# WTLS 규격 (안)

2001. 5. 10.

정보보호기술연구본부

무선인터넷정보보호연구팀

이 석 준

ETRI

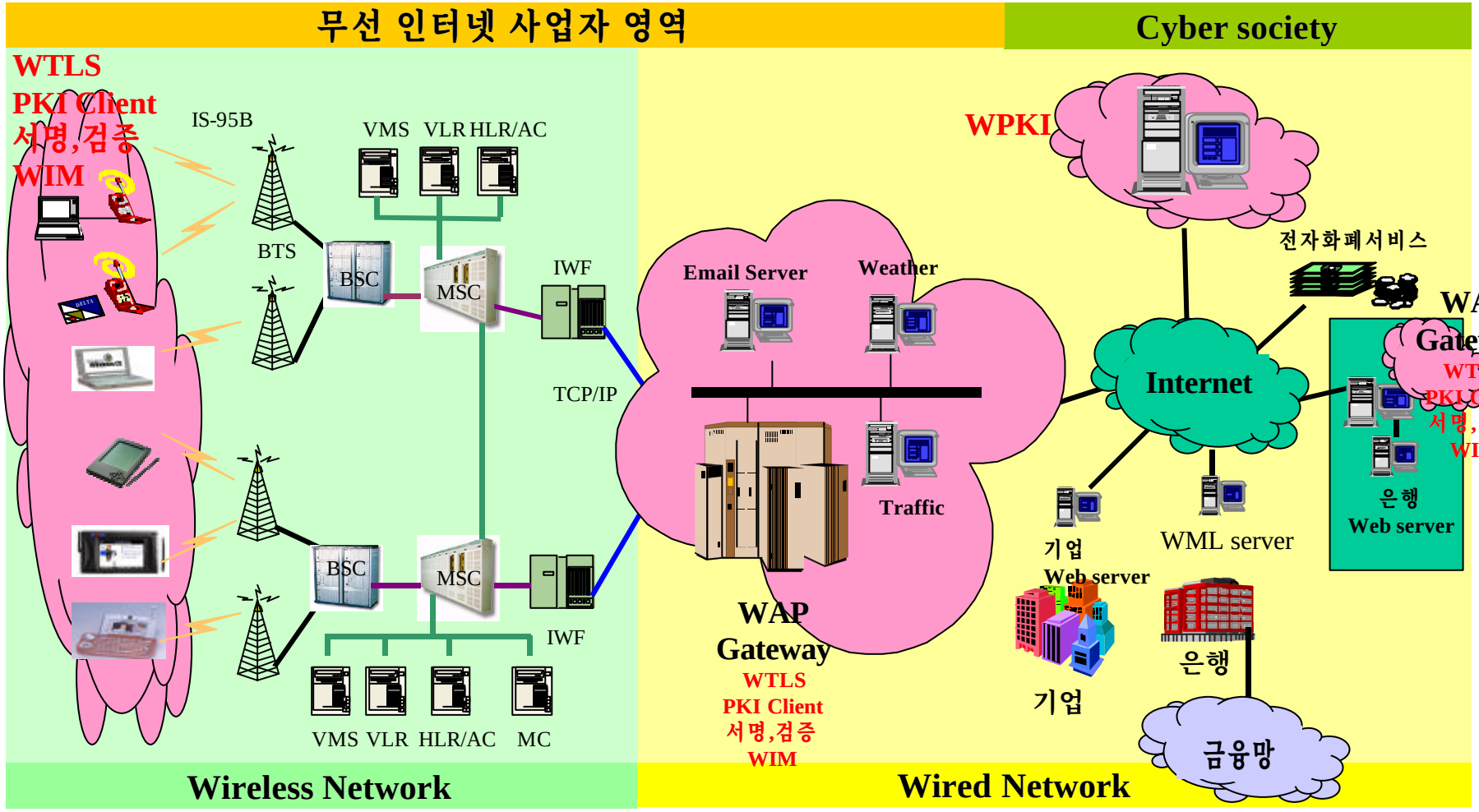
한국전자통신연구원

# 목 차

- WAP
- WTLS 규격 (안) – 개요
- WTLS 규격 (안) – 상세 설명
- TLS 와의 관계
- 결 론

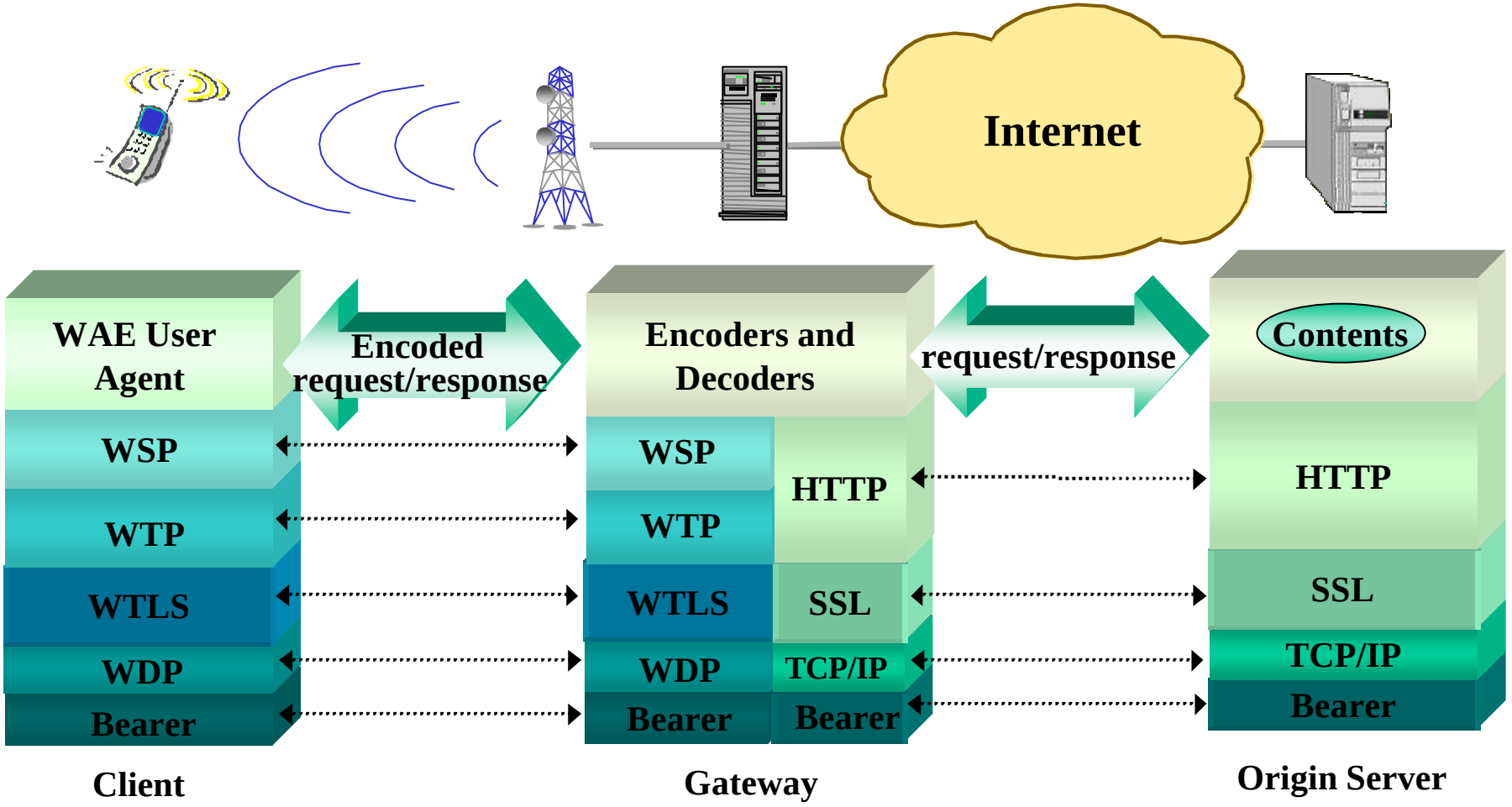
# 1. WAP 개요

# WAP 개요 – 1/3


무선 인터넷 사업자 영역
Cyber society
WTLS
PKI Client
서명,검증
WIM
IS-95B
VMS VLR HLR/AC
BTS
BSC
MSC
IWF
TCP/IP
BSC
MSC
IWF
VMS VLR HLR/AC MC
Email Server
Weather
Traffic
WAP
Gateway
WTLS
PKI Client
서명,검증
WIM
WPKI
전자화폐서비스
Internet
WAP
Gateway
WTLS
PKI Client
서명,검증
WIM
은행
Web server
기업
Web server
WML server
기업
은행
금융망
Wireless Network
Wired Network

# WAP 개요 – 2/3

## WAP Architecture



# WAP 개요 – 3/3

## WAP Protocol Stack 주요 기능

| | |
|---|---|
| WAE | • WML decode<br>• WML formatting & display<br>• WMLScript 실행 |
| WSP | • 세션 설립(connection orientied or connectionless)<br>• 세션 suspend/resume<br>• reliable & unreliable data push 처리 |
| WTP | • 트랜잭션 처리<br>• 재전송, 중복 전송 제거, acknowledgements<br>• Concatenation and Separation |
| WTLS | • 데이터 압축<br>• 암호화 및 해쉬(기밀성 및 무결성)<br>• 인증 및 키 일치 |
| WDP | • Port number Addressing<br>• Segmentation and Re-assembly (if provided)<br>• Error Detection (if Provided) |

