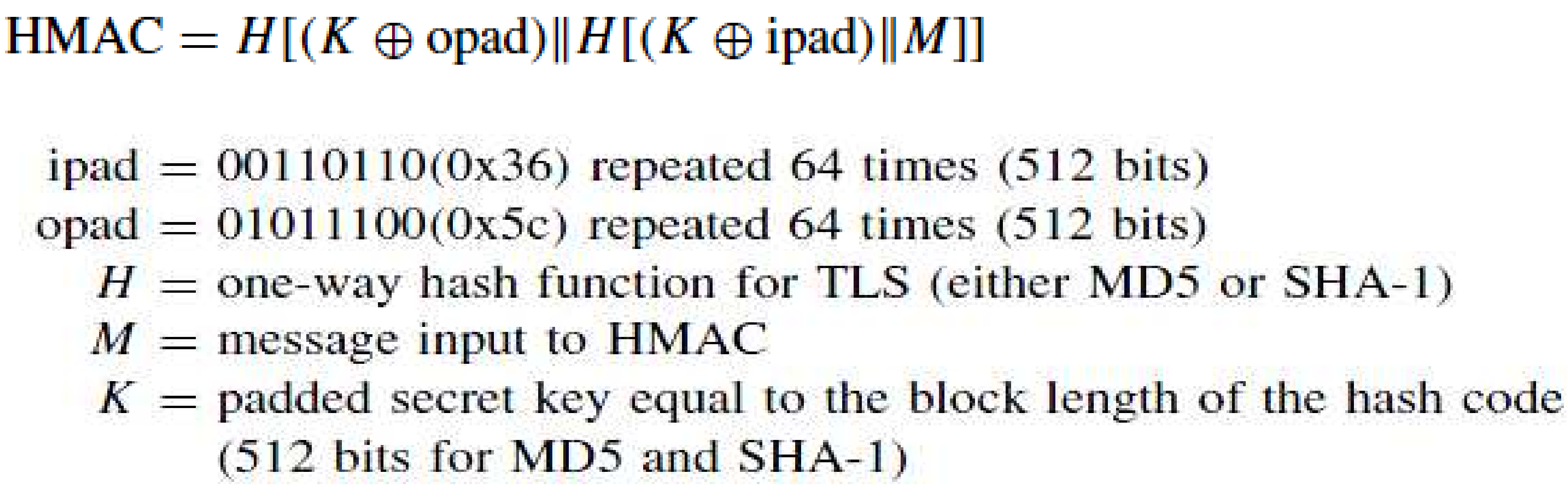
TLS Protocol

TLS Protocol

- The TLSv1 -based on the SSLv3
- No dramatic difference between them
- Algorithm, Data structures, Rules are very close
- Comparative studies RFC 2246

TLS Protocol

- **HMAC Algorithm**



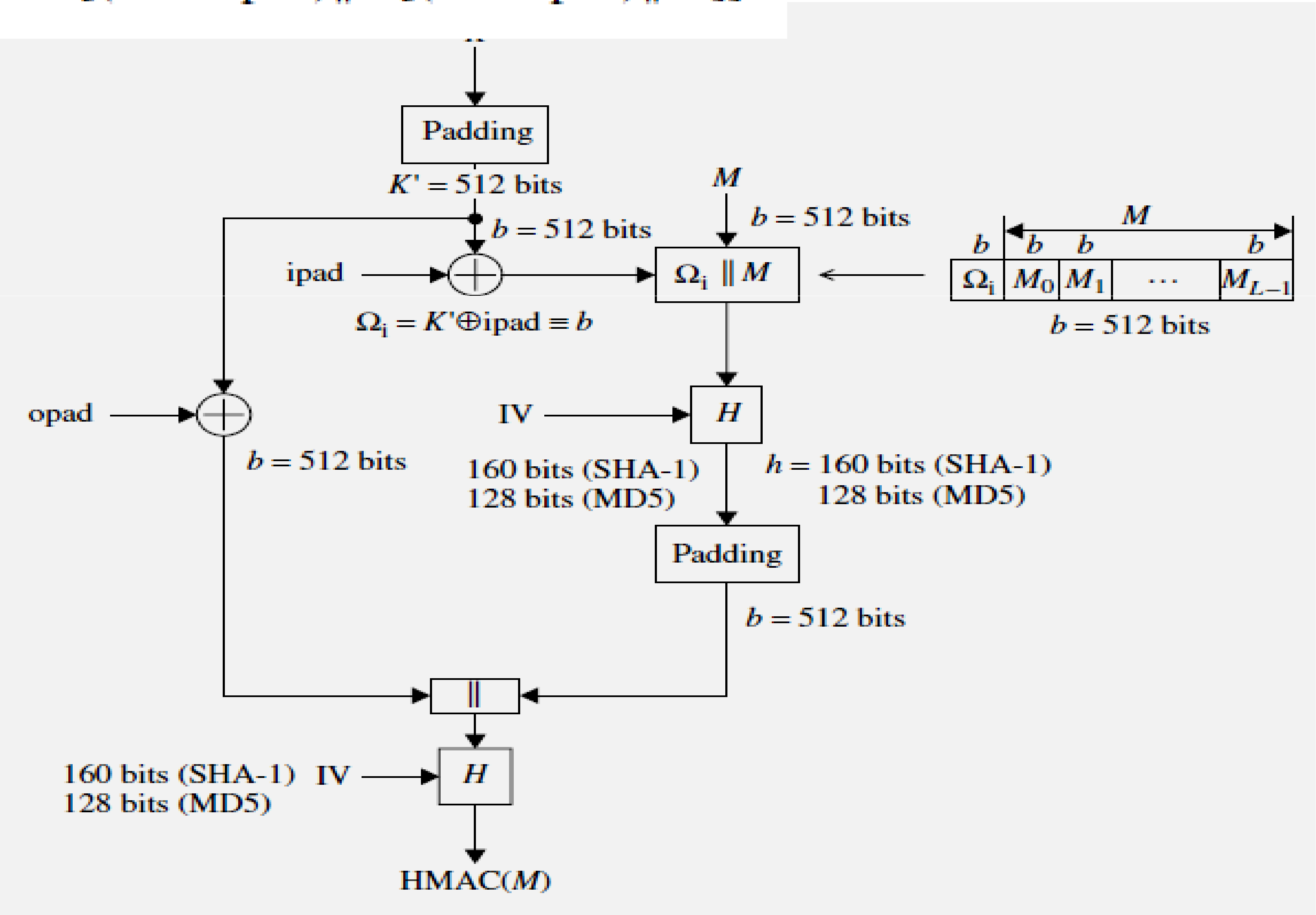
$$\text{HMAC} = H[(K \oplus \text{opad}) \| H[(K \oplus \text{ipad}) \| M]]$$

