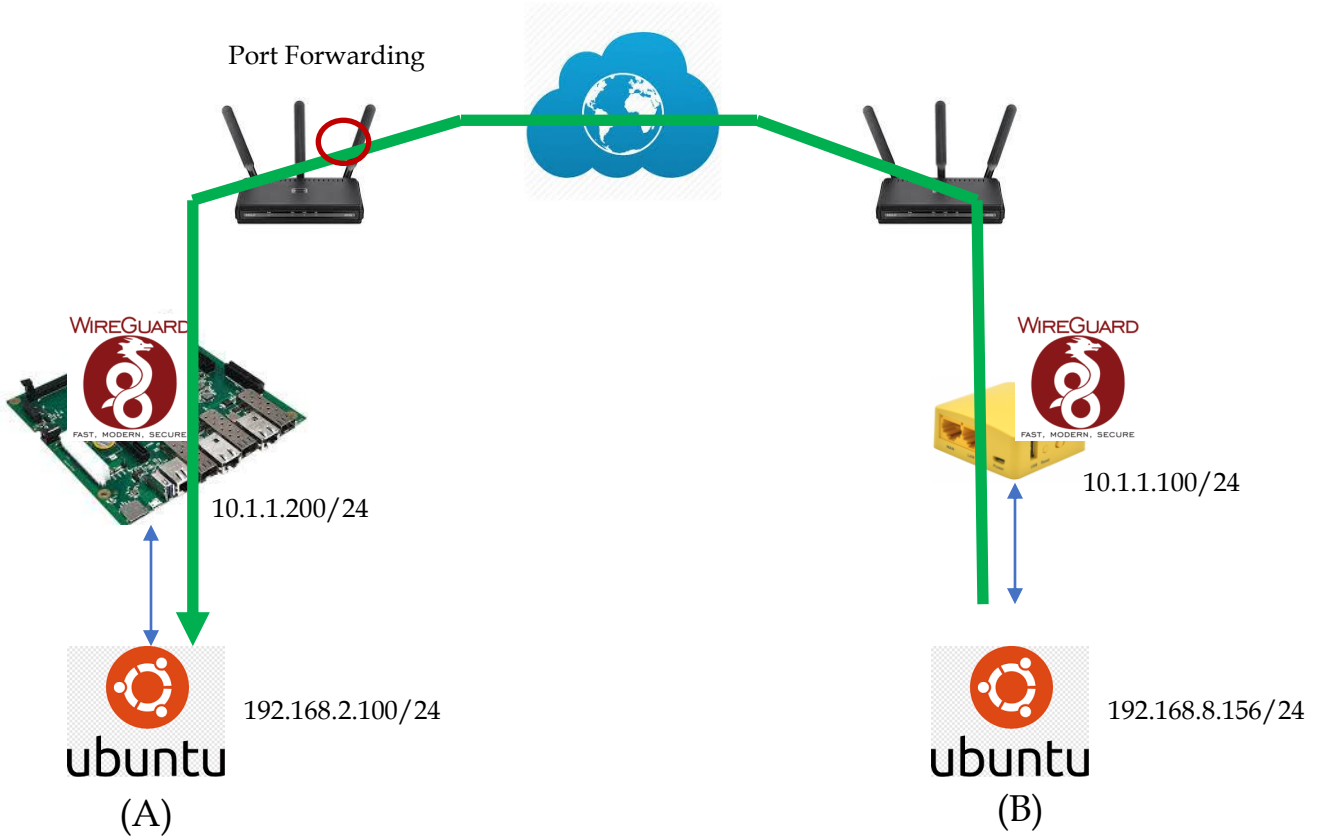
WireGuard Analysis

- WireGuard Protocol 및 코드 분석 -