# WTLS의 요구 조건

## 요구 조건

- 전송계층 보안프로토콜으로서의 기본 요구 조건
  - 기밀성, 무결성, 사용자 인증 제공
  - 다양한 응용 계층을 위한 상호호환성 제공
  - 새로운 알고리즘의 추가가 쉬워야 함
- 무선 구간에 보다 적합한 전송계층 보안프로토콜의 필요
  - 데이터그램 전송 프로토콜(UDP/WDP) 지원
    - 패킷 중복/손실에 대한 처리
  - 무선상의 느린 상호연결시간/낮은 전송율
    - 이를 고려한 전송 매커니즘의 필요성
    - 전송상의 오버헤드 최소화
  - 단말기의 제한된 계산 능력/메모리 용량
    - 이를 고려한 암호 알고리즘 선택 필요성
- 수출과 암호 사용의 제한
  - 가급적 각 지역의 법이 허용하는 최고수준의 보안 수준 달성

# 2. WTLS 규격 (안) - 개요

# WTLS 규격 (안) – 1/4

- 규격 (안)
  - 무선인터넷 환경에서 임의의 데이터에 대해 기밀성, 무결성, 사용자 인증을 제공하는 무선전송계층 보안프로토콜(WTLS, Wireless Transport Layer Security) 규정
- 참조 규격 및 관련성
  - 본 규격은 WAP WTLS 1.2를 근간으로 함
  - 특정한 언급이 없을 시 본 규격에서 명시적이지 않은 부분은 WAP WTLS 1.2를 참조한다.
- 적용 범위
  - 무선인터넷에서 통신하는 두 주체간의 인증, 통신 데이터의 기밀성, 무결성 등 정보보호 서비스의 제공
  - 이 규격에 따르는 기기, 응용 개발시 구현 적합성 여부 확인

# WTLS 규격 (안) – 2/4

- 관련 단체
  - WAP Forum
    - WAP Security Working Group (WSG)
    - WAP Architecture Committee (Arch)
  - IETF
    - Transport Layer Security Group (TLS)

# WTLS 규격 (안) – 3/4

- 암호 알고리즘
  - Key exchange algorithm
    - RSA
    - ECDH_ECDSA
    - RSA_anon, DH_anon, ECDH_anon
  - Bulk cipher algorithm
    - RC5_CBC
    - DES_CBC
    - 3DES_CBC
    - IDEA_CBC
  - Hash
    - SHA-1
    - MD5

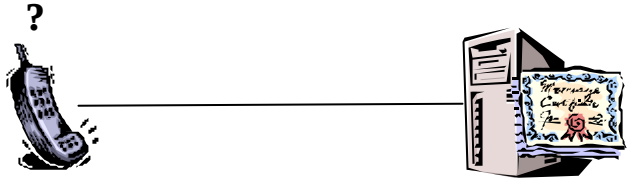
# WTLS 규격 (안) – 4/4

## Implementation Class

| Feature | Class 1 | Class 2 | Class 3 |
|---|---|---|---|
| 공개 키 교환 | M | M | M |
| 서버 인증서 | O | M | M |
| 클라이언트 인증서 | O | O | M |
| Shared-secret handshake | O | O | O |
| 압축 | - | O | O |
| 암호화 | M | M | M |
| MAC | M | M | M |
| 스마트 카드 인터페이스 | - | O | O |

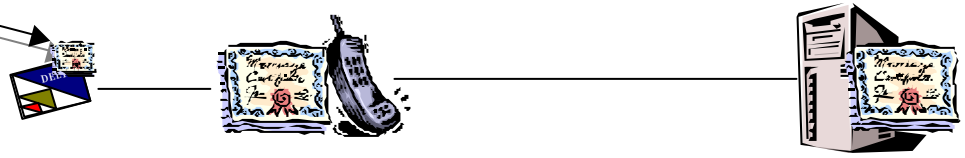
- **Class 1 – Anonymous**
  - **No Authentication**


- **Class 2**
  - **Server Authentication ONLY**



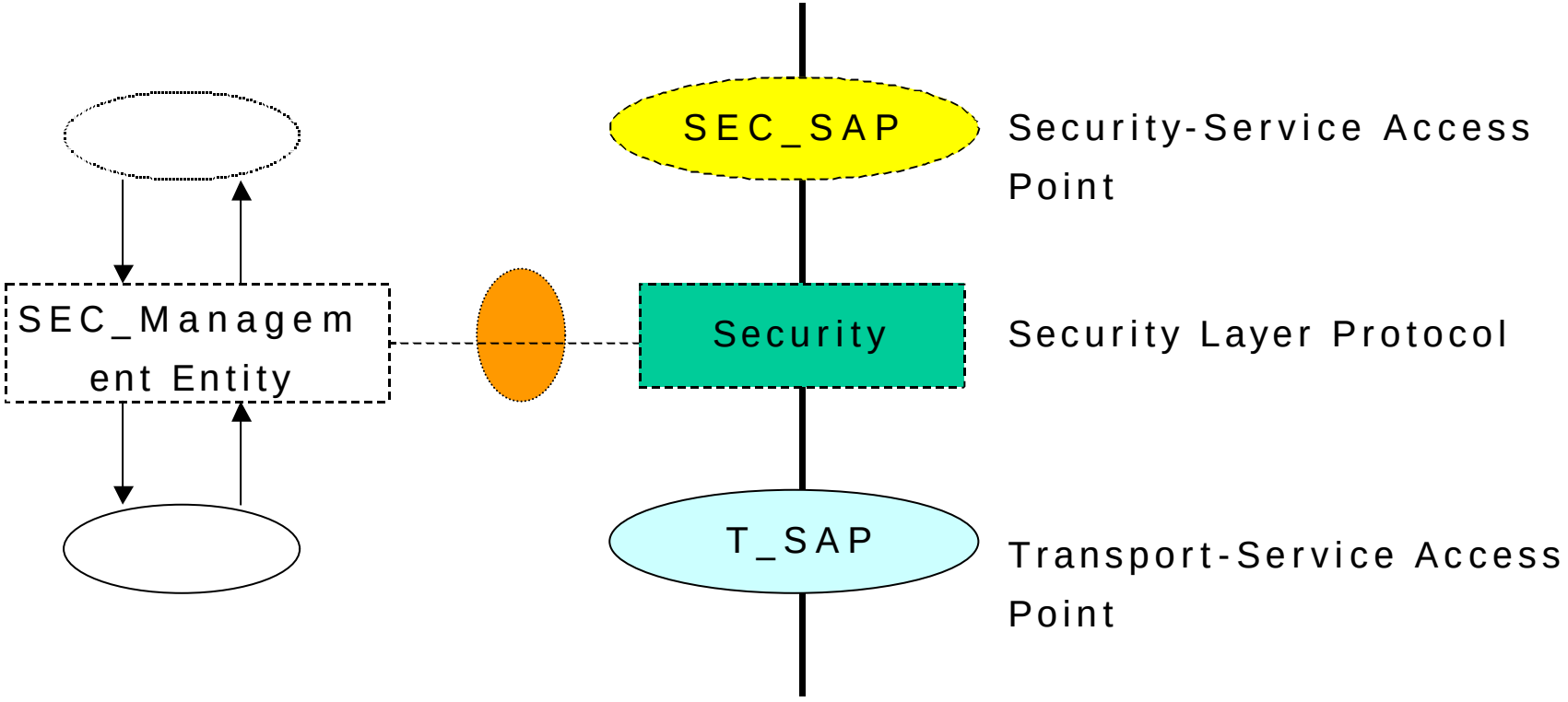

- **Class 3**
  - **Client & Server Authentication**

- **X.509 certificate**
- **WAP certificate**
- **ANSI X9.68**
- **URL certificate**



# 3. WTLS 규격 (안) – 상세 설명

# WTLS 구조


SEC_SAP
Security-Service Access Point
SEC_Management Entity
Security
Security Layer Protocol
T_SAP
Transport-Service Access Point

