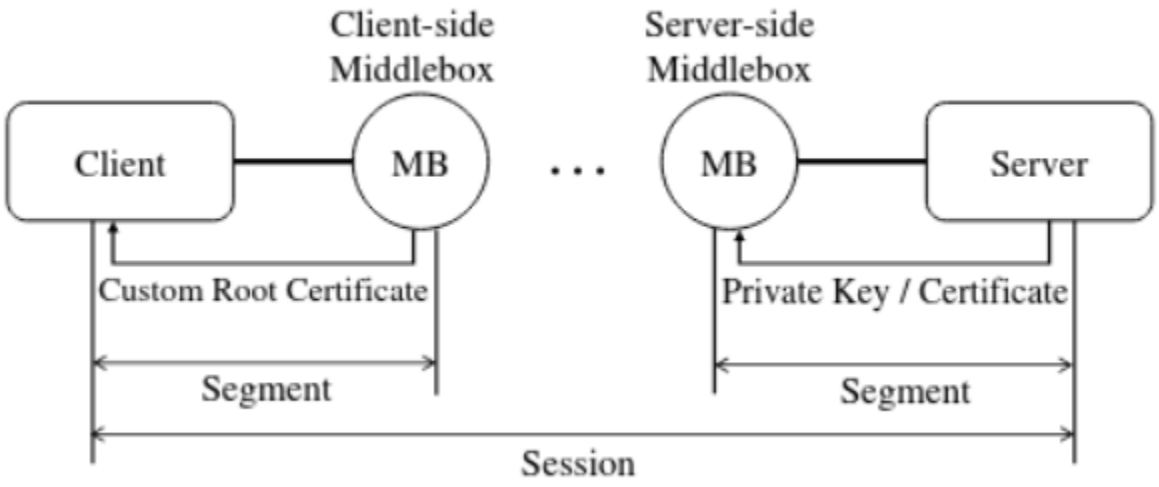
Split TLS :



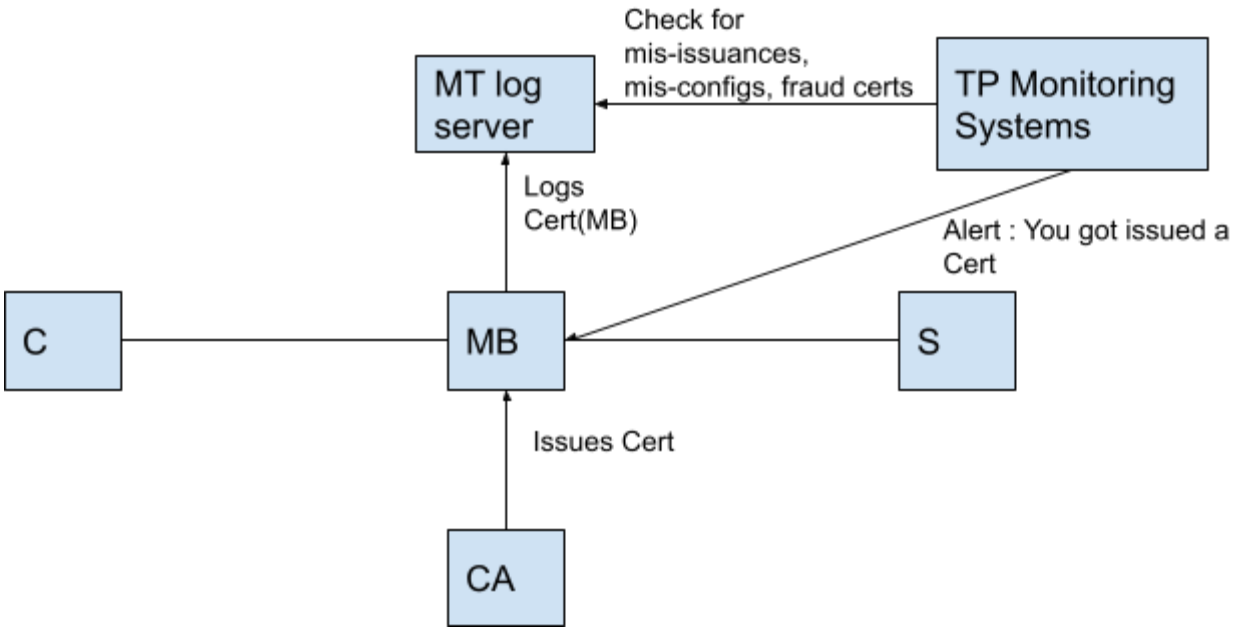
Security problems in SplitTLS :