ipad = 00110110(0x36) repeated 64 times (512 bits)
opad = 01011100(0x5c) repeated 64 times (512 bits)
H = one-way hash function for TLS (either MD5 or SHA-1)
M = message input to HMAC
K = padded secret key equal to the block length of the hash code (512 bits for MD5 and SHA-1)

TLS Protocol

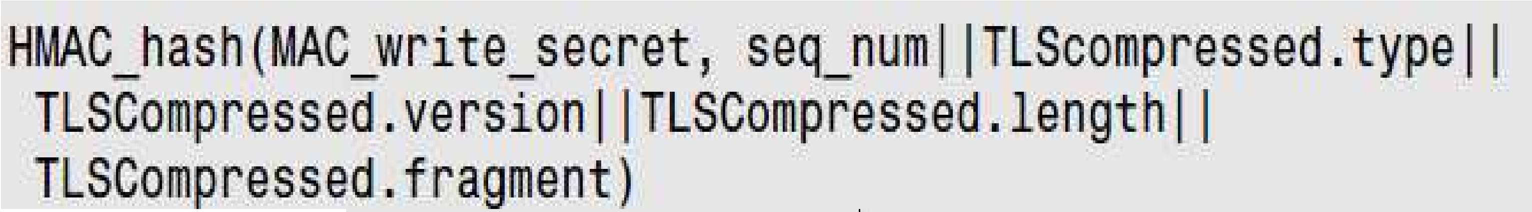
- **HMAC Algorithm**

$$\text{HMAC} = H[(K \oplus \text{opad}) \| H[(K \oplus \text{ipad}) \| M]]$$

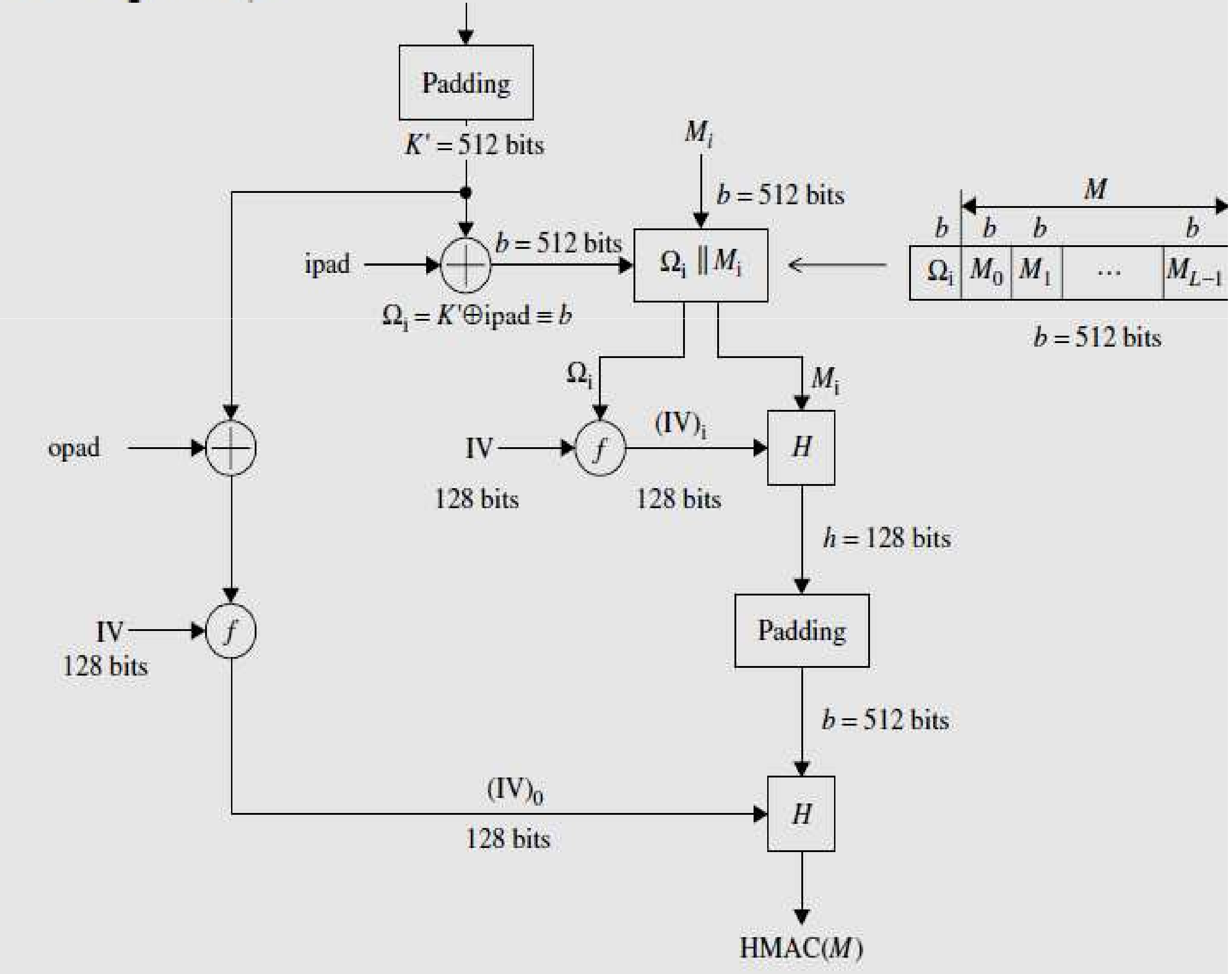