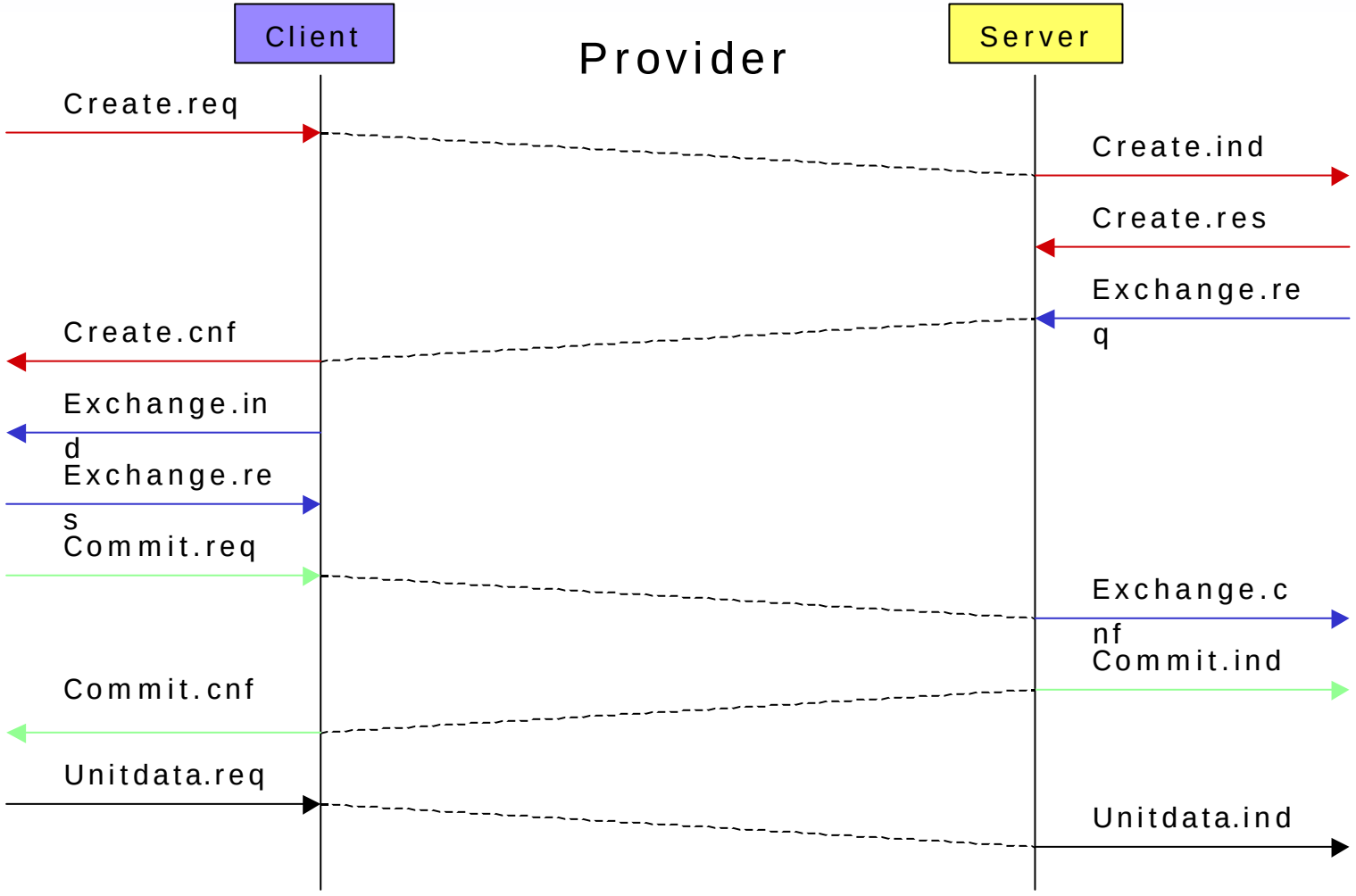
# 서비스 프리미티브 – 1/2

- 서비스 프리미티브
  - 각 계층간의 통신 수단
  - 함축적으로 설명
  - 제어와 정보에 대한 논리적 교환
  - 명령과 그에 상응하는 응답으로 구성
  - 문법 : X-서비스.형식 (파라미터)
    - X : 서비스 제공 계층
    - 형식 : request(req), indication(ind), response(res), confirm(cnf)
- WTLS 전송용 프리미티브
  - SEC_Unitdata

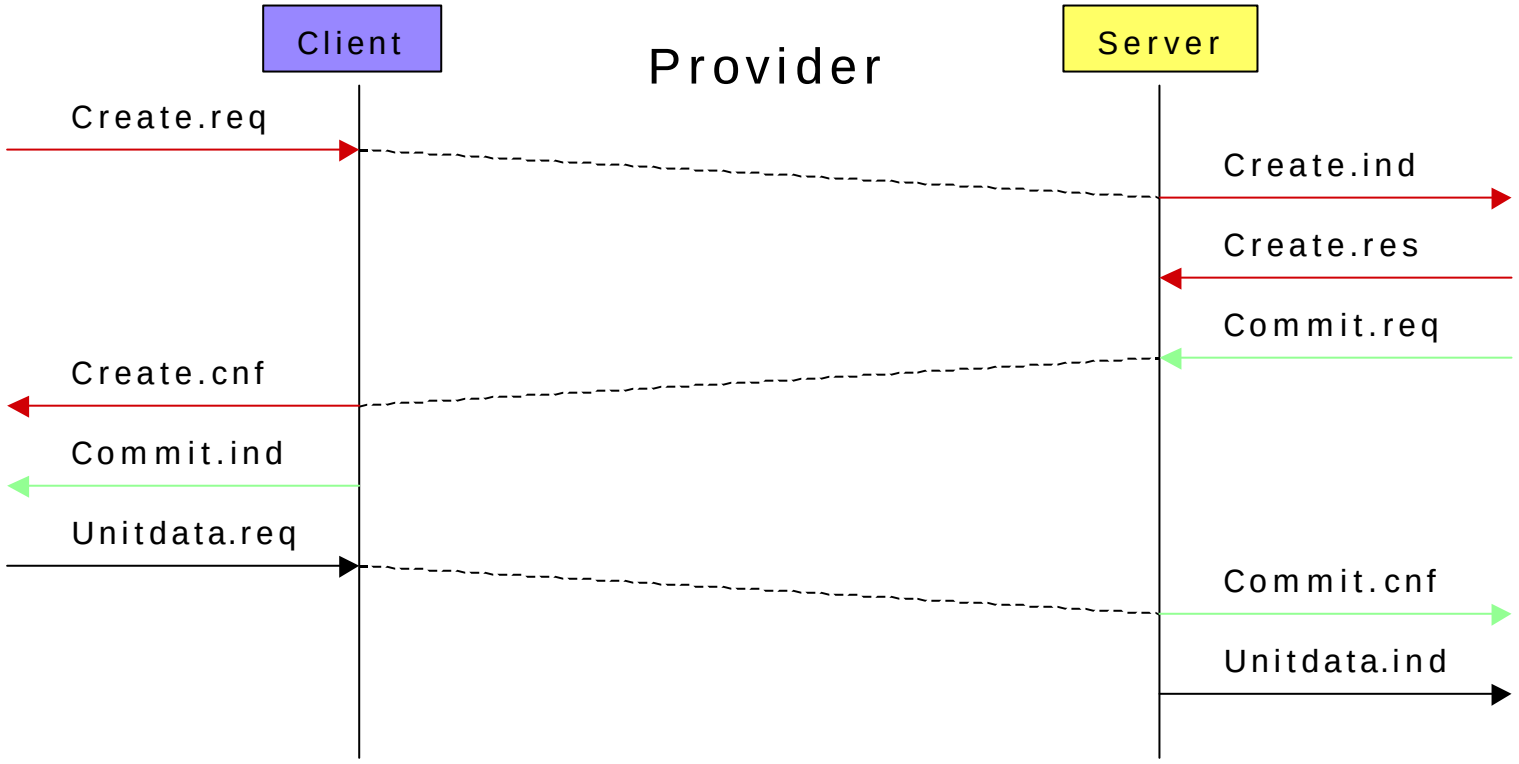
# 서비스 프리미티브 – 2/2

- WTLS 연결 관리용 프리미티브
  - SEC_Create
    - 보안 연결 설립을 시작할 때
  - SEC_Exchange
    - 서버가 공개키 인증을 원하거나 키 교환을 원할 때
  - SEC_Commit
    - 핸드쉐이크 과정이 완료되고, 클라이언트나 서버가 새로 협상된 연결 상태로 변경하자고 요구할 때
  - SEC_Terminate
    - 현재의 연결을 종료하고자 할 때
  - SEC_Exception
    - 상대방에게 경고 수준의 알림이 필요할 때
  - SEC_Create_Request
    - 클라이언트가 새로운 핸드쉐이크를 시작할 것을 서버가 요구할 때

# 프리미티브 순서 (완전 핸드쉐이크)



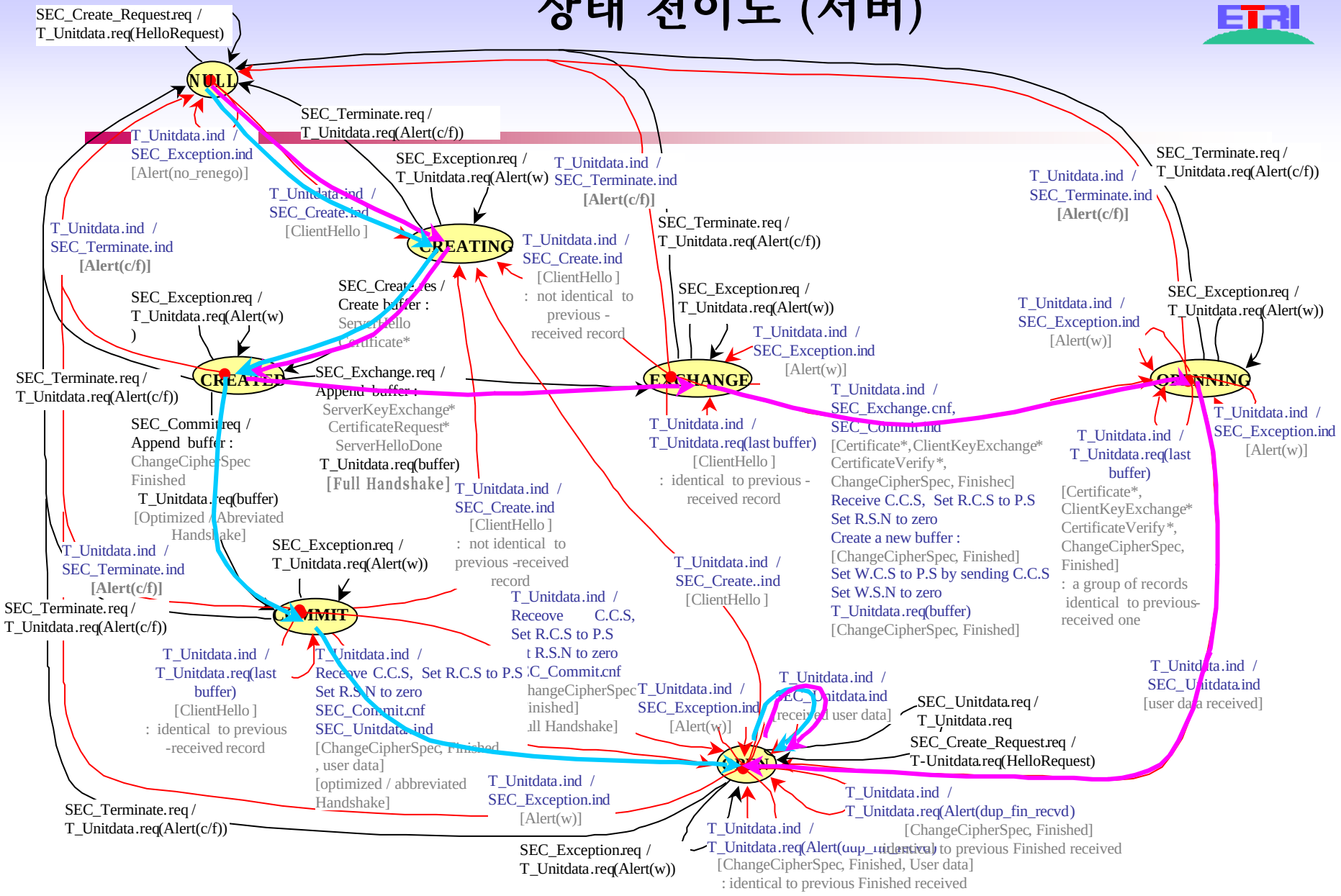
# 프리미티브 순서 (압축/최적 핸드쉐이크)



