Middle-Box Aware TLS Protocol (maTLS)

Team Members:

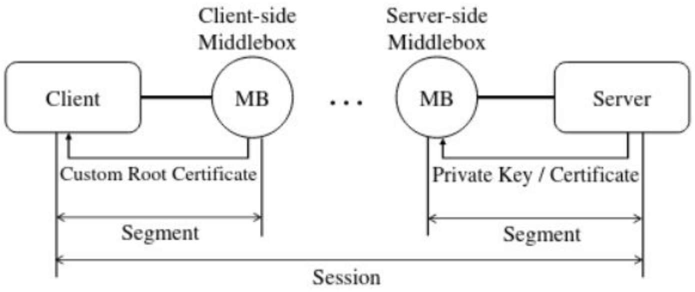
Tejas Deshmukh [cs22mtech12005]
Gantasala Naga Aneesh Ajaroy [cs19btech11010@iith.ac.in]

Guidance:
Prof. Bheemarjuna Reddy Tamma
Prof, CSE-IITH
(Course Instructor)

T.A. Mentor:
Harinder Kaur
Satvik Padhiyar

Problem Statement

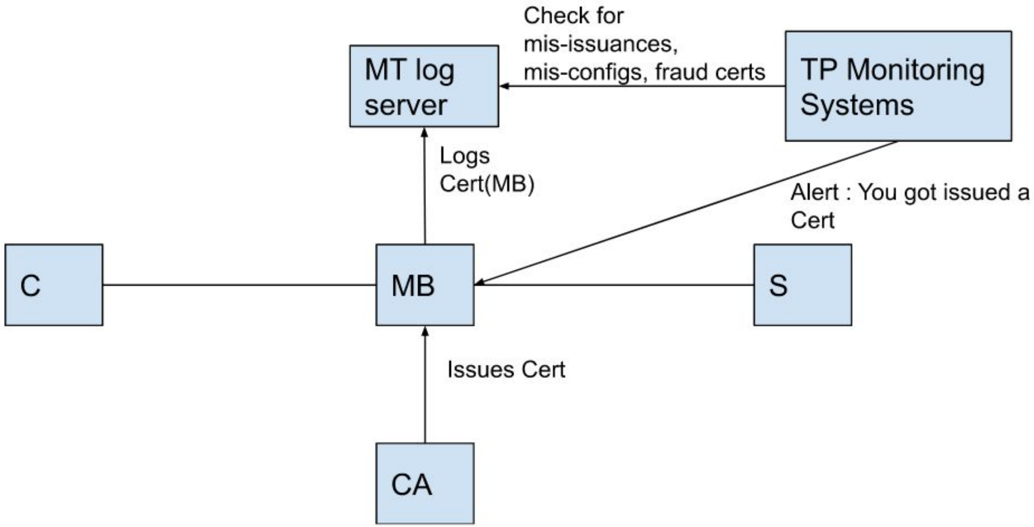
- Middleboxes are widely used for various purposes like Security enhancements, content filtering, etc.
- But, they are rendered useless with the introduction of TLS protocol.
- Proposed solutions to include MBs were Encryption-based, TEE. But these have limited functionalities. Thus, we adopt TLS Extension Based.



- Existing solutions like SplitTLS includes the middleboxes in TLS sessions, but this compromises Confidentiality, Integrity and Authenticity. To achieve these goals, we implement a Middlebox Aware protocol (maTLS).

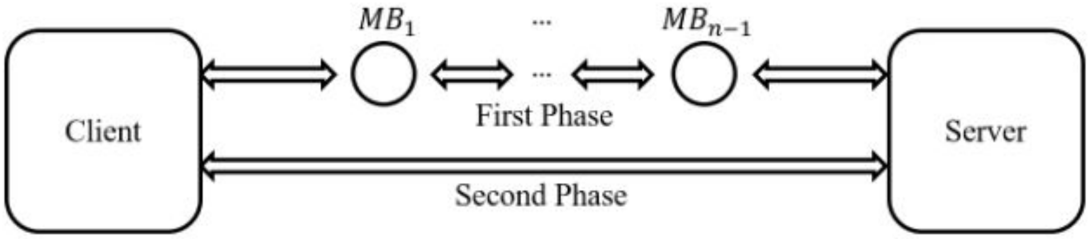
MOTIVATION

- Inclusion of MBs in TLS without compromising previous goals by making them “Auditable”.
- Goals :
  - Authentication:
    - Server and MBs authentication
  - Confidentiality
    - Segment Secrecy
    - Individual Secrecy
  - Integrity
    - Data Source authentication
    - Modification Accountability
    - Path Integrity