Some MBs fail to correctly validate the certificates, degrade to weaker CCs or insert malicious scripts.

1. **Authentication** : Client cannot auth sv as MB replaces sv's cert with a forged one by MB.
2. **Confidentiality** : C negotiates key with MB and not the sv.
   C—(strong cc)-----MB —-(weak cc)----------MB—----(weak cc)----------S
3. **Integrity** : MBs can read and modify.(but that's not the issue) Issue : C cannot detect any modification done by a MB to his msg sent to S. (Someone can become MITM and inject without being noticed by C).

Solution : Make MBs visible to C and publicly auditable.

Auditable MBs :



MB cert :

- Need : Behaviors like - Role of MBs (eg. FWs), Permissions (r/w)
- Implementation : +Ext **MB_InfoAccess : MB_Description { MB_InfoType(eg.permission), MB_Info(eg. Read only)}**

MT :

……(yet to understand in depth).....

Properties of Auditable MBs :

1. No longer need SplitTLS(install cert/ share priv key).
2. C can detect if a MB has modified traffic without authorization. (Can check permission item in MBs cert).
3. MB cert's private key is no longer safe -> OSCP, CRL

---

maTLS :

TLS 1.2 + DHE

Security Goals :

1. Authentication :
    a. Sv + all MBs[called explicit Auth]. Helps with accountability.
    b. Achieved by Certs

2. Confidentiality :

    a. each maTLS segment ->
        i. Segment Secrecy : high TLS ver, Strong CC
        ii. Individual Secrecy : should have its own security association (eg. unique session key).
    b. Achieved by Security paras verification(end points checks the negotiated sec paras b/w each segment).
3. Integrity :
    a. Data Source Authentication (C should confirm that msg recv is from S itself)
    b. Modification Accountability (C should know which MBs have made modn)
    c. Path Integrity (to verify that msg has passed through auth MBs in the established order)
    d. Achieved by Valid Modn Checks.

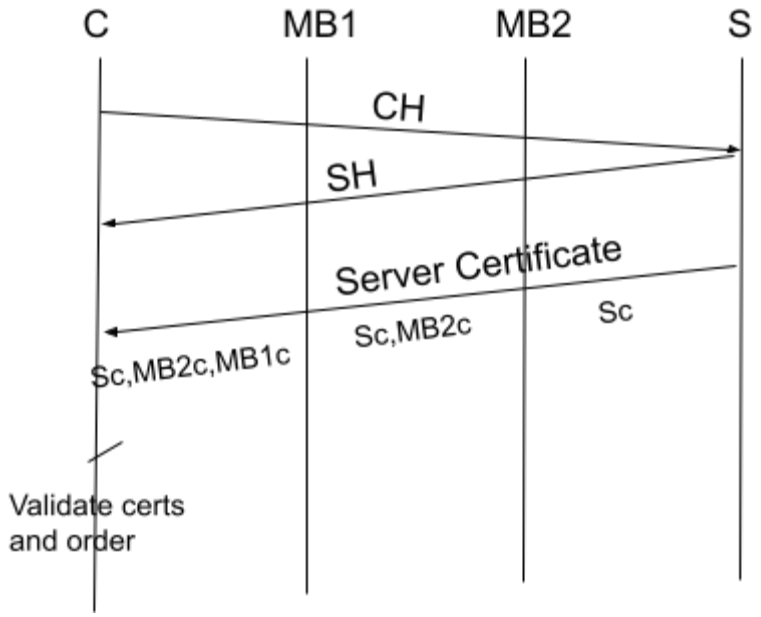
maTLS design approach :