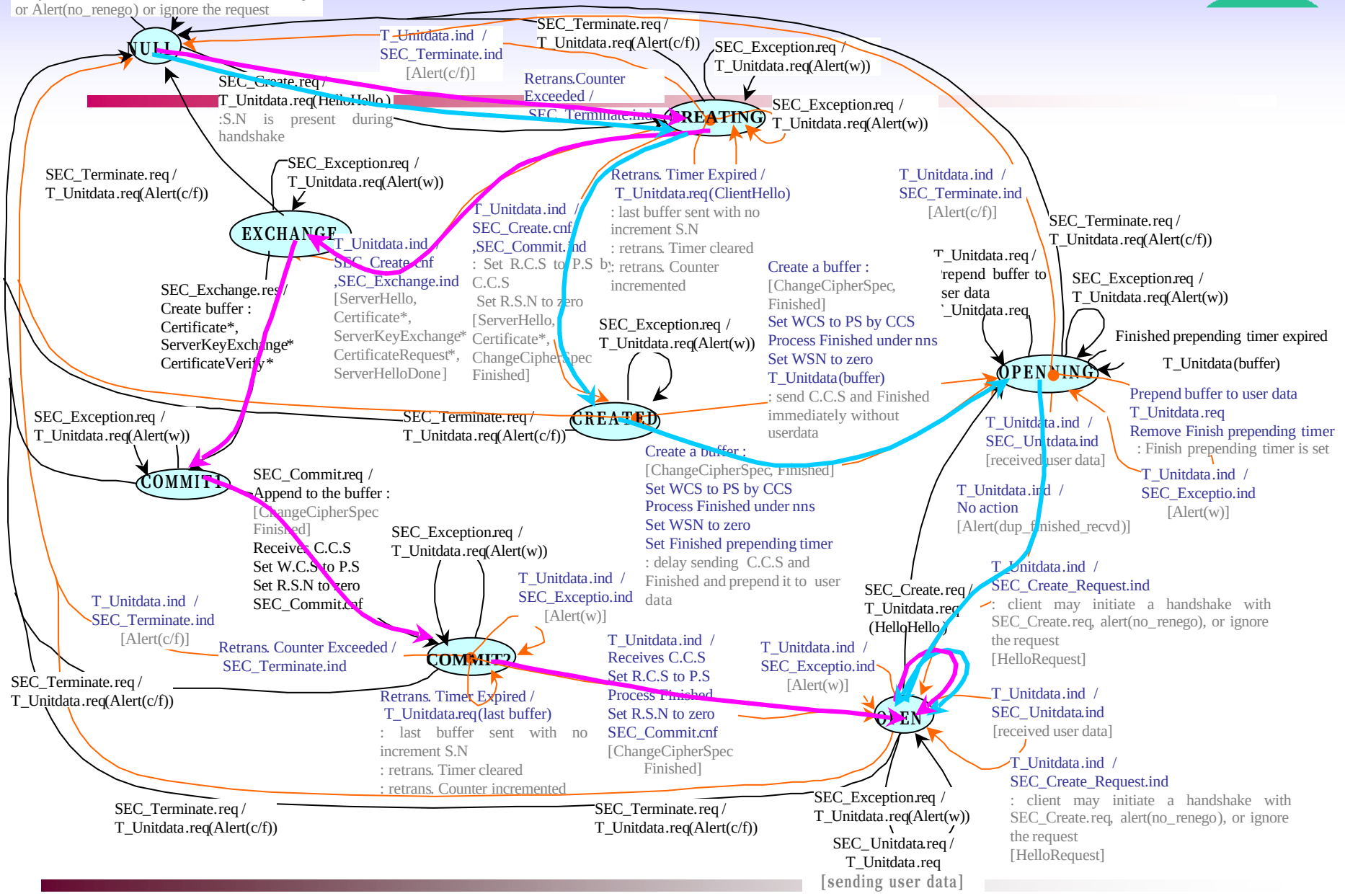
# 프로토콜 상태 테이블

- 상태 테이블
  - 규격상에서 프로토콜 상태 테이블 정의
  - 클라이언트와 서버는 서로 비대칭
  - 각 상태와 조건에 따라 해야 할 일과 다음 상태 정의
  - 규격상에서 불분명한 부분은 구현상의 문제

# 상태 천이도 (서버)


SEC_Create_Request.req /
T_Unitdata.req(HelloRequest)
NULL
SEC_Terminate.req /
T_Unitdata.req(Alert(c/f))
T_Unitdata.ind /
SEC_Exception.ind
[Alert(no_renego)]
SEC_Exception.req /
T_Unitdata.req(Alert(w)
)
T_Unitdata.ind /
SEC_Terminate.ind
[Alert(c/f)]
T_Unitdata.ind /
SEC_Create.ind
[ClientHello ]
CREATING
T_Unitdata.ind /
SEC_Create.ind
[ClientHello ]
: not identical to
previous -
received record
SEC_Create..es /
Create buffer :
ServerHello
Certificate*
SEC_Exchange.req /
Append buffer :
ServerKeyExchange*
CertificateRequest*
ServerHelloDone
T_Unitdata.req(buffer)
[Full Handshake]
SEC_Commit.req /
Append buffer :
ChangeCipherSpec
Finished
T_Unitdata.req(buffer)
[Optimized / Abreviated
Handshake]
SEC_Terminate.req /
T_Unitdata.req(Alert(c/f))
T_Unitdata.ind /
SEC_Terminate.ind
[Alert(c/f)]
SEC_Exception.req /
T_Unitdata.req(Alert(w))
T_Unitdata.ind /
SEC_Create.ind
[ClientHello ]
: not identical to
previous -received
record
T_Unitdata.ind /
SEC_Terminate.ind
[Alert(c/f)]
SEC_Terminate.req /
T_Unitdata.req(Alert(c/f))
T_Unitdata.ind /
SEC_Exception.ind
[Alert(w)]
SEC_Exception.req /
T_Unitdata.req(Alert(w))
T_Unitdata.ind /
SEC_Exception.ind
[Alert(w)]
T_Unitdata.ind /
T_Unitdata.req(last buffer)
[ClientHello ]
: identical to previous -
received record
T_Unitdata.ind /
SEC_Create..ind
[ClientHello ]
T_Unitdata.ind /
SEC_Exchange.cnf,
SEC_Commit.ind
[Certificate*, ClientKeyExchange*
CertificateVerify*,
ChangeCipherSpec, Finishec]
Receive C.C.S, Set R.C.S to P.S
Set R.S.N to zero
Create a new buffer :
[ChangeCipherSpec, Finished]
Set W.C.S to P.S by sending C.C.S
Set W.S.N to zero
T_Unitdata.req(buffer)
[ChangeCipherSpec, Finished]
T_Unitdata.ind /
T_Unitdata.req(last
buffer)
[Certificate*,
ClientKeyExchange*
CertificateVerify*,
ChangeCipherSpec,
Finished]
: a group of records
identical to previous-
received one
SEC_Terminate.req /
T_Unitdata.req(Alert(c/f))
T_Unitdata.ind /
SEC_Terminate.ind
[Alert(c/f)]
SEC_Exception.req /
T_Unitdata.req(Alert(w))
T_Unitdata.ind /
SEC_Exception.ind
[Alert(w)]
T_Unitdata.ind /
SEC_Exception.ind
[Alert(w)]
T_Unitdata.ind /
SEC_Unitdata.ind
[user data received]
SEC_Terminate.req /
T_Unitdata.req(Alert(c/f))
T_Unitdata.ind /
T_Unitdata.req(last
buffer)
[ClientHello ]
: identical to previous
-received record
T_Unitdata.ind /
Receive C.C.S, Set R.C.S to P.S
Set R.S.N to zero
SEC_Commit.cnf
SEC_Unitdata.ind
[ChangeCipherSpec, Finished
, user data]
[optimized / abbreviated
Handshake]
T_Unitdata.ind /
Receove C.C.S,
Set R.C.S to P.S
t R.S.N to zero
C_Commit.cnf
hangeCipherSpec
inished]
ll Handshake]
T_Unitdata.ind /
SEC_Exception.ind
[Alert(w)]
T_Unitdata.ind /
SEC_Unitdata.ind
[received user data]
SEC_Unitdata.req /
T_Unitdata.req
SEC_Create_Request.req /
T-Unitdata.req(HelloRequest)
T_Unitdata.ind /
T_Unitdata.req(Alert(dup_fin_recvd)
[ChangeCipherSpec, Finished]
SEC_Terminate.req /
T_Unitdata.req(Alert(c/f))
SEC_Exception.req /
T_Unitdata.req(Alert(w))
T_Unitdata.ind /
T_Unitdata.req(Alert(dup_fin_recvd)
[ChangeCipherSpec, Finished, User data]
: identical to previous Finished received