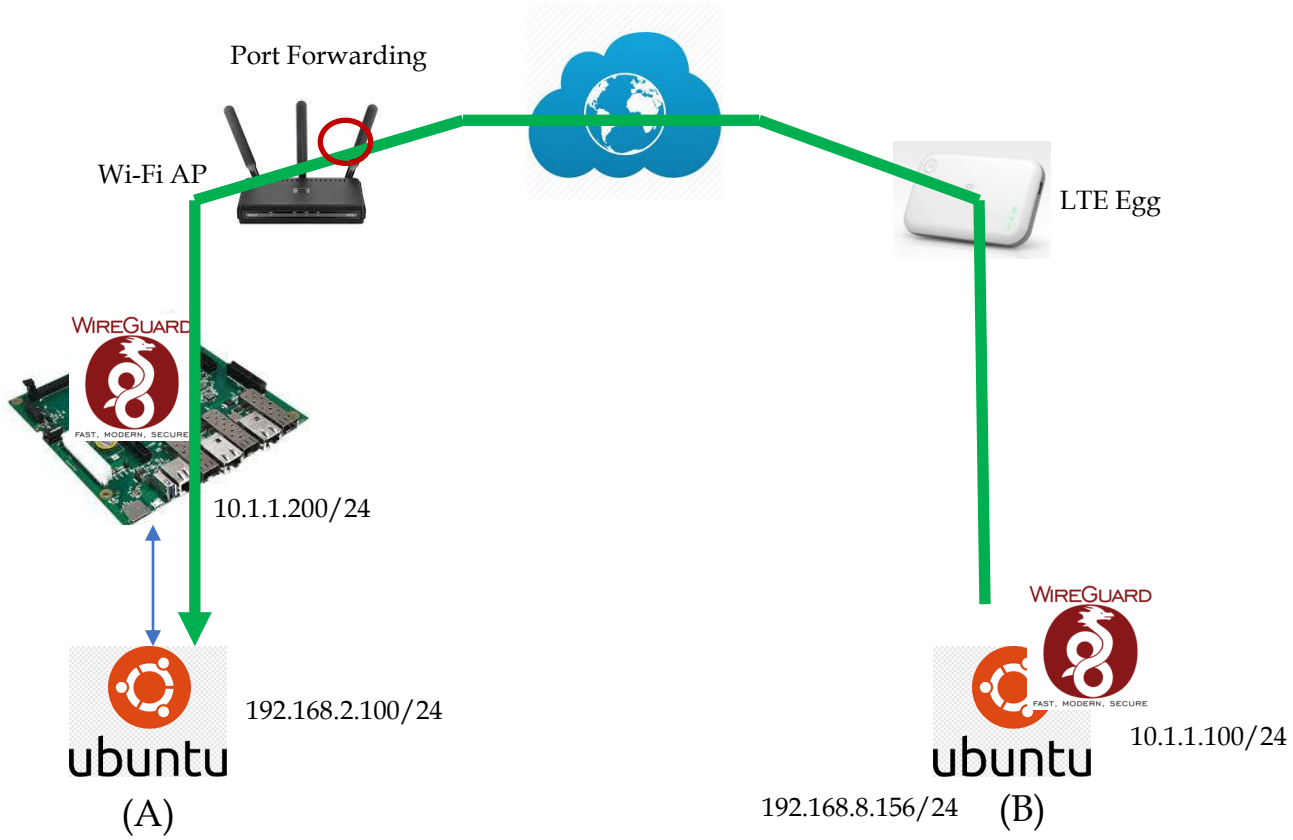
Chunghan.Yi(chunghan.yi@gmail.com)
Doc. Revision: 0.3