- TDN : C establishes TLS with S. Then both endpoints pass seg keys to auth MBs via sep TLS connections. Entire session uses the same keys. No incremental deployment.
- BUP : C & MB initiates TLS segments sequentially up to S. Two entities of each segment negotiate their sec paras individually. Partial establishment of maTLS(good).

Audit Mechanisms for C :

[Note : Accountability keys will be used as HMAC keys. Ak keys b/w C & S, b/w MB and each endpoint.]

1. **Explicit Auth :**
    a. Auth the S & MBs.
    b. S sends its cert in “ServerCertificate” msg, each MB appends its own cert.
    c. C checks:
        i. All the certs.
        ii. Order of the certs.



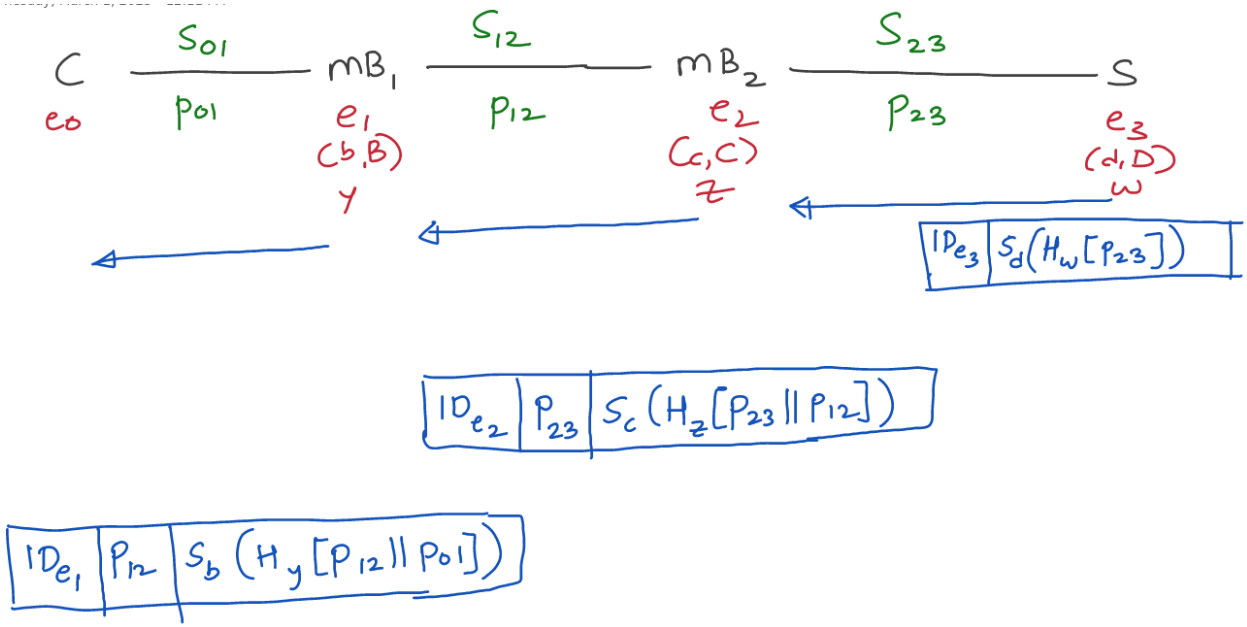
2. **Sec Para Verfin :**
    a. C can check the sec paras of each TLS segment, AKs and their order.
    b. **Sec paras** (pij)= [Chosen TLS ver, Negotiated CC, H(MS), Hased Transcript of TLS HS=verify_data in FIN]
        i. TLS ver, CC →shows deg of conf of this segment
        ii. H(MS) → uniqueness of seg keys
        iii. H(Trans of HS) →in this seg, the HS was done w/h any modn by attacker

    c. **Sec Block** of ei :