# 상태 천이도 (클라이언트)

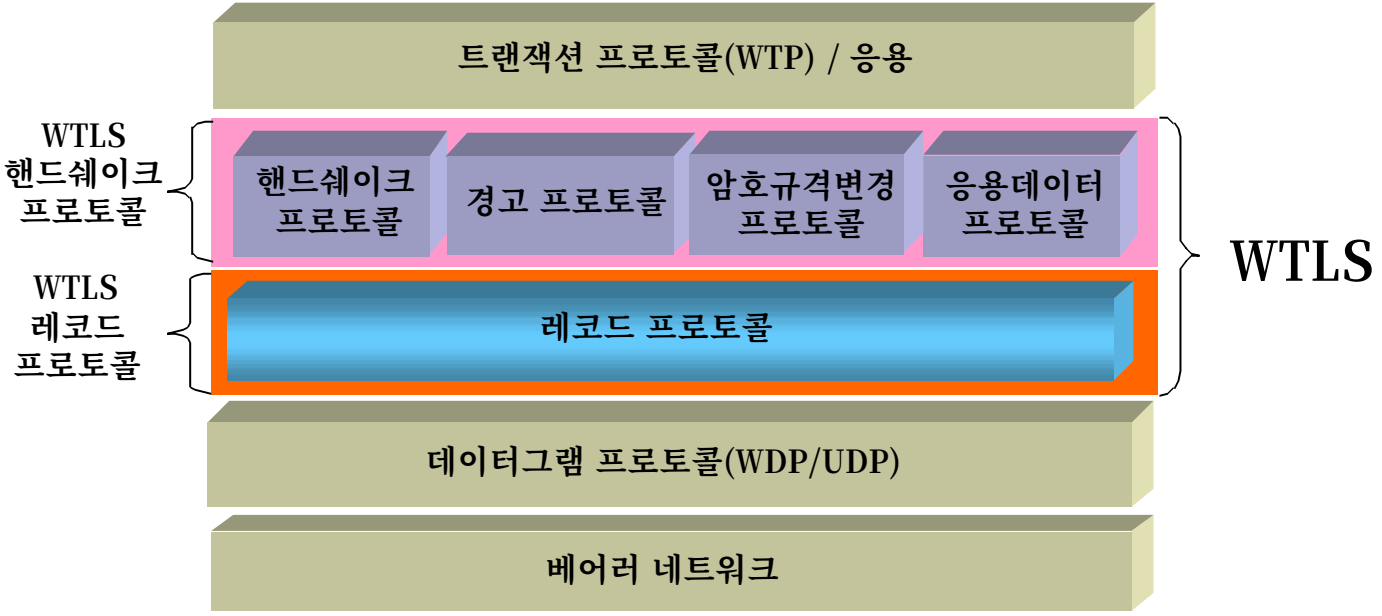





한국전자통신연구원 / 이석준

# WTLS 프로토콜

- WTLS 핸드쉐이크 프로토콜
  - 암호규격변경 프로토콜
  - 경고 프로토콜
  - 핸드쉐이크 프로토콜
- WTLS 레코드 프로토콜



# WTLS 핸드쉐이크 프로토콜 – 1/6

- WTLS 핸드쉐이크 프로토콜의 역할
  - 레코드 계층을 위해서 보안 파라미터에 대하여 서로 동의
  - 상호 인증
  - 협상된 보안 파라미터를 실제화(instantiate)
  - 서로에게 오류조건을 보고

# WTLS 핸드쉐이크 프로토콜 – 2/6

## WTLS 핸드쉐이크 프로토콜에서 합의해야 하는 세션 정보

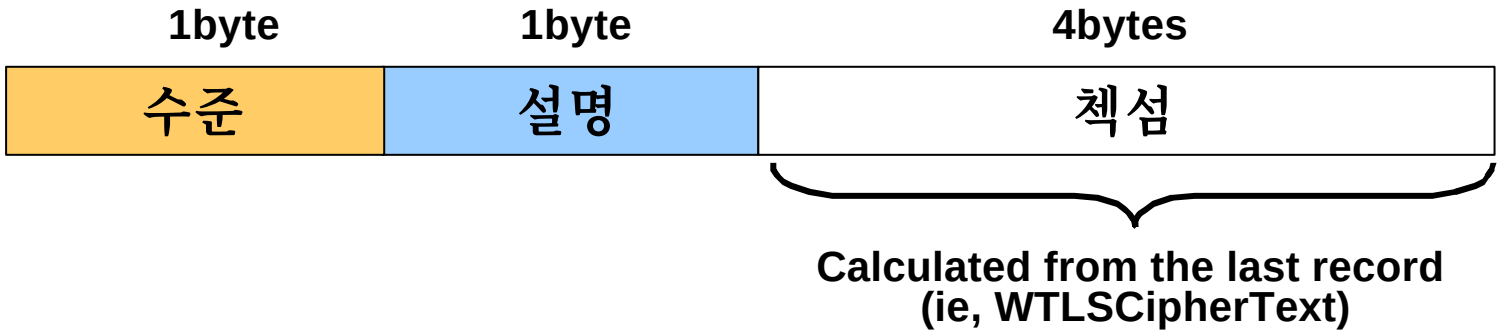
| 항목 | 설 명 |
|---|---|
| 세션 식별자 | 보안 세션을 식별하기 위해 서버가 선택한 임의의 바이트 열 |
| 프로토콜 버전 | WTLS 프로토콜 버전 넘버 |
| 상대방 인증서 | 상대방의 인증서로 NULL 가능 |
| 압축 방법 | 암호화전 압축할 때 사용할 알고리즘 |
| 암호 규격 | Bulk cipher 알고리즘과 MAC 알고리즘 규격 |
| 매스터 시크릿 | 서버와 클라이언트가 공유하는 20바이트 비밀 정보 |
| 순서번호 방식 | 현재 보안 세션에 적용되고 있는 순서번호 방식 (미적용, 묵시적, 명시적) |
| 키 갱신 주기 | 암호화 키, MAC 비밀값, IV 등의 연결 상태 정보가 얼마나 자주 계산되어야 하는 가의 정보 |
| 세션 재사용성 | 보안 세션이 새로운 보안 연결의 생성에 사용될 지 아닐지를 지정 |

# WTLS 핸드쉐이크 프로토콜 – 3/6

- 암호규격변경 프로토콜
  - 암호규격변경을 알리기 위해 존재
  - 하나의 1 byte 메시지로 구성
  - 현재 연결 상태로 암호화
    - 이 프로토콜에 의한 메시지 이후 새로운 연결 상태로 변경
  - 암호 파라미터의 상호간의 동의 이후 전송
  - Finished 메시지 생성/검증 전에 전달

# WTLS 핸드쉐이크 프로토콜 – 4/6

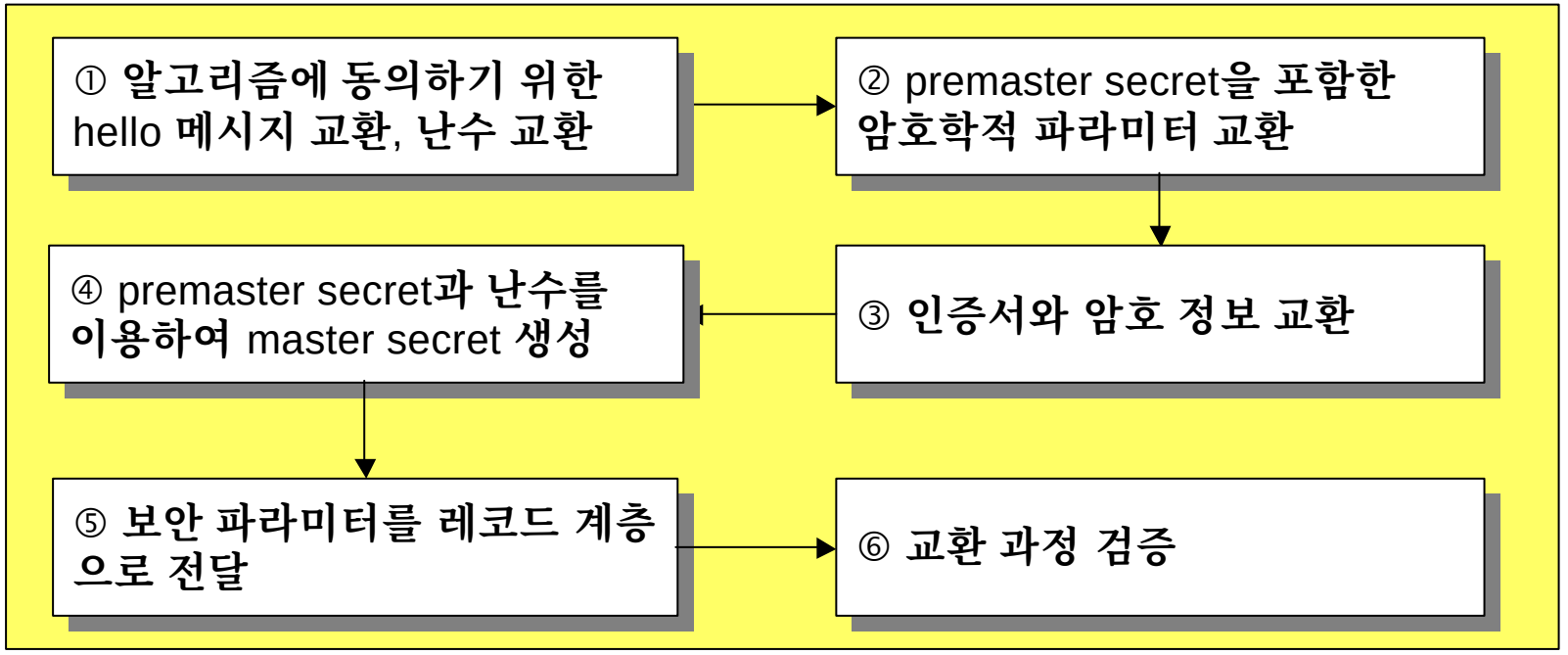
- 경고 프로토콜
  - 경고 수준과 경고 설명, 체섬(checksum)으로 구성
  - 경고 수준
    - Fatal/Critical/Warning
  - 경고 설명
    - 특정 상황 발생 사유에 대하여 설명
  - 체섬
    - 상대방으로부터 전달 받은 마지막 레코드로부터 계산



