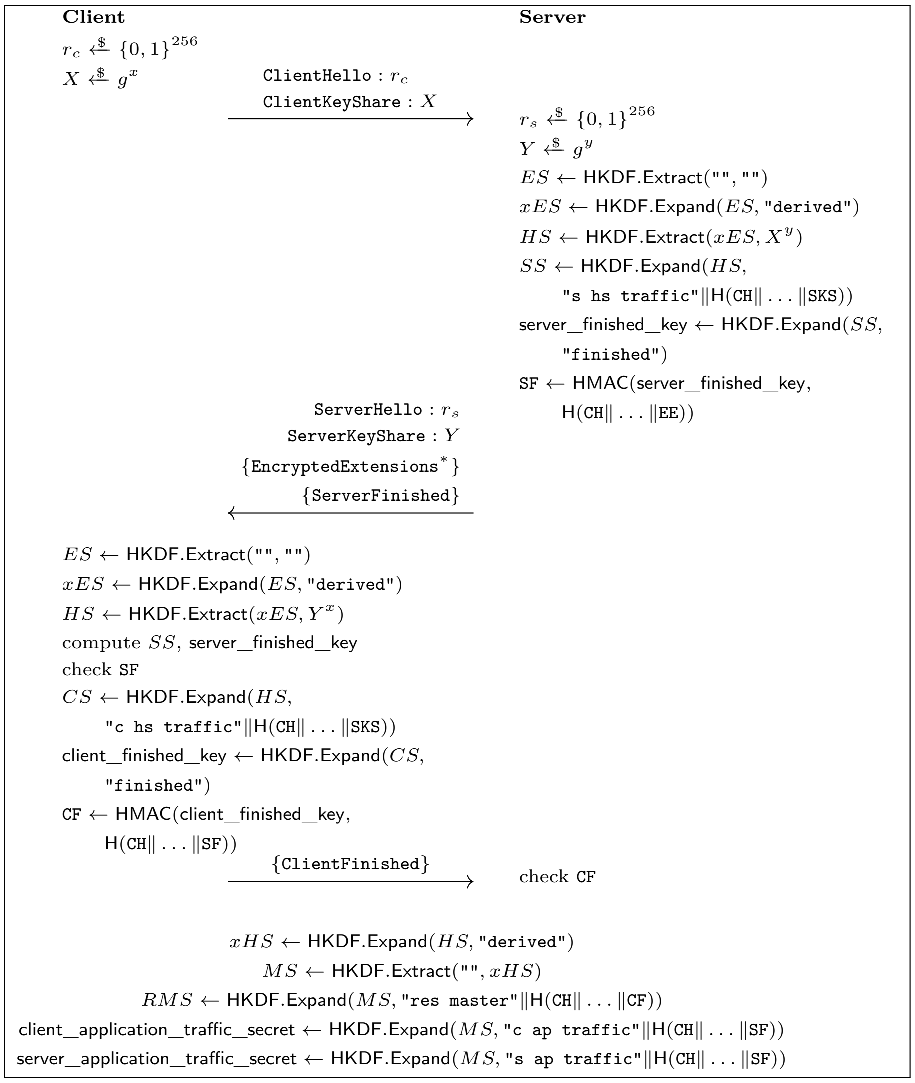

Client
Server
$r_c \xleftarrow{\$} \{0,1\}^{256}$
$X \xleftarrow{\$} g^x$
ClientHello : r_c
ClientKeyShare : X
$r_s \xleftarrow{\$} \{0,1\}^{256}$
$Y \xleftarrow{\$} g^y$
$ES \leftarrow \mathsf{HKDF.Extract}(\texttt{""}, \texttt{""})$
$xES \leftarrow \mathsf{HKDF.Expand}(ES, \texttt{"derived"})$
$HS \leftarrow \mathsf{HKDF.Extract}(xES, X^y)$
$SS \leftarrow \mathsf{HKDF.Expand}(HS, \texttt{"s hs traffic"} \| \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{SKS}))$
$\mathsf{server\_finished\_key} \leftarrow \mathsf{HKDF.Expand}(SS, \texttt{"finished"})$
$\texttt{SF} \leftarrow \mathsf{HMAC}(\mathsf{server\_finished\_key}, \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{EE}))$
ServerHello : r_s
ServerKeyShare : Y
{EncryptedExtensions*}
{ServerFinished}
$ES \leftarrow \mathsf{HKDF.Extract}(\texttt{""}, \texttt{""})$
$xES \leftarrow \mathsf{HKDF.Expand}(ES, \texttt{"derived"})$
$HS \leftarrow \mathsf{HKDF.Extract}(xES, Y^x)$
compute SS, server_finished_key
check SF
$CS \leftarrow \mathsf{HKDF.Expand}(HS, \texttt{"c hs traffic"} \| \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{SKS}))$
$\mathsf{client\_finished\_key} \leftarrow \mathsf{HKDF.Expand}(CS, \texttt{"finished"})$
$\texttt{CF} \leftarrow \mathsf{HMAC}(\mathsf{client\_finished\_key}, \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{SF}))$
{ClientFinished}
check CF
$xHS \leftarrow \mathsf{HKDF.Expand}(HS, \texttt{"derived"})$
$MS \leftarrow \mathsf{HKDF.Extract}(\texttt{""}, xHS)$
$RMS \leftarrow \mathsf{HKDF.Expand}(MS, \texttt{"res master"} \| \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{CF}))$
$\mathsf{client\_application\_traffic\_secret} \leftarrow \mathsf{HKDF.Expand}(MS, \texttt{"c ap traffic"} \| \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{SF}))$
$\mathsf{server\_application\_traffic\_secret} \leftarrow \mathsf{HKDF.Expand}(MS, \texttt{"s ap traffic"} \| \mathsf{H}(\texttt{CH}\| \ldots \|\texttt{SF}))$

Fig. 4: The TLS 1.3 anonymous (EC)DHE handshake protocol. Starred messages are situation-dependent and not always sent. Messages enclosed in curly brackets are protected by the handshake traffic secrets CS resp. SS.