Identifier	Sec Para of segment in the direction of server	Signature with ei's private key on { Hash(done by ak of ei with client) on [Sec paras in both directions]}

MBs : $ID_i||p_{i,i+1}||Sign(sk_i, Hmac(ak_{i,0}, p_{i-1,i}||p_{i,i+1}))$

Sv : $ID_n||Sign(sk_n, Hmac(ak_{n,0}, p_{n-1,n}))$



i. ID = H(pk)
ii. C receives all these 3 Sec para blocks and now confirms all the sec paras b/w each segment by verifying signature of signed HMACs. Verification fail → Sec paras are modified, or missing or incorrect order of MBs.
iii) C thus verifies aks, sec paras, order, then decides to accept/ reject(eg. Weak algo like rc4 is used somewhere in between) the session.

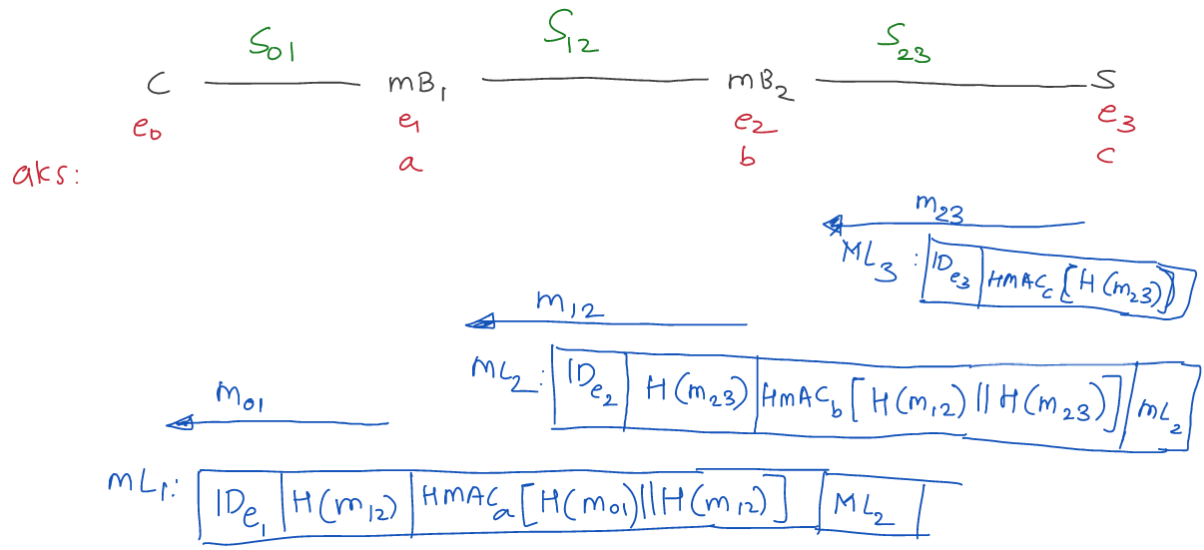
3. **Valid Modn Checks :**
   a. Allows a C to check which entity has modified the msg.
   b. When ei forwards a msg to ei+1, it also sends a ML(modification log) which is a cryptographic proof. It is to compare the incoming and outgoing msgs from ei. A ML has :
      i. HMAC(using ak) on recv and sent msg
      ii. Digest of received msg + id.
   c. A ML keeps track of modifications of a packet.
   d. **ML** of ei:

ID	Hash of previous message	HMAC with ak on (Hash of received msg + Hash of sent msg)	Previous ML