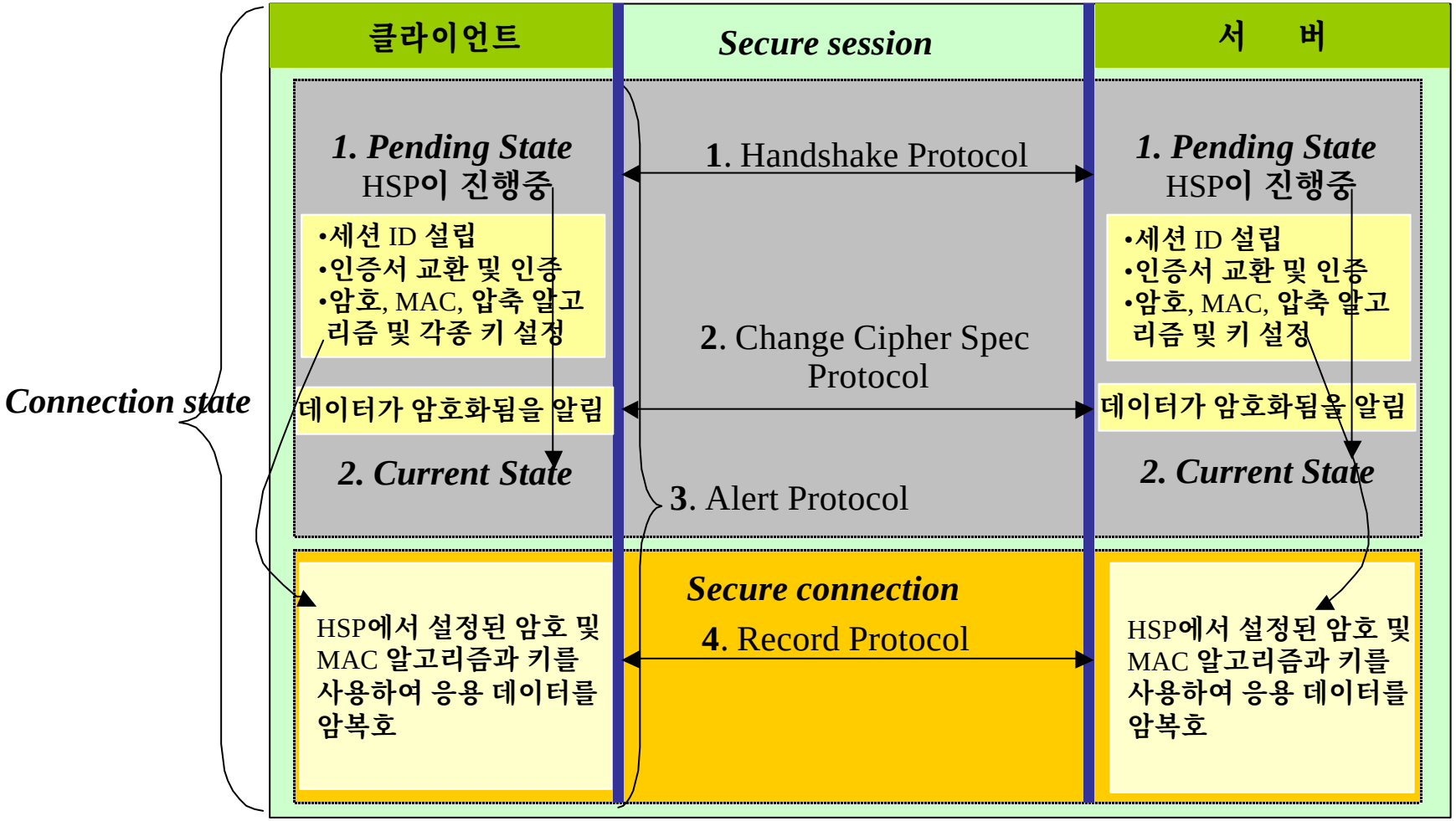
# WTLS 핸드쉐이크 프로토콜 – 5/6

## 핸드쉐이크 프로토콜

- 보안 세션의 암호학적 파라미터 생성
- 프로토콜 버전 동의, 암호 알고리즘 선택, 상호 인증
- (대칭) 키 공유를 위해 공개키 암호 기술 사용