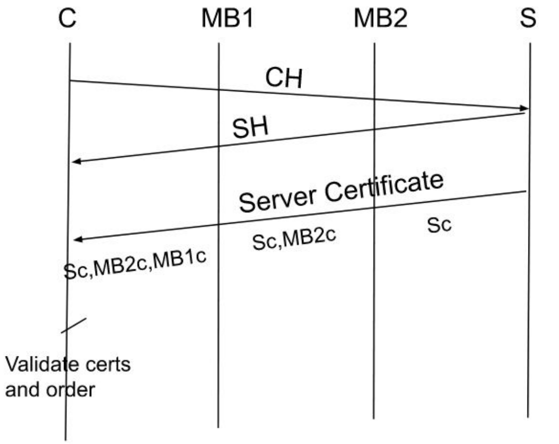
Approach

Bottom-up approach :



Audit Mechanisms :

Authentication : Explicit Validation[Certificates]



Server cert sending

```
# loading server cert
cert = OpenSSL.crypto.load_certificate(
OpenSSL.crypto.FILETYPE_PEM,
open('/root/prj/new_certs/server_cert.crt').read())

# converting cert to string to send to client for explicit authentication
output=OpenSSL.crypto.dump_certificate(OpenSSL.crypto.FILETYPE_PEM, cert)
string=str(output)
encoded_str=string.encode('ascii')
clientsock.sendall(encoded_str)
```

MB appending and Sending

```
# reciving cert from server
recv_data = fake_clientsock.recv(BUFFSIZE)

# loading MB cert
cert = OpenSSL.crypto.load_certificate(
OpenSSL.crypto.FILETYPE_PEM,
open('/root/prj/new_certs/mb_cert.crt').read())

# converting cert to string to send to client for explicit authentication
output=OpenSSL.crypto.dump_certificate(OpenSSL.crypto.FILETYPE_PEM, cert)
string=str(output)

# concatinating server cert and MB cert string swith "$" delimiter
final = recv_data.decode('ascii') + "$" + string
final = final.encode('ascii')
print("recieved ", recv_data.decode(), " from ", serverhostname, "\n")
print("sending ", final.decode(), " to ", clienthostname, "\n")
client_sock.sendall(final)
```

Explicit Authentication

```
# Explicit Authentication of clients

os.system("openssl verify -verbose -CAfile new_certs/root.crt server_test.crt")
os.system("openssl verify -verbose -CAfile new_certs/root.crt MB_test.crt")
```

```
server_test.crt: OK
MB_test.crt: OK
MB and server validation successful
Explicit Authentication done!!
```