TLS Protocol

- **HMAC Algorithm**



```
HMAC_hash(MAC_write_secret, seq_num||TLScompressed.type||
 TLSCompressed.version||TLSCompressed.length||
 TLSCompressed.fragment)
```



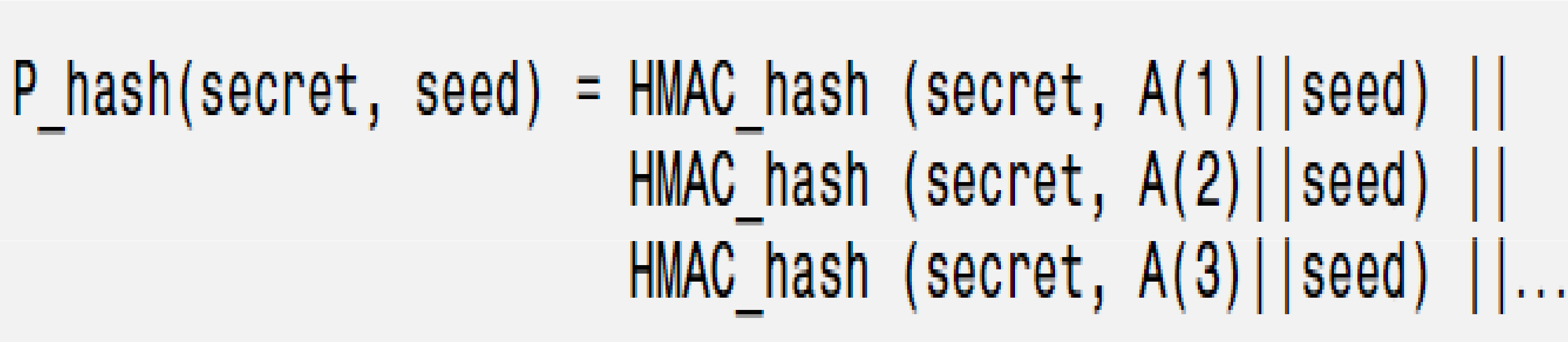
TLS Protocol

- **Pseudo-random Function (PRF)**
  - PRF - expand secrets into blocks of data for the purposes of key generation or validation
  - It takes relatively small values such as
    - a secret
    - a seed
    - An identifying label