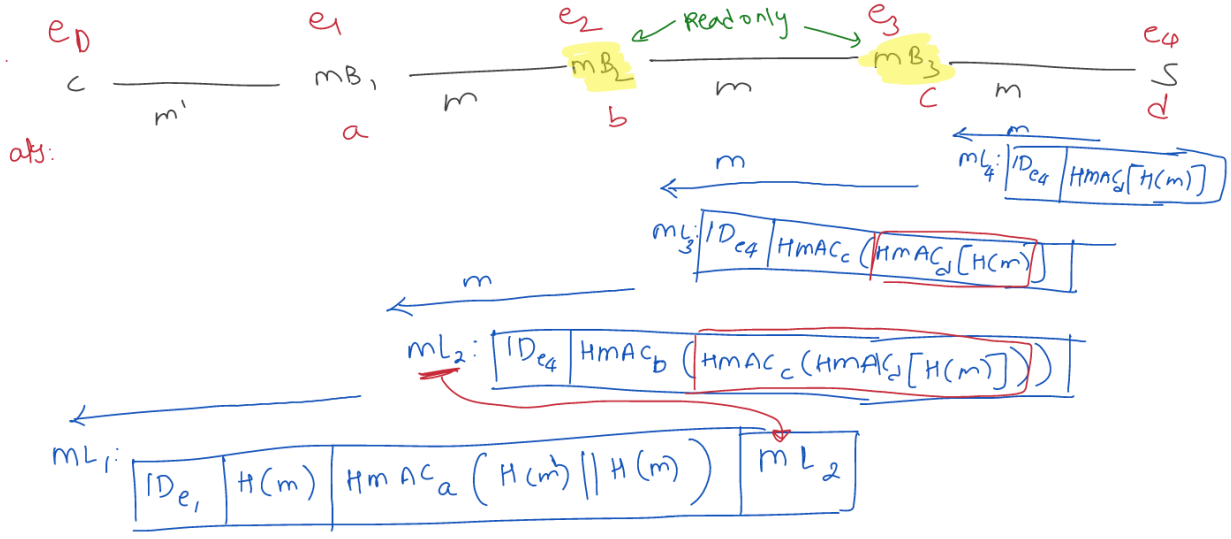
MBs : $$ID_i||H(m_{i+1})||Hmac(ak_{i,0}, H(m_i)||H(m_{i+1}))||ML_{i+1}$$