Table of Contents

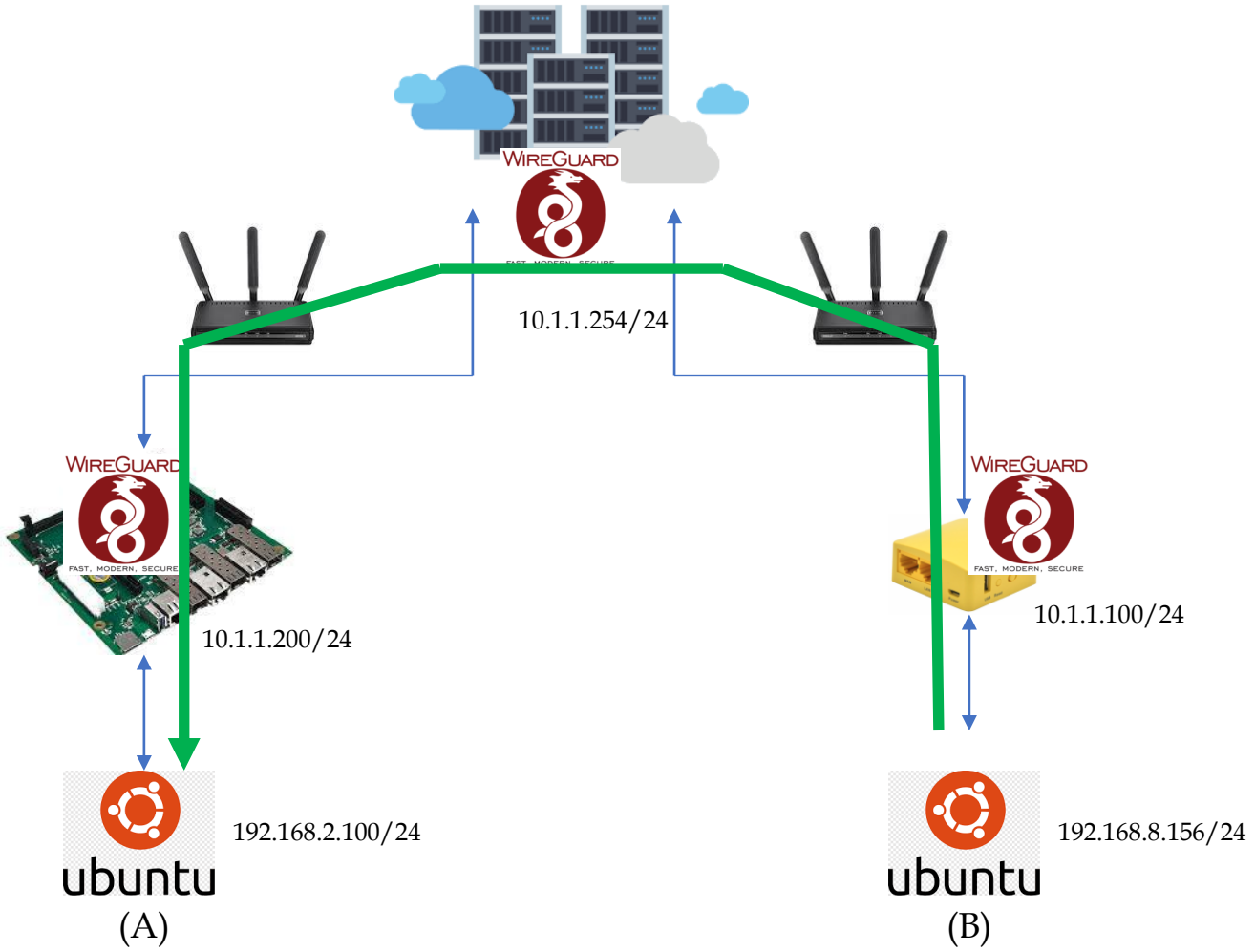
1. WireGuard Overview(1) – Direct Connection(1)