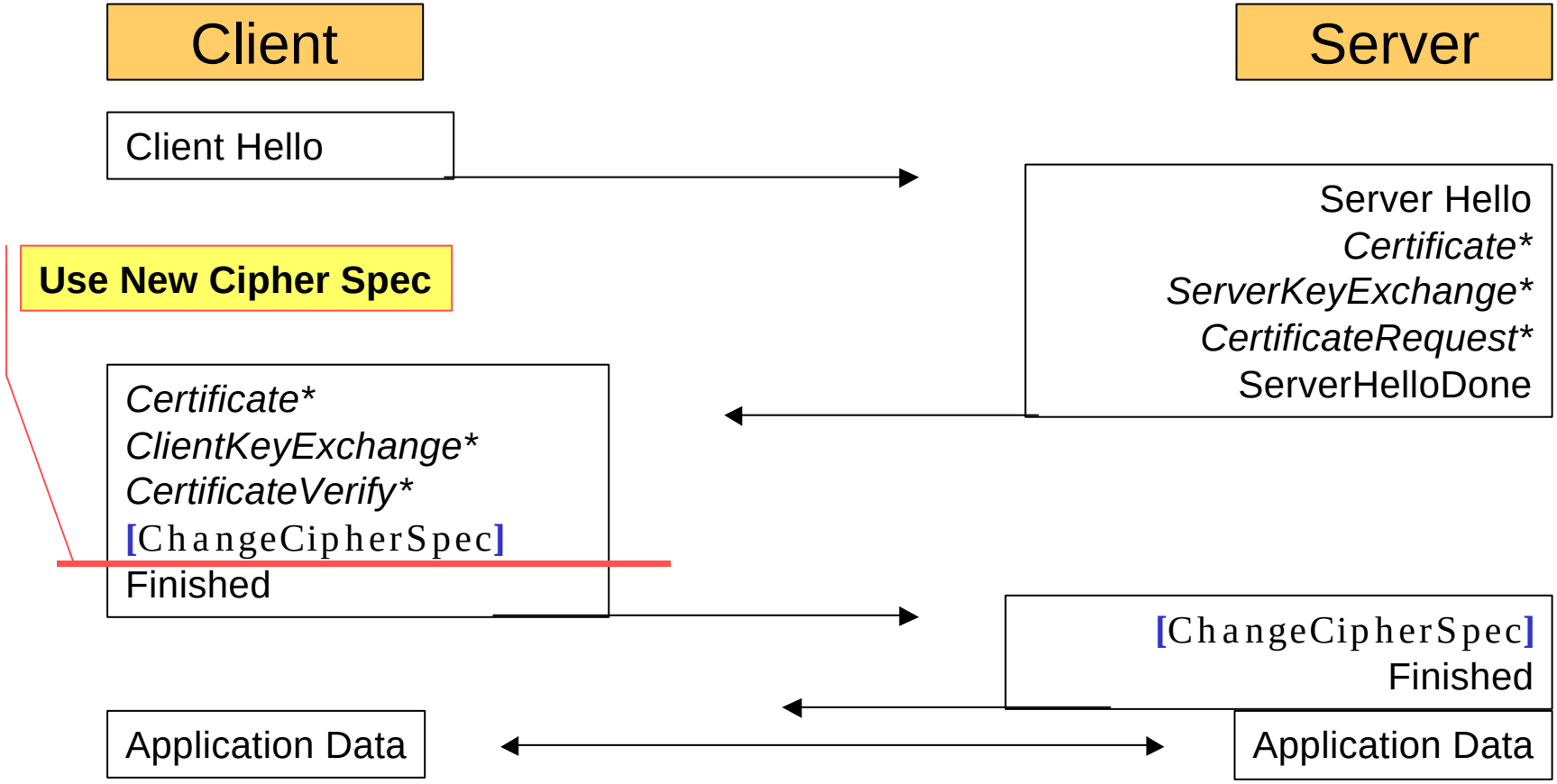


# WTLS 핸드쉐이크 프로토콜 – 6/6



# 핸드쉐이크 종류 – 1/4

## Full Handshake

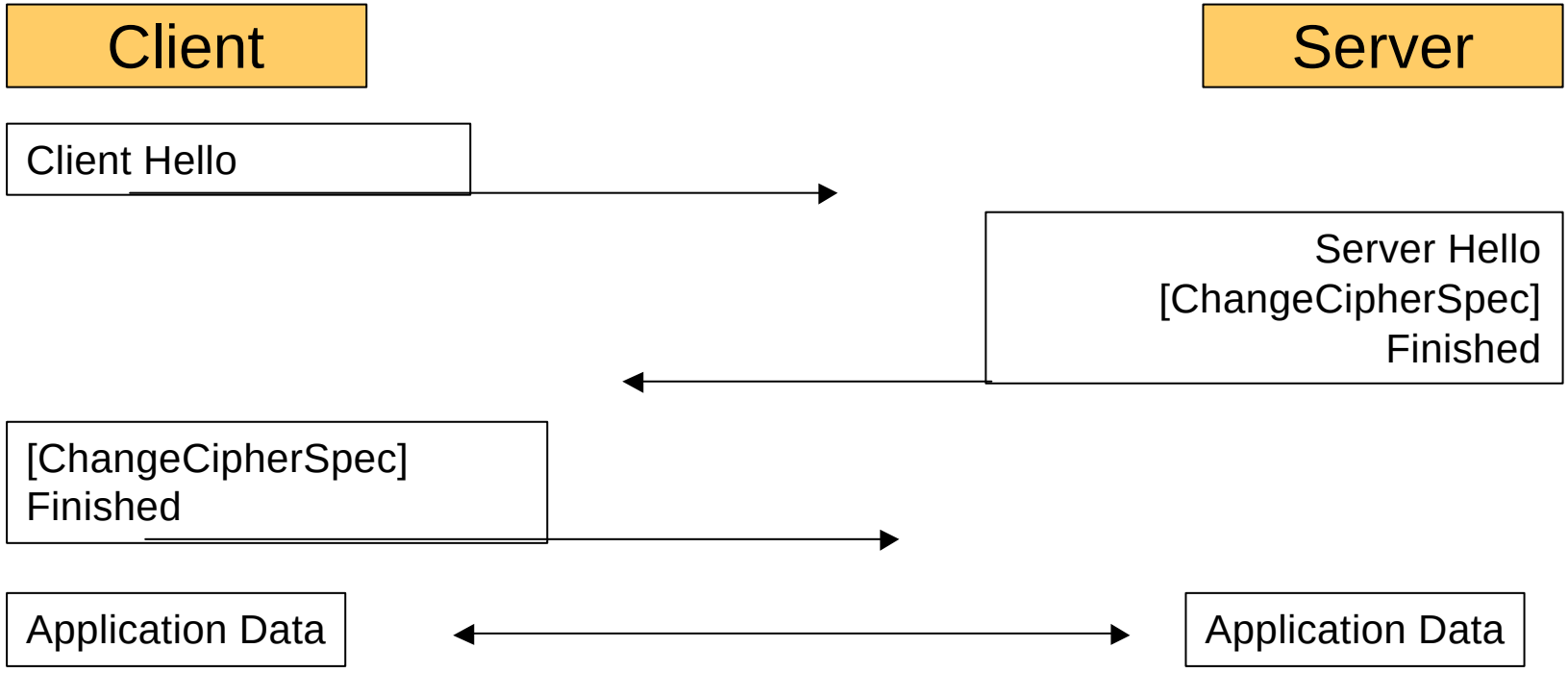



[ ] not including handshake messages(excepted at hash)

* *optional or situation-dependent messages*

# 핸드쉐이크 종류 – 2/4

- Abbreviated Handshake
  - 이전의 보안 세션을 재개하기 위한 핸드쉐이크

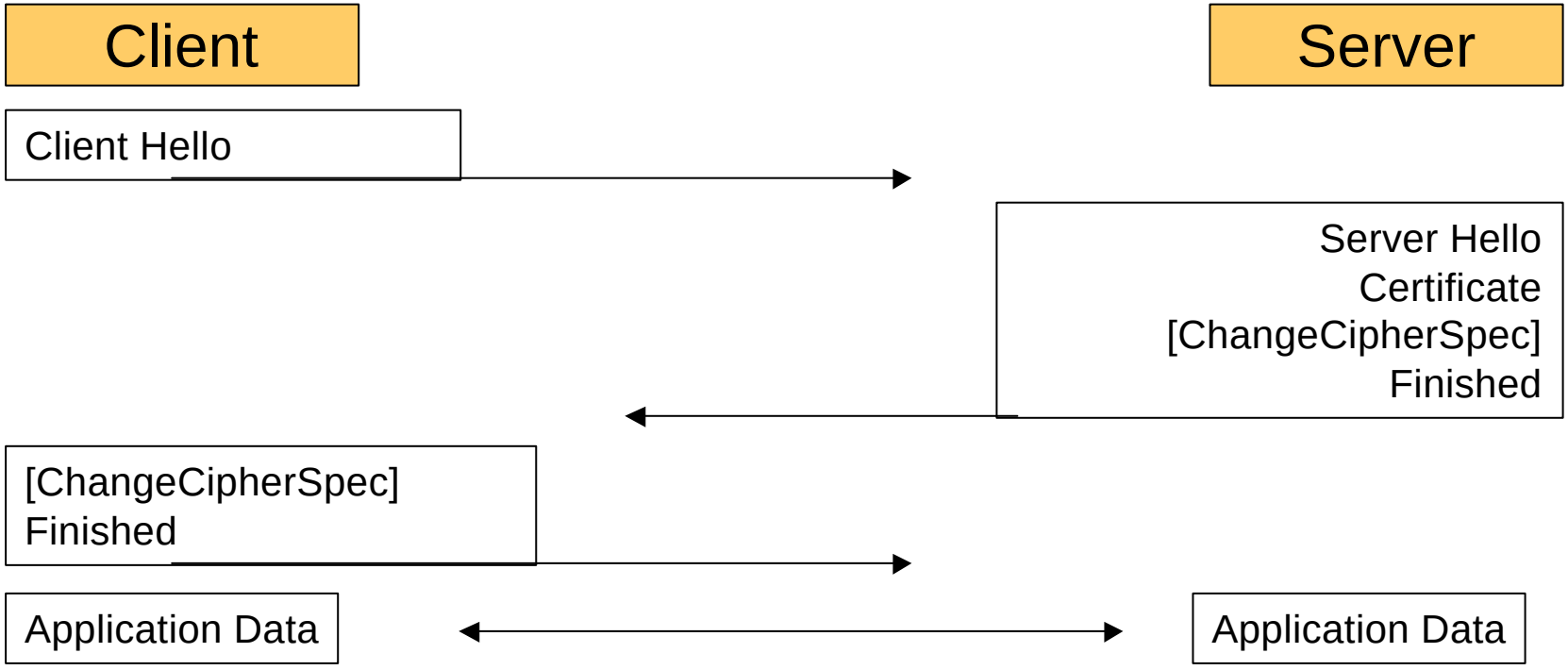


# 핸드쉐이크 종류 – 3/4

- Shared secret handshake
  - 양단간에 기본적으로 shared secret이 있다는 가정하에 시작됨
  - Shared secret : pre-master secret and SHARED_SECRET key exchange
  - Abbreviate Handshake와 다음의 내용을 제외한 모든 부문이 동일
    - ClientHello.session_id = NULL
    - ClientHello.client_key_ids = SHARED_SECRET
  - resumable

# 핸드쉐이크 종류 – 4/4

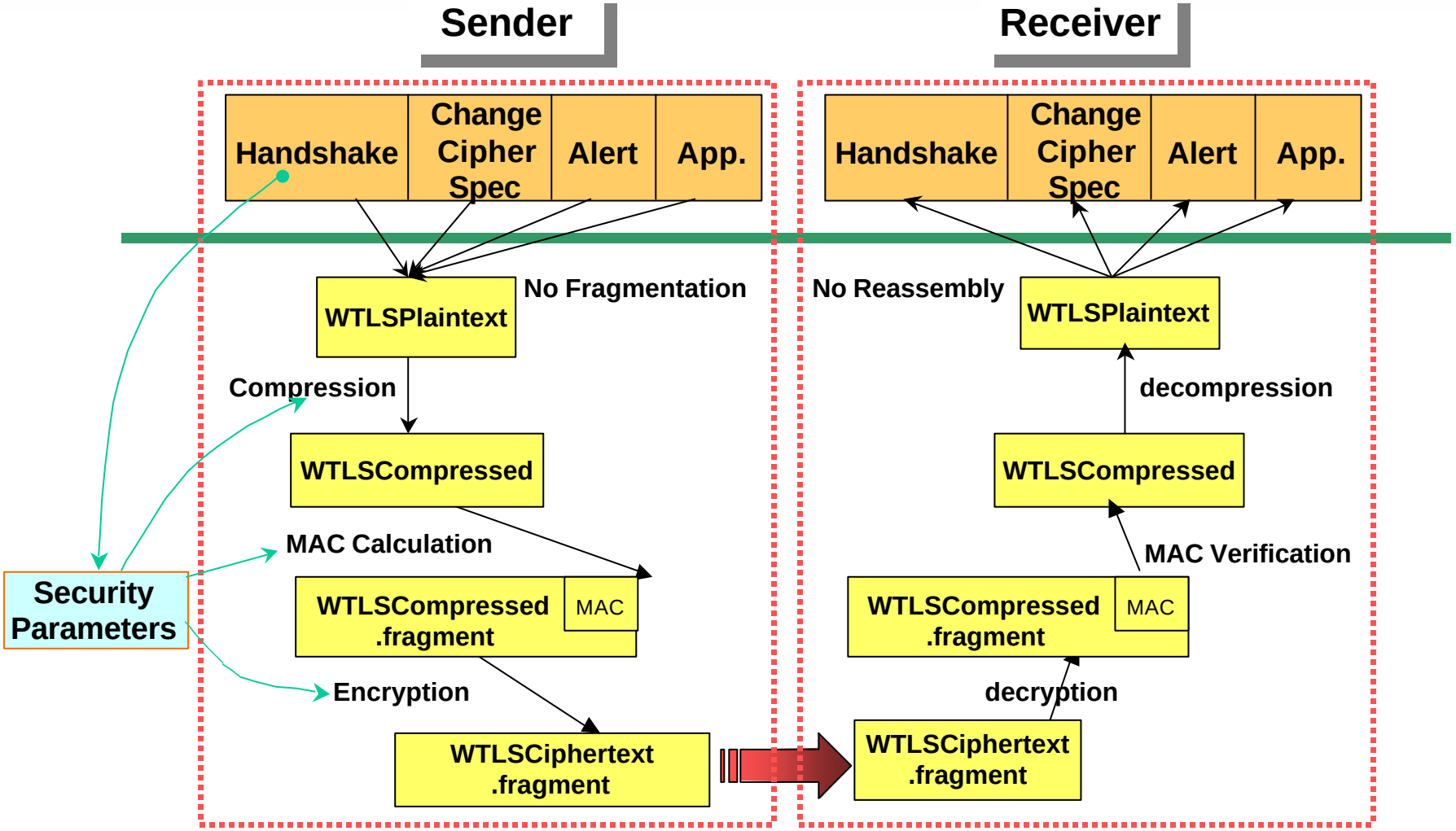
- Optimized Full Handshake
  - 서버가 client certificate를 다른 전송 경로를 통하여 얻고,
  - 이 Client Certificate가 EC Diffie-Hellman type key exchange method를 사용한 경우