Approach

Confidentiality : Security Parameter Verification

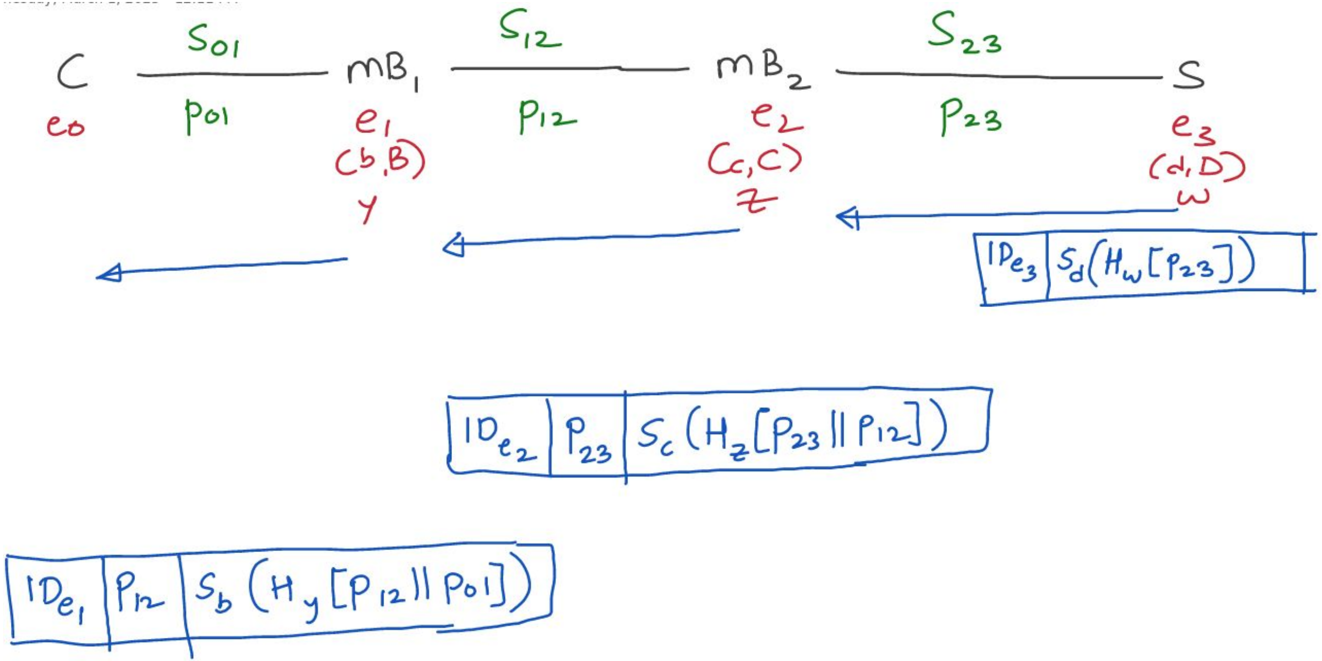
- Sec paras (pij)= [Chosen TLS ver, Negotiated CC, H(MS), Hased Transcript of TLS HS=verify_data in FIN]
- Security Block :

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

Approach


C
S01
p01
mB1
S12
p12
mB2
S23
p23
S
e0
e1
(b,B)
y
e2
(c,C)
z
e3
(d,D)
w
IDe3 | Sd(Hw[p23])
IDe2 | p23 | Sc(Hz[p23 || p12])
IDe1 | p12 | Sb(Hy[p12 || p01])

Creating Security Parameter Block of Server

```python
# getting master key and its hash
t = sslkeylog.get_master_key(clientsock)
hash_obj = hashlib.sha256()
hash_obj.update(t)
master_hash = hash_obj.hexdigest()
print("master hash ",master_hash, "\n")

# got security parameters
sec_param_server += master_hash
print("Security Param Server: ",sec_param_server, "\n")

# hashing of security parameters with accountability key assumed to be sent by client
sec_param_server_bytes = sec_param_server.encode('utf-8')
hash_obj = hmac.new(key=b'server key', digestmod=hashlib.sha256)
hash_obj.update(sec_param_server_bytes)
sec_param_hash = hash_obj.hexdigest()
```

```python
# Sign the security parameters hash using the private key
sec_param_hash_bytes = sec_param_hash.encode('utf-8')
sec_param_hash_sign_bytes = sign(private_key, sec_param_hash_bytes, 'sha256')
sec_param_hash_sign = sec_param_hash_sign_bytes.hex()

print("\nsec parameter signed hash : ",sec_param_hash_sign, "\n")

creating sec_param_block contining server ID and signed hash of secutiry params

sec_param_block = id_server + "\n" + sec_param_hash_sign
print("\n\nsec param block of server : ", sec_param_block,"\n")
```

```python
# getting sec params
sec_param_client_mb = ""
t = client_sock.cipher()
print("TLS protocol:",t, "\n")
sec_param_client_mb += t[0] + "$" + t[1] + "$"

# getting master key
t = sslkeylog.get_master_key(client_sock)
hash_obj = hashlib.sha256()
hash_obj.update(t)
master_hash = hash_obj.hexdigest()
print("master hash ",master_hash, "\n")

# got security parameters of client MB (p12 as in NS rough notes)
sec_param_client_mb += master_hash
print("sec param of client MB conn:", sec_param_client_mb, "\n")


# concatenate sec params of (MB-server) & (client-MB)
sec_params_final = sec_param_mb_server + sec_param_client_mb
```

```python
print("MB<----->Server\n")

# getting master key
t = sslkeylog.get_master_key(fake_clientsock)
hash_obj = hashlib.sha256()
hash_obj.update(t)
master_hash = hash_obj.hexdigest()
print("master hash ",master_hash, "\n")

# got securing parameters of MB server (p23 as in NS rough notes)
sec_param_mb_server += master_hash
print("sec param of MB server conn:", sec_param_mb_server, "\n")
```