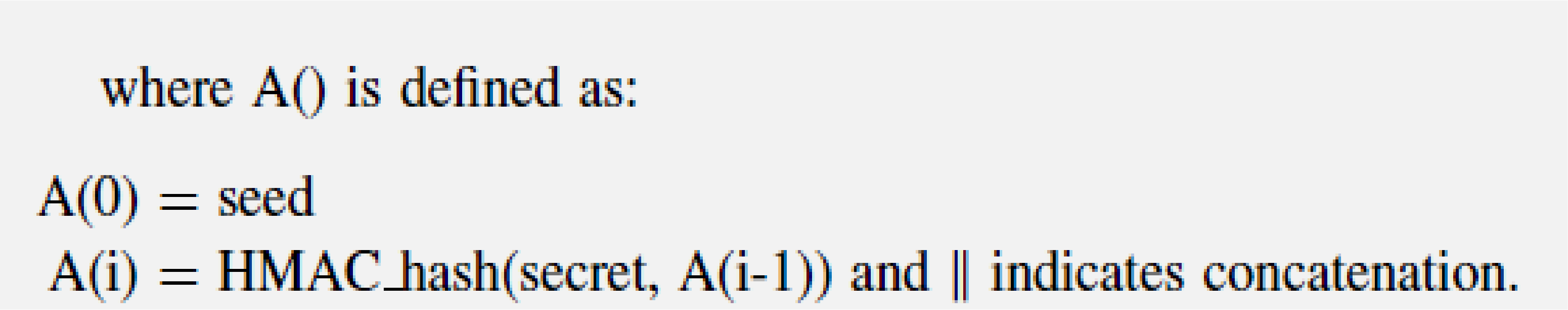
  as input and generates an output of arbitrary longer blocks of data

TLS Protocol

- **Pseudo-random Function (PRF)**
  - The data expansion function P



```
P_hash(secret, seed) = HMAC_hash (secret, A(1)||seed) ||
                       HMAC_hash (secret, A(2)||seed) ||
                       HMAC_hash (secret, A(3)||seed) ||...
```



where A() is defined as:

$A(0) = \text{seed}$

$A(i) = \text{HMAC\_hash(secret, A(i-1))}$ and $\|$ indicates concatenation.

TLS Protocol Pseudo-random Function (PRF)