Sv : $$ID_n||Hmac(ak_{n,0}, H(m_n))$$

S sends a msg to C

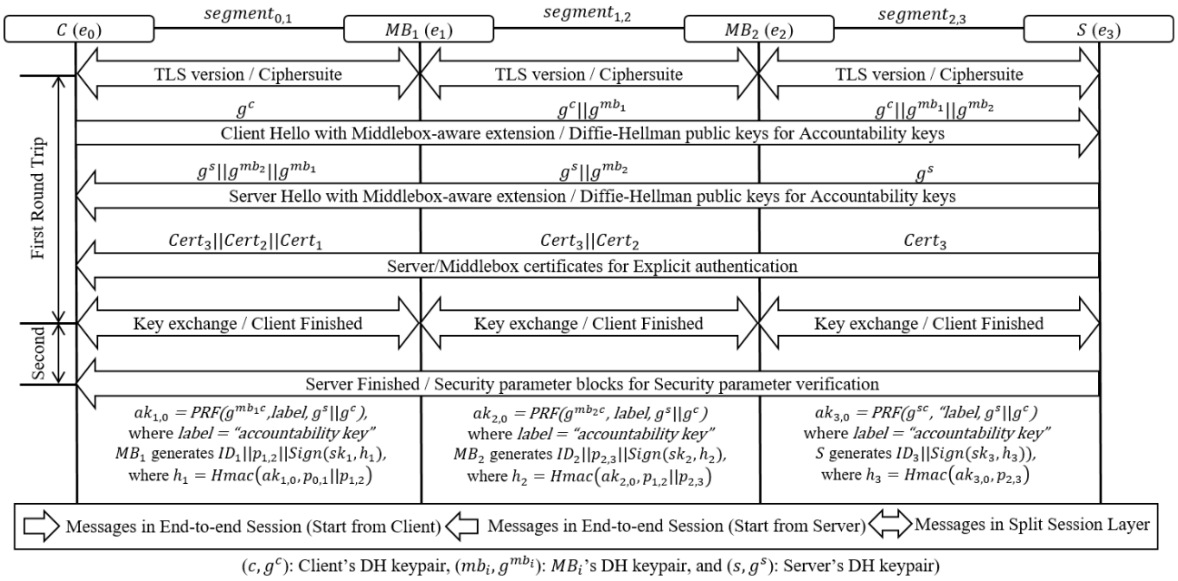


Optimisation : for a read-only MBi, omit the H(mi+1) and IDi. generate the HMACi from HMACi+1. So if C detects an omitted ID while parsing the received ML, it can assume the msg is not modified.



Implies that the msg wasnt modified b/w e2 to e4.
Once recv gets all the MLs, it extracts the digests of all modified msgs, tracks the identifiers of the MBs that performed the write operation, and **finally verifies each ML using HMAC.**

maTLS HS protocol:



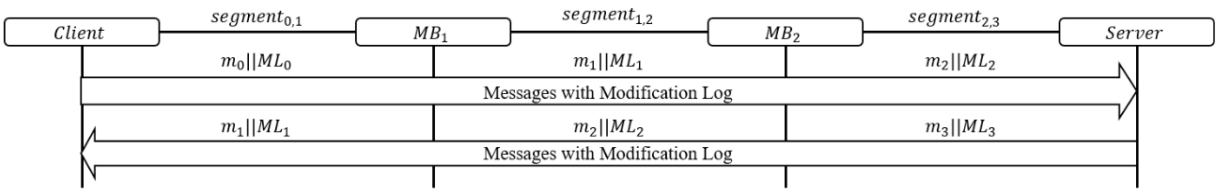
(c, g^c): Client's DH keypair, (mb_i, g^{mb_i}): MB_i's DH keypair, and (s, g^s): Server's DH keypair)

(a) The maTLS-DHE handshake protocol on TLS 1.2 (server-only authentication)

1. Client generates (c,g^c), puts it into extension.
2. When a MB receives a CH with maTLS ext, it generates its own DHE key pair, extracts the list of DH public keys from maTLS ext, appends its own DH pk and sends a new CH.
3. S generates (s,g^s) and sends SH.
4. When a MB receives a SH, it determines the TLS ver + CC to be used in maTLS segment.
5. Each entity negotiates the TLS ver+CC with its neighbors for each maTLS segment. (doubt : same CC for all segments?)
6. Both C and S get the DH pks from all other entities, each MB has 2 DH pub keys.
7. All the entities generate accountability keys from their own DH priv keys by using PRF function[9] with server's DH pub key & Client's DH public key as seeds. For a label, one of the input paras of the PRF function, we use the string "accountability key".
8. ServerCert is sent. Client does **explicit auth** here.
9. Client maps each ak to its corresponding ID (ID = digest of entity's pk).
10. Each maTLS segment then exchanges the key materials via ServerKeyExchange and ClientKeyExchange msgs. FIN msgs are exchanged.
11. After this normal TLS FIN, a new **ExtendedFinished** msg is sent from sv to client. It includes sec para blocks. Client does the **sec para verification** here. Also confirms the proofs of possession of private key by verifying the signatures by processing the ExtendedFinished msg.

maTLS Record protocol :

Does **modification checks** during data exchange.



(b) The maTLS record protocol with a modification log.

Each MB in the session leaves their own MAC in the ML whenever data is passed, the endpoints can confirm whether the order or MBs is preserved by verifying the MACs with the ak keys in sequence.

Security Verification :

Tamarin to analyze sec goals of maTLS.