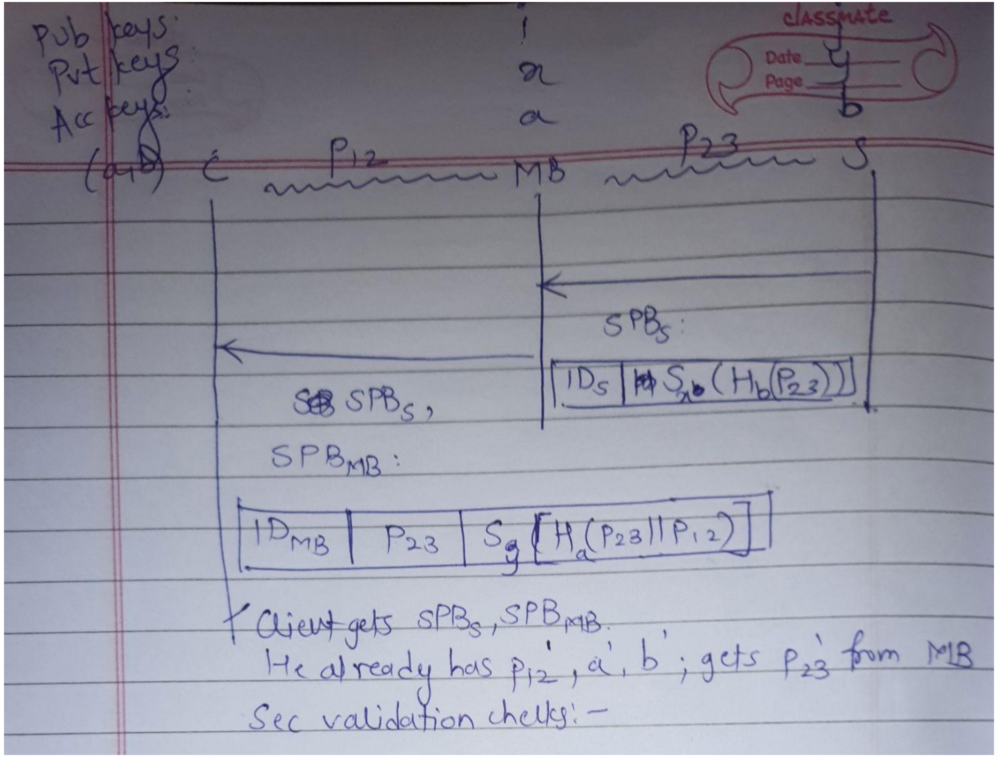
Security parameter block of Middle Box

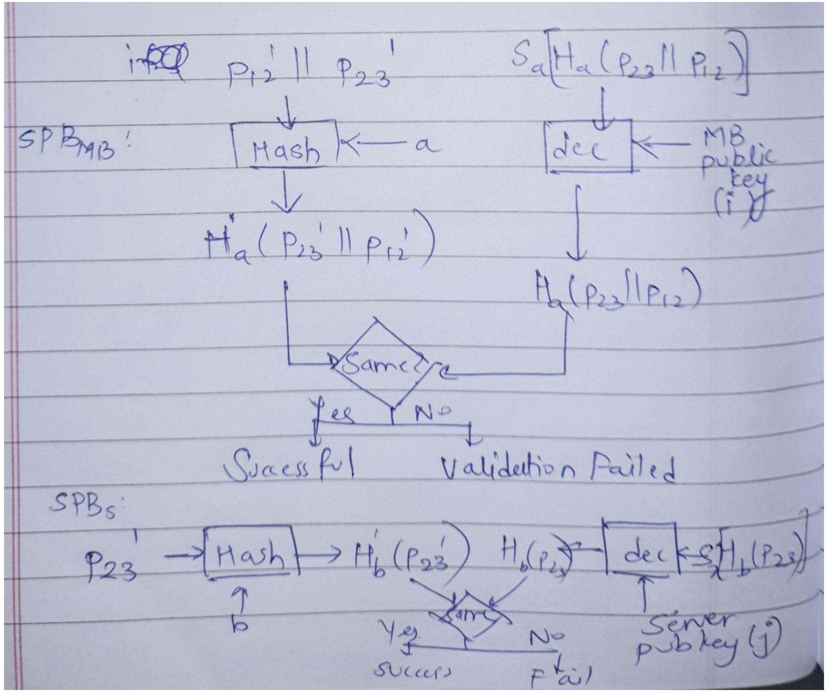
```python
# Load private key from a PEM file
with open('/root/prj/new_certs/mb_key.pem', 'rb') as f:
    private_key = load_privatekey(FILETYPE_PEM, f.read())

# Sign the final security parameters hash using the private key
sec_param_final_hash_bytes = sec_param_final_hash.encode('utf-8')
sec_param__final_hash_sign_bytes = sign(private_key, sec_param_final_hash_bytes, 'sha256')
sec_param_final_hash_sign = sec_param__final_hash_sign_bytes.hex()

print("\nconcat sec parameter signed hash : ",sec_param_final_hash_sign)

# creating sec_param_block contining MB ID, sec param of MB-server, signed hash of secutiry params

sec_param_block_mb = id_MB + "\n" + sec_param_mb_server + "\n" + sec_param_final_hash_sign
print("\n\nsec param block of MB : ", sec_param_block_mb,"\n")
```