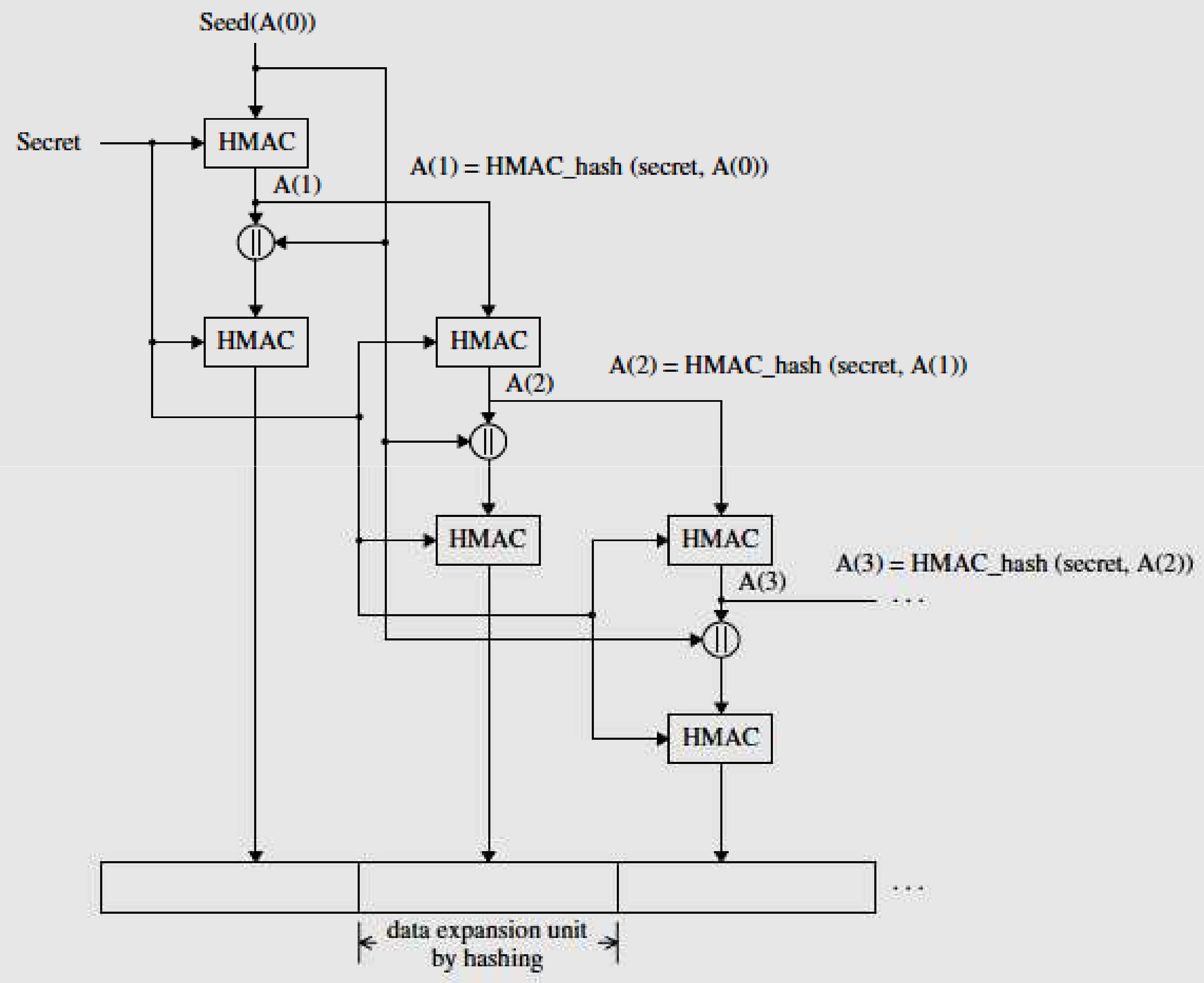

Seed(A(0))
Secret
HMAC
A(1)
A(1) = HMAC_hash (secret, A(0))
HMAC
HMAC
A(2)
A(2) = HMAC_hash (secret, A(1))
HMAC
HMAC
A(3)
A(3) = HMAC_hash (secret, A(2))
. . .
HMAC
. . .
data expansion unit
by hashing

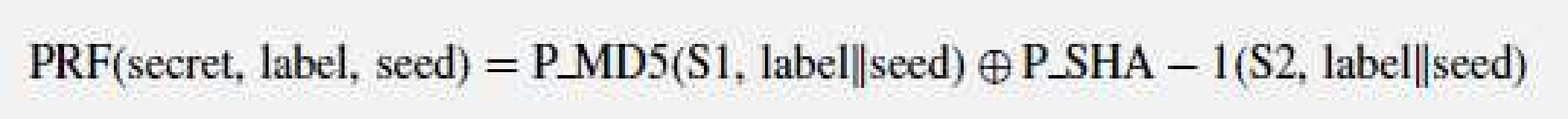
Figure 8.10 TLS data expansion mechanism using P_hash(secret,seed).

TLS Protocol

- **Pseudo-random Function (PRF)**
  - The data expansion function P
  - P hash is iterated as many times as necessary to produce the required quantity of data
    - SHA-1 = 20 bytes (160 bits)
      - 64 bytes (512 bits) - iterated four times up to A(4)
      - 20 × 4 = 80 bytes (640 bits) of output data
      - Last 16 bytes (128 bits) of the final iteration A(4) must be discarded
      - leaving *(80 − 16) = 64 bytes of output data*
      - 80-byte output while iterate through A(4)
    - MD5 = 16 bytes (128 bits)
      - 64 bytes (512 bits) - iterated five times up to A(5)
      - 16 × 5 = 80 bytes (640 bits) of output data
      - Last 16 bytes (128 bits) of the final iteration A(5) must be discarded
      - 80-byte output, P MD5 should exactly be iterated through A(5)

TLS Protocol

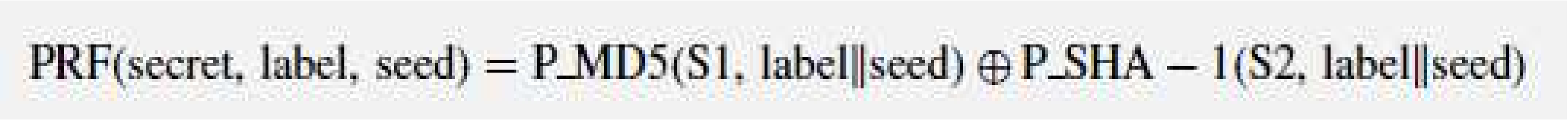
- **Pseudo-random Function (PRF)**
  - PRF is created by splitting the secret into two halves (S1 and S2)
    - S1 is taken from the first half of the secret
    - S2 from the second half
  - One half to generate data with P MD5
  - Other half to generate data with P SHA-1
  - These two results are then XORed to produce the output