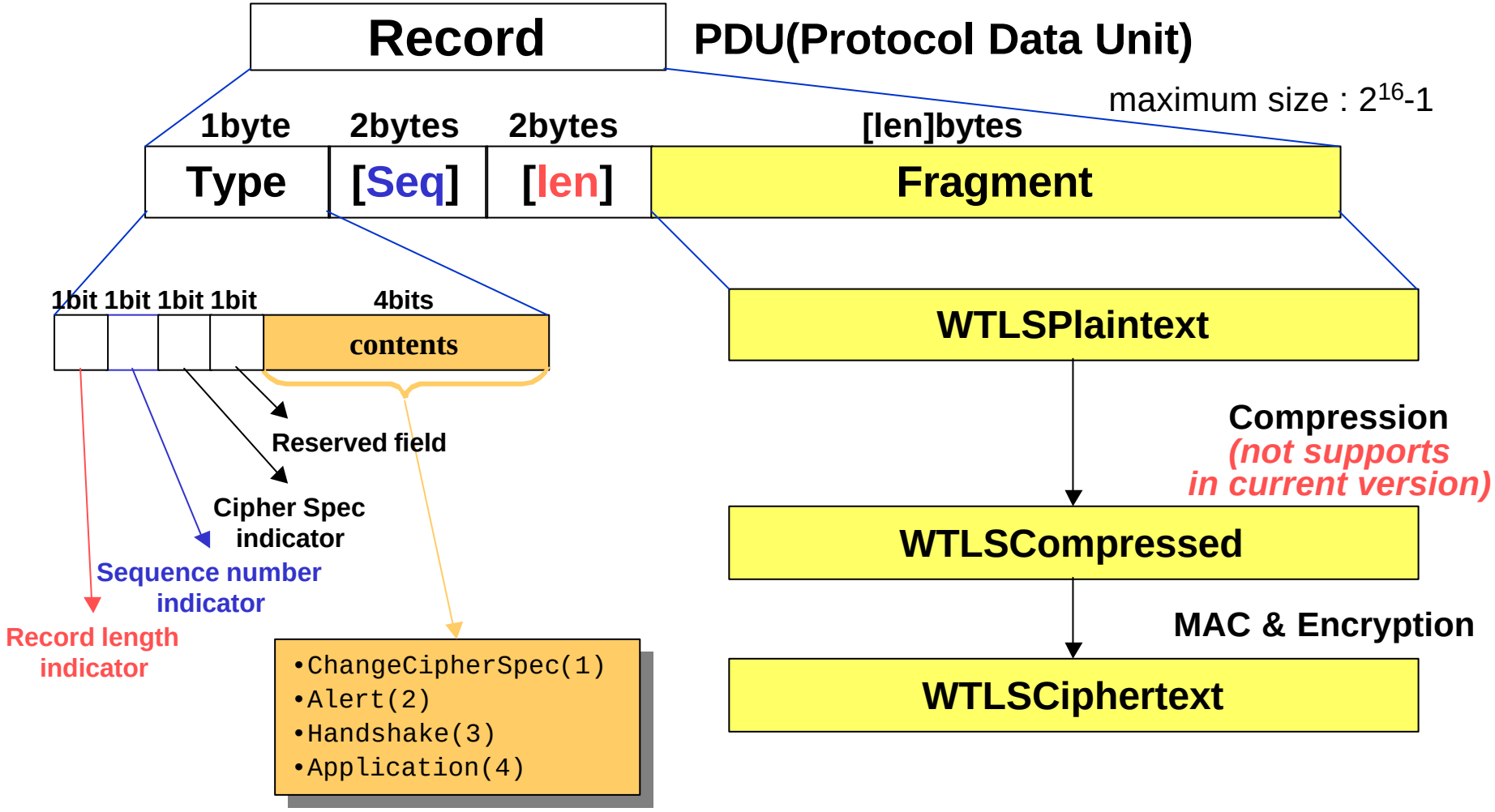
# WTLS 레코드 프로토콜 – 1/2

- 레코드 계층
  - 상위 계층으로부터 데이터를 받음
  - 단편화는 수행하지 않음
  - 압축/압축풀기
    - 현재 버전에서는 지원하지 않음
  - 명시적 순서 번호
    - 데이터그램 전송 프로토콜에 신뢰성 제공
  - 암/복호화
    - 기밀성, 무결성 제공
    - CBC Block cipher, MAC

# WTLS 레코드 프로토콜 – 2/2



# WTLS 레코드 구조


Record
PDU(Protocol Data Unit)
maximum size : 2^16-1
1byte
2bytes
2bytes
[len]bytes
Type
[Seq]
[len]
Fragment
1bit 1bit 1bit 1bit
4bits
contents
Reserved field
Cipher Spec indicator
Sequence number indicator
Record length indicator
•ChangeCipherSpec(1)
•Alert(2)
•Handshake(3)
•Application(4)
WTLSPlaintext
Compression
(not supports in current version)
WTLSCompressed
MAC & Encryption
WTLSCiphertext

# 4. TLS와의 관계

# TLS와의 비교

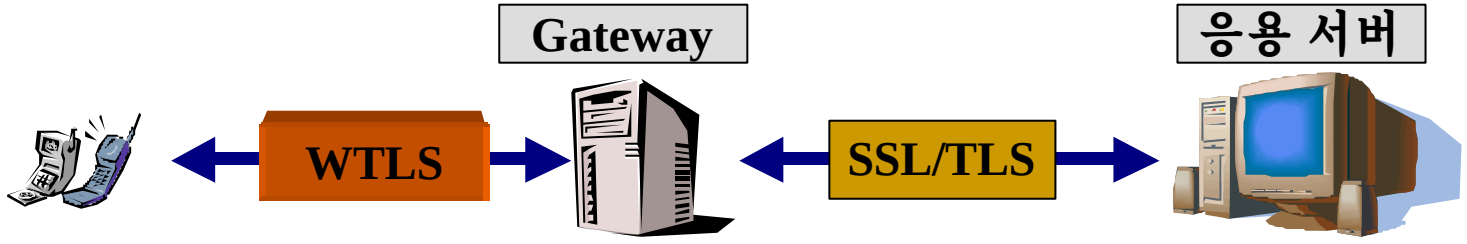
- 전송계층
  - WTLS는 데이터그램 전송계층 프로토콜도 지원
- 핸드쉐이크
  - WTLS에 Optimized/Shared-Secret 핸드쉐이크 추가
- Key Refresh
  - WTLS에서 사용함
- 레코드 프로토콜
  - WTLS는 Fragment를 수행하지 않음
  - TLS는 상위 프로토콜로부터 임의의 크기를 가지는 데이터가 전달되는데 반해 WTLS는 $2^{16}-1$ bytes의 데이터만 전달
- 경고 프로토콜
  - WTLS에 Critical 수준의 레벨 추가
  - 보다 더 구체적인 경고 설명

# TLS와의 비교

- 사용가능한 알고리즘
  - WTLS에 ECC 추가
- 사용가능한 인증서
  - WTLS에 X9.62, WTLS, URL 인증서 타입 추가
- 암호학적 계산
  - Master secret : 48 bytes vs. 20 bytes
  - Random number : 32 bytes vs. 16 bytes
  - WTLS에서는 클라이언트와 서버가 다르게 키 블록 생성
  - WTLS에서는 키 블록 생성시 순서 번호 첨가하여 생성
- WIM과의 연동 가능성
  - WTLS는 WIM과 연동 가능

# TLS와의 비교

- WTLS에 보안 홀의 존재?
  - 보안 홀은 WAP 모델상의 문제일 뿐, WTLS의 구조적인 문제는 아님
  - WTLS 자체는 TLS와 마찬가지로 암호학적으로 E2E security 제공



# WAP WSG-IETF TLS 협력 움직임

- Proposal for WAP-IETF cooperation on a wireless friendly TLS
  - By Tim Wright (WSG 의장)
  - 2000년 8월 IETF 정기 모임에서 발표
  - 발표 내용
    - WSG의 계획
      - 인증서는 가급적 X.509에 따름
      - WIM의 사용
      - IETF와 함께 wireless-friendly TLS 1.1 공동개발
    - 고려 사항
      - ECC, RC5의 사용
      - 클라이언트의 URL 인증서 사용
      - 클라이언트는 서버의 최상위 인증기관에 대한 정보를 가지고 있음
      - Datagram과 WTLS 인증서 지원은 생각하지 않을 수 있음
    - TLS 1.0에 수정 및 차기 버전(TLS 1.1)에 대한 요구 사항 정리 요구

# WAP WSG-IETF TLS 협력 움직임

- 무선 환경을 고려한 TLS 확장 규격 제안
  - By Simon Blake-Wilson, Magnum Nystrom
  - 2000년 12월 IETF 정기 모임에서 토론
  - 제안 내용
    - Record size의 최대값 협상
    - 클라이언트 URL 인증서 사용 가능
    - 클라이언트는 그가 가지고 있는 CA root key를 서버에게 알림
    - 서버가 작은 세션 ID를 사용할 것을 권장
    - 크기를 줄인 MAC의 사용 가능
    - CRL대신 OCSP 사용 가능
  - 현재 제안서 재정리중

# 5. 결 론

# 결 론

- WTLS 규격 (안)
  - WAP Forum의 WTLS 1.2를 근간으로 함
  - 무선인터넷에서 통신하는 두 주체를 위한 정보보호 서비스의 제공
  - 이 규격에 따르는 기기, 응용 개발시 구현 적합성 여부 확인
  - 무선전송계층 보안프로토콜로서의 요구 사항 수용

- Future Works
  - 국내 실정에 맞는 수정 필요
    - SEED, HAS 등의 표준에 대한 정리 필요
    - 국내법과의 관계 정리 필요
  - IETF TLS working group과 WAP Forum의 동향을 계속 지켜볼 필요가 있음