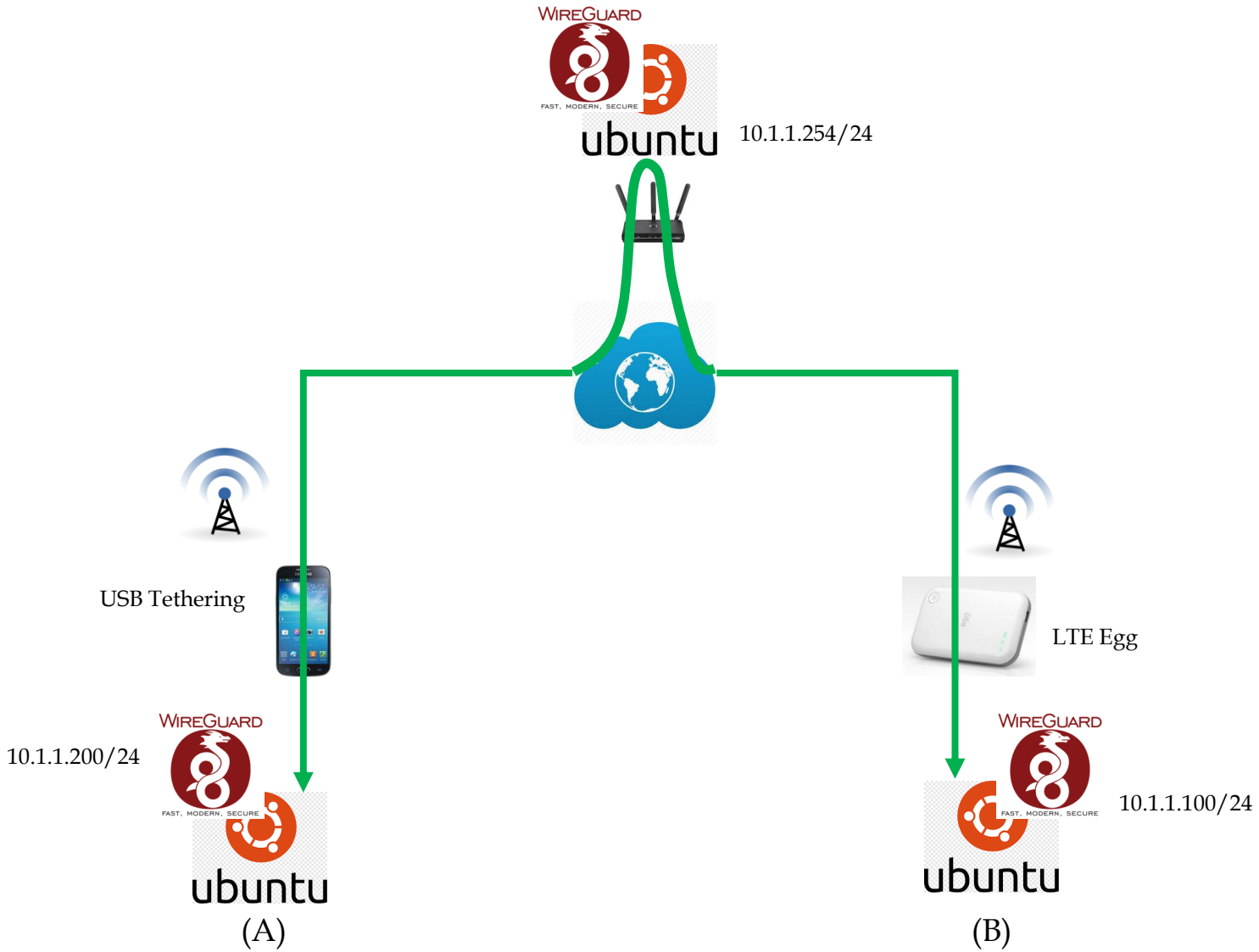
1. WireGuard Overview(1) – Direct Connection(2)