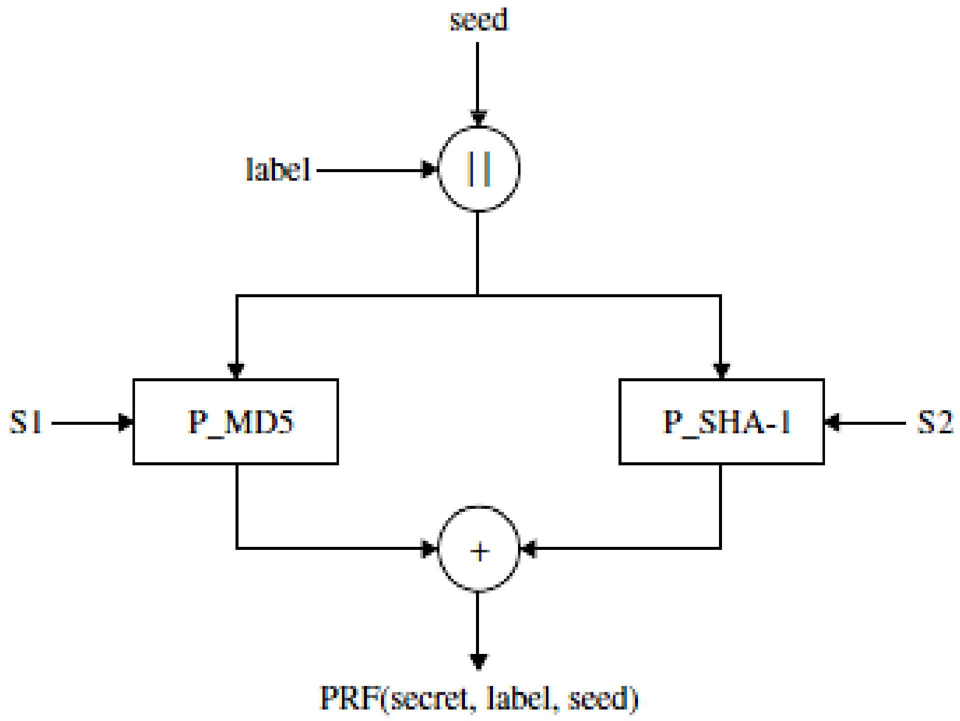
$$\mathrm{PRF}(\mathrm{secret}, \mathrm{label}, \mathrm{seed}) = \mathrm{P\_MD5}(\mathrm{S1}, \mathrm{label} \| \mathrm{seed}) \oplus \mathrm{P\_SHA} - 1(\mathrm{S2}, \mathrm{label} \| \mathrm{seed})$$

TLS Protocol

- **Pseudo-random Function (PRF)**



$$\mathrm{PRF}(\mathrm{secret},\ \mathrm{label},\ \mathrm{seed}) = \mathrm{P\_MD5}(\mathrm{S1},\ \mathrm{label}\|\mathrm{seed}) \oplus \mathrm{P\_SHA} - 1(\mathrm{S2},\ \mathrm{label}\|\mathrm{seed})$$



TLS Protocol

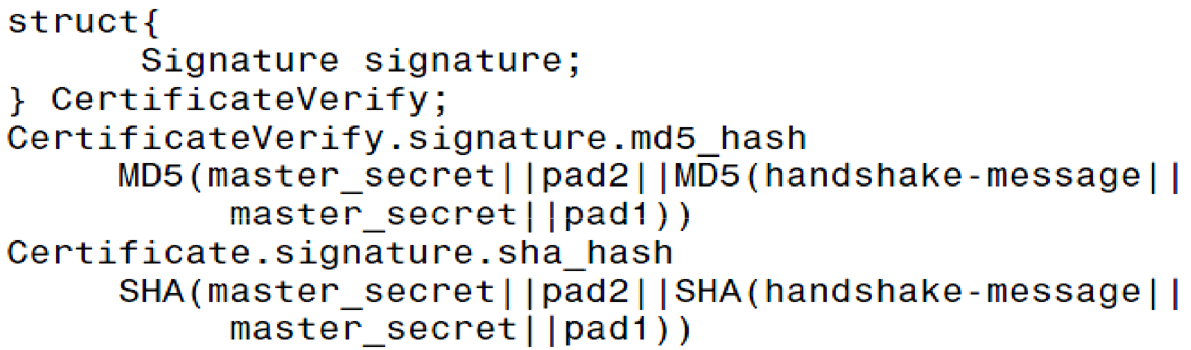
- **Error Alerts**
- Alert messages convey the severity of the message and a description of the alert
- Classified into the **closure alert and the error alert**
- **Closure alert**
  - Either party may initiate a close by sending a close notify alert
  - This message notifies the recipient that the sender will not send any more messages on this connection
  - In a truncation attack, an attacker inserts into a message a TCP code indicating the message has finished, thus preventing the recipient picking up the rest of the message.
  - To prevent this, a closing handshake alert is used
  - Recipient knows the message has not ended until **closure alert is received**
- **Error alert**
  - When an error is detected, the detecting party sends a message to the other party
  - Upon transmission or receipt of a fatal alert message, both parties immediately close the connection

TLS Protocol

- **Error alerts**
  - TLS supports all of the error alerts defined in SSLv3 with additional alert
    - Decryption failed
    - Record overflow
    - Unknown CA
    - Access denied
    - Decode error
    - Decrypt error
    - Decrypt error:
    - Export restriction
    - Protocol version
    - Insufficient security
    - Internal error:
    - User cancelled
    - No renegotiation
- **Alert level**
  - Not explicitly specified, the sending party may determine at its discretion whether this is a fatal error or not
  - Warning is received, the receiving party may decide at its discretion whether to treat this as a fatal error or not
  - Fatal is received , all messages must be treated as fatal messages and close connection