Validating secure parameter blocks at client

```
#verify the signature now !
signed_hash_server_param_by = signed_hash_server_param.encode('utf-8')
sec_param_hash_by = sec_param_hash.encode('utf-8')
try:
    public_key.verify(
        signed_hash_server_param_by,
        sec_param_hash_by,
        padding.PKCS1v15(),
        hashes.SHA256()
    )
    print("Signature is valid")
except InvalidSignature:
    print("Signature is invalid")
```

How the security parameter blocks verified ?





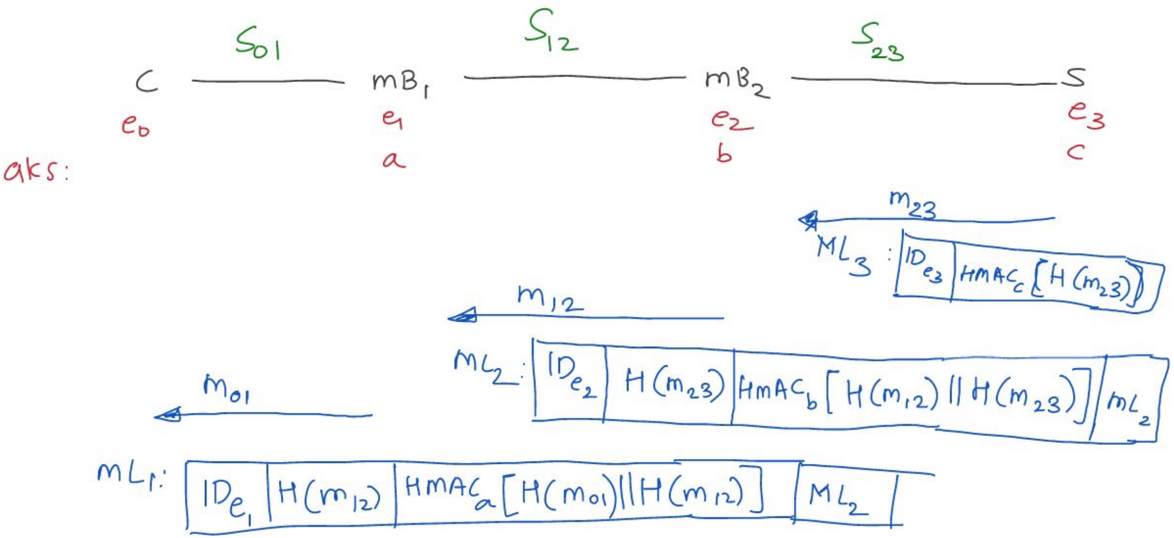
Approach

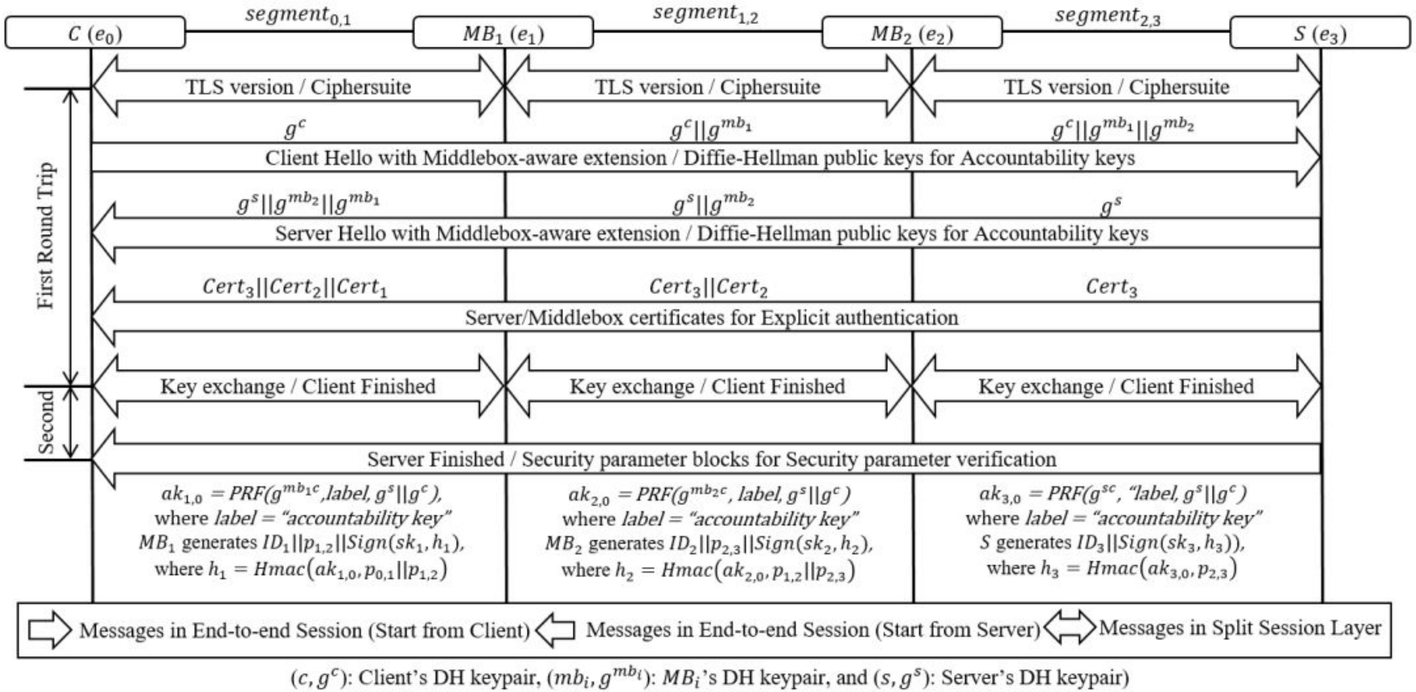
Integrity : Valid modification checks

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

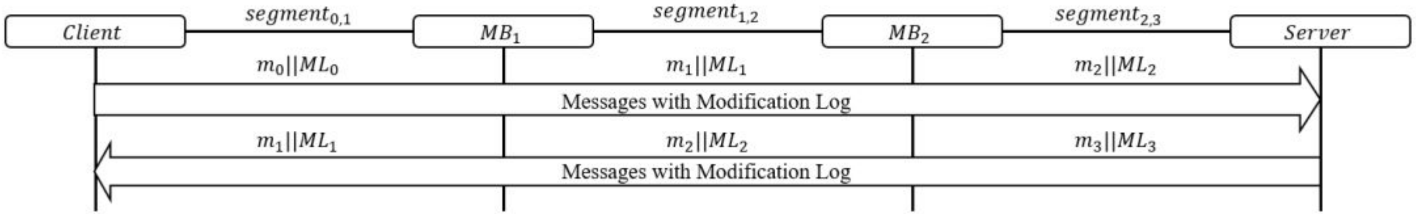
MBs : $ID_i||H(m_{i+1})||Hmac(ak_{i,0}, H(m_i)||H(m_{i+1}))||ML_{i+1}$

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





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



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

DELIVERABLES

- Python scripts for Client, Middlebox and Server
- Certificates, Report, Summary, PPT.

REFERENCES

1. maTLS: How to Make TLS middlebox-aware? - NDSS Symposium 2022, https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_01B-6_Lee_paper.pdf
2. D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. L´opez, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste, “Multi-context tls (mctls): Enabling secure in-network functionality in tls,” in ACM SIGCOMM Computer Communication Review, vol. 45, no. 4. ACM, 2015, pp. 199–212.