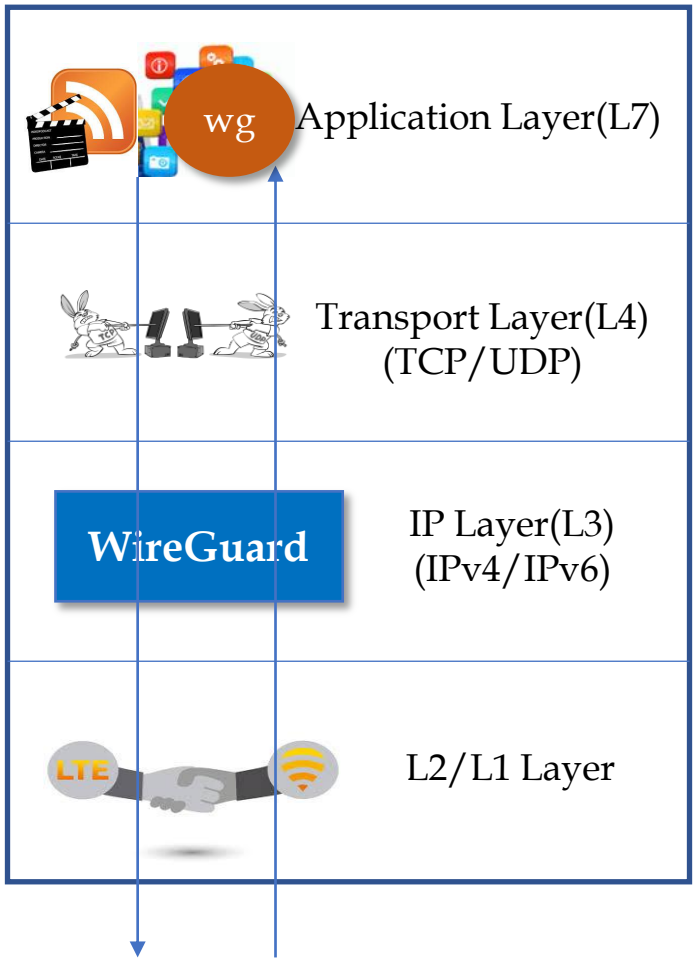
1. WireGuard Overview(2) – Relay Connection(1)

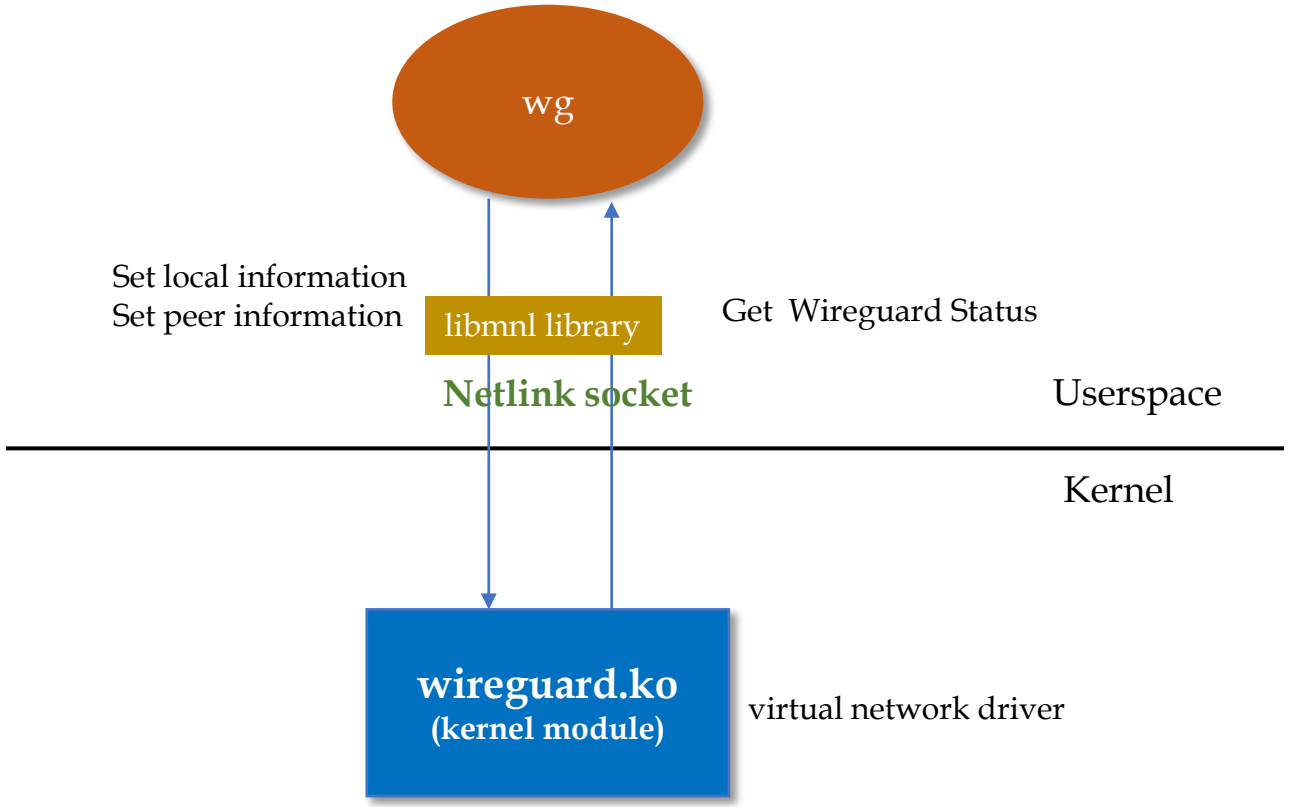


1. WireGuard Overview(2) – Relay Connection(2)



1. WireGuard Overview(3) – S/W Architecture(1)





1. WireGuard Overview(3) - S/W Architecture(2)

ECC(타원곡선/Curve25519)

WireGuard Tunnel Identifier
- Public key
- IP address(private ip address)

Public/private key 생성

사전 Public key 전달 과정

Public/private key 생성

Key Exchange
(ECDH: 상호 인증 & 세션 키 교환)

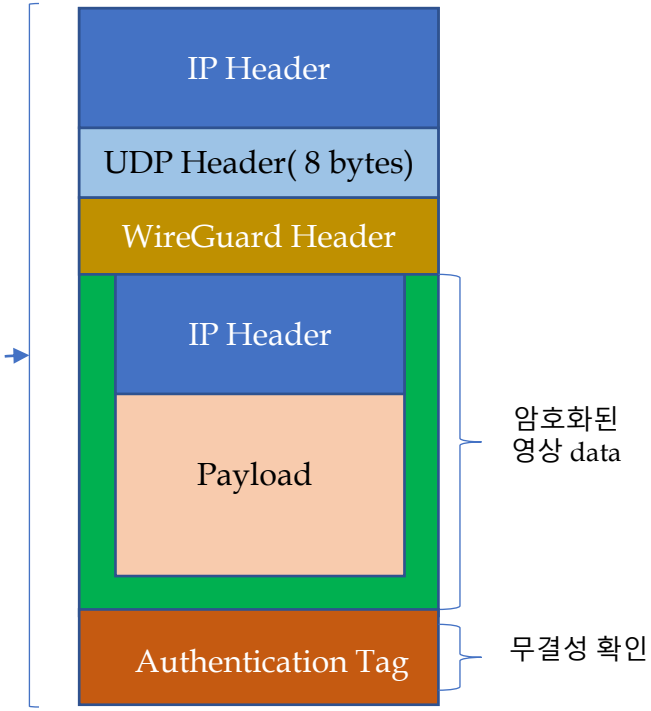
Session key(암호 키)

암/복호화 과정
(Symmetric Stream Cipher)

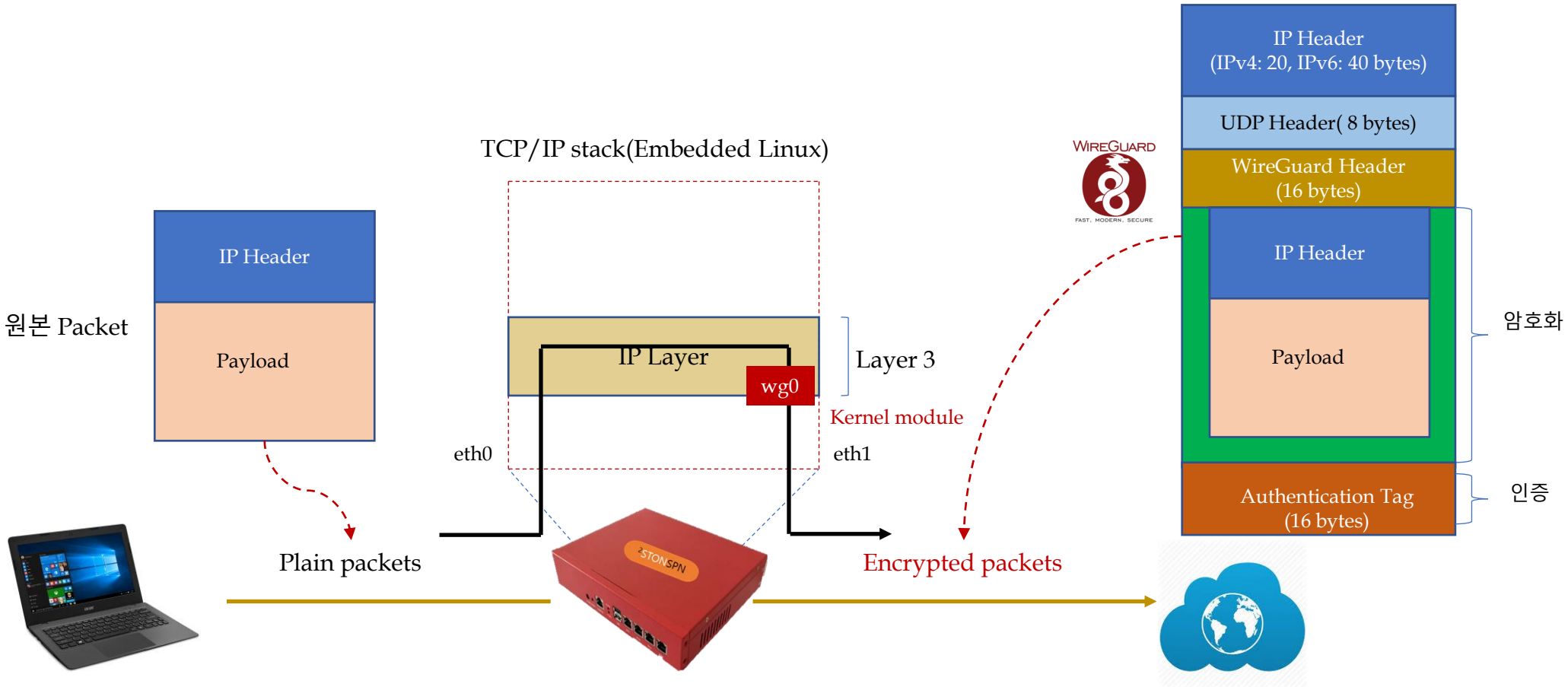
IP Tunnel Packet 생성

Linux Kernel

WIREGUARD
FAST, MODERN, SECURE



1. WireGuard Overview(3) - S/W Architecture(3)



1. WireGuard Overview(4) – Crypto Algorithms(1)

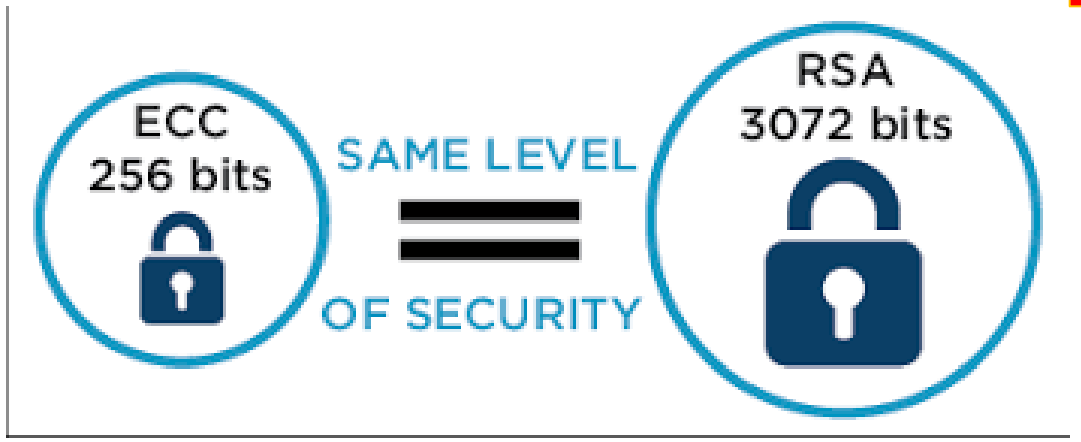
보안 알고리즘	상세 내용
Key 교환 방식 및 상호 인증	**NoiseIK handshake 방식(Noise IKpsk2)** ▪ ECDH(Diffie-Hellman) 기반 ▪ Curve25519 Public key(32 byte)를 교환 후, 이를 통해 안전하게 shared secret 생성 ▪ Static/Ephemeral public key(2개) 이용 ▪ Key 교환 시 아래 기능 보장 ▪ *키 침해 신분 위장 방지 기능, replay attack 방지 기능* ▪ *Perfect forward secrecy 보장, Identity 감춤 기능 제공, DoS 공격 완화 기능(Cookie)* **Hash 알고리즘** ▪ **BLAKE2s** – fast secure hashing 알고리즘 ▪ SHA series 보다 빠름. 즉 MD5 수준임.
암호 알고리즘	**ChaCha20** – 256 bit stream cipher(20 round cipher Salsa20 기반) ▪ Stream cipier는 일반적인 block cipher(예: AES-256-CBC)에 비해 속도가 빠름 ▪ key(32 bytes)는 대칭키를 사용(즉, 암호화 용 키와 복호화 용 키 동일) ▪ **Video/Audio 등 stream 암호화에 적합**
무결성(Integrity) 검사 알고리즘	**Poly1305** - message authentication code 알고리즘(16 byte output 생성)

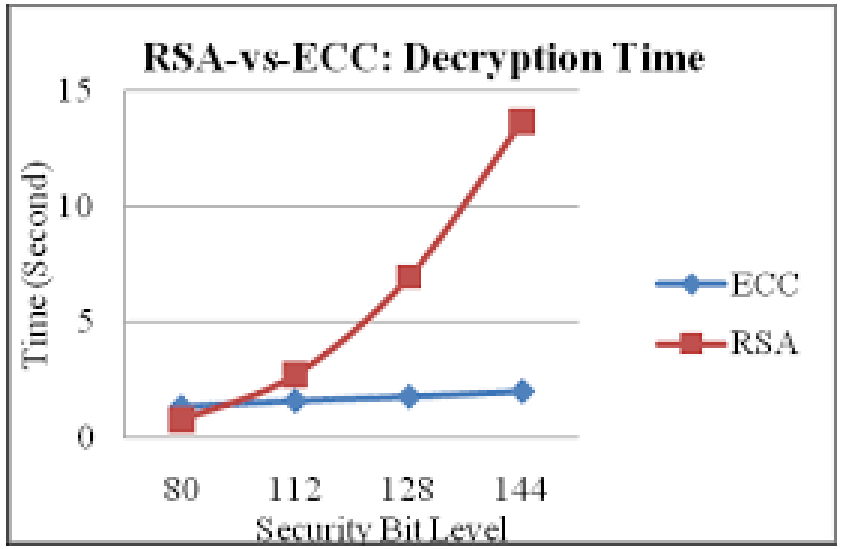
(*) 최대한 안전하면서도 빠른 알고리즘을 선택하므로써 전체적으로 network 성능을 끌어 올리도록 함.

1. WireGuard Overview(4) – Crypto Algorithms(2)

1단계: 상호 인증과 암호키 교환

ECC(Curve25519), Hash(Blake2s) => ECDH