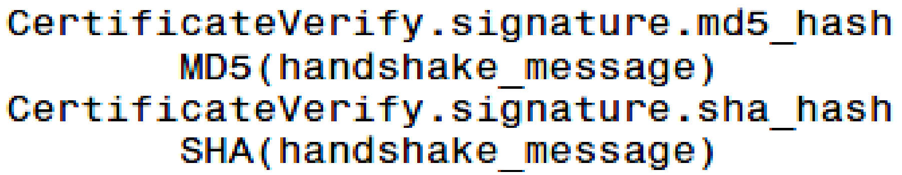
TLS Protocol

- **Certificate Verify Message**
  - SSLv3 included with the master secret, the handshake message and pads



```
struct{
      Signature signature;
} CertificateVerify;
CertificateVerify.signature.md5_hash
     MD5(master_secret||pad2||MD5(handshake-message||
          master_secret||pad1))
Certificate.signature.sha_hash
     SHA(master_secret||pad2||SHA(handshake-message||
          master_secret||pad1))
```

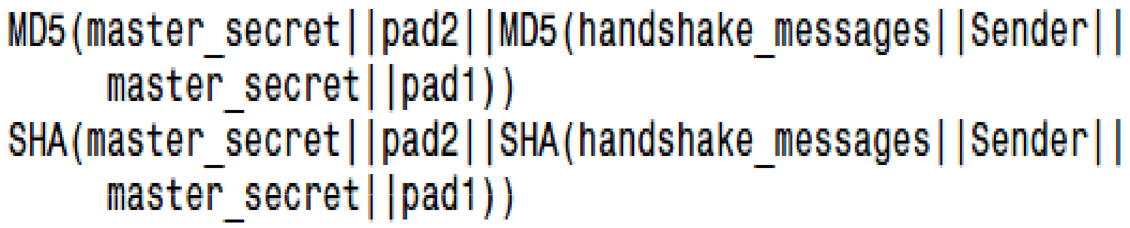
  - TLS certificate verify message, the MD5 and SHA-1 hashes are calculated only over handshake messages as shown below



```
CertificateVerify.signature.md5_hash
        MD5(handshake_message)
CertificateVerify.signature.sha_hash
        SHA(handshake_message)
```

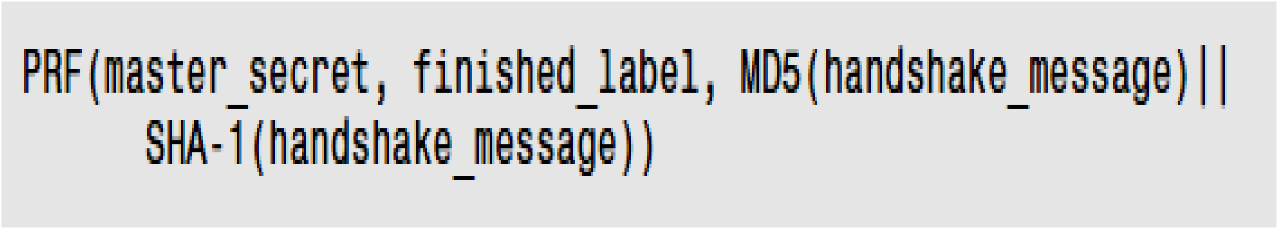
TLS Protocol

- **Finished Message**
  - **SSLv3**



```
MD5(master_secret||pad2||MD5(handshake_messages||Sender||
     master_secret||pad1))
SHA(master_secret||pad2||SHA(handshake_messages||Sender||
     master_secret||pad1))
```

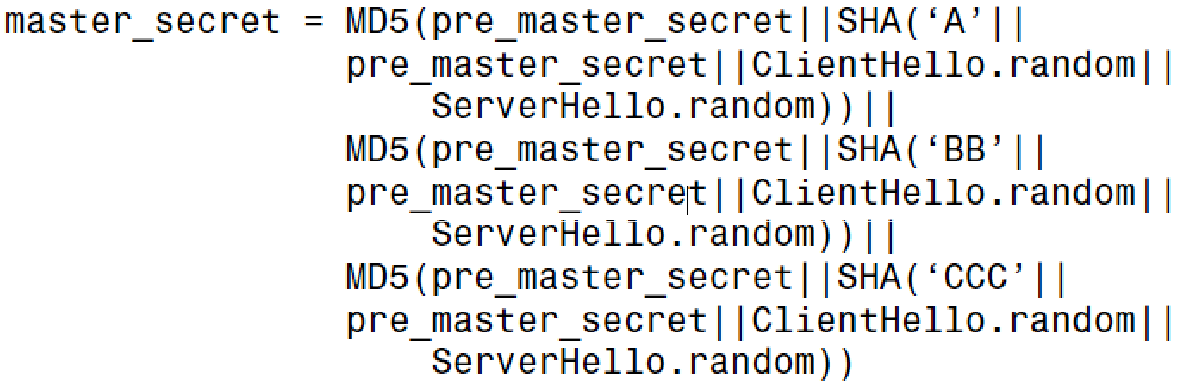
  - **TLS**



```
PRF(master_secret, finished_label, MD5(handshake_message)||
      SHA-1(handshake_message))
```

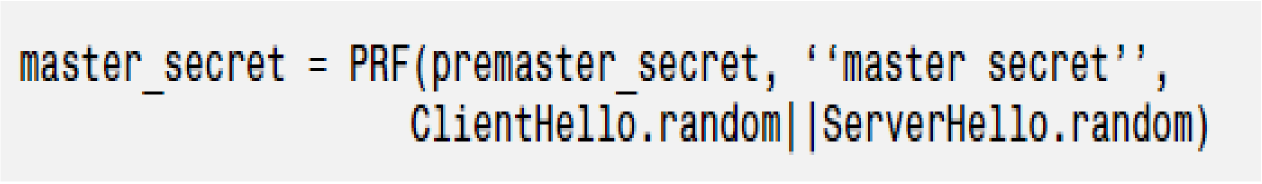
TLS Protocol

- Cryptographic Computations -Master secret
  - SSLv3



```
master_secret = MD5(pre_master_secret||SHA(‘A’||
                pre_master_secret||ClientHello.random||
                    ServerHello.random))||
                MD5(pre_master_secret||SHA(‘BB’||
                pre_master_secret||ClientHello.random||
                    ServerHello.random))||
                MD5(pre_master_secret||SHA(‘CCC’||
                pre_master_secret||ClientHello.random||
                    ServerHello.random))
```

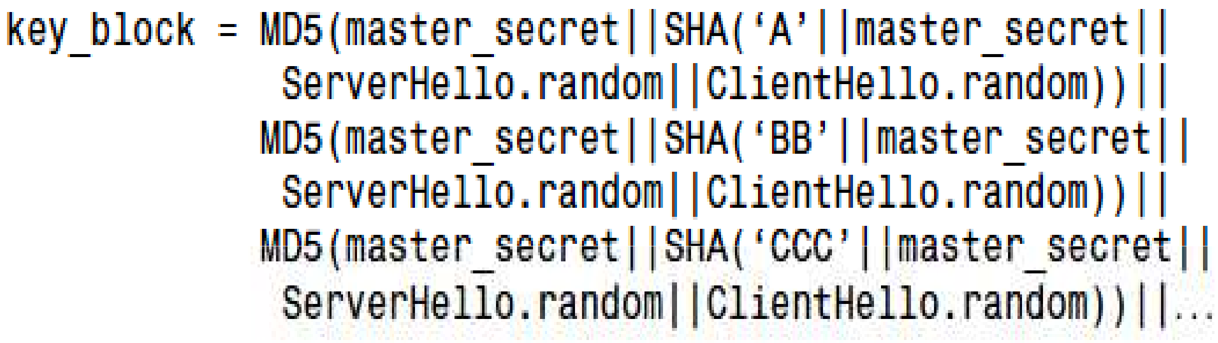
  - TLS



```
master_secret = PRF(premaster_secret, ‘‘master secret’’,
                      ClientHello.random||ServerHello.random)
```

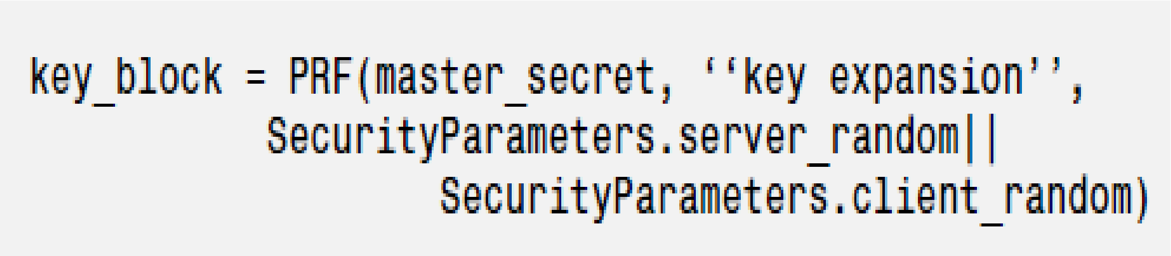
TLS Protocol

- **Cryptographic Computations -Key**
  - **SSLv3**



```
key_block = MD5(master_secret||SHA('A'||master_secret||
             ServerHello.random||ClientHello.random))||
            MD5(master_secret||SHA('BB'||master_secret||
             ServerHello.random||ClientHello.random))||
            MD5(master_secret||SHA('CCC'||master_secret||
             ServerHello.random||ClientHello.random))||...
```

  - **TLS**



```
key_block = PRF(master_secret, ''key expansion'',
            SecurityParameters.server_random||
                    SecurityParameters.client_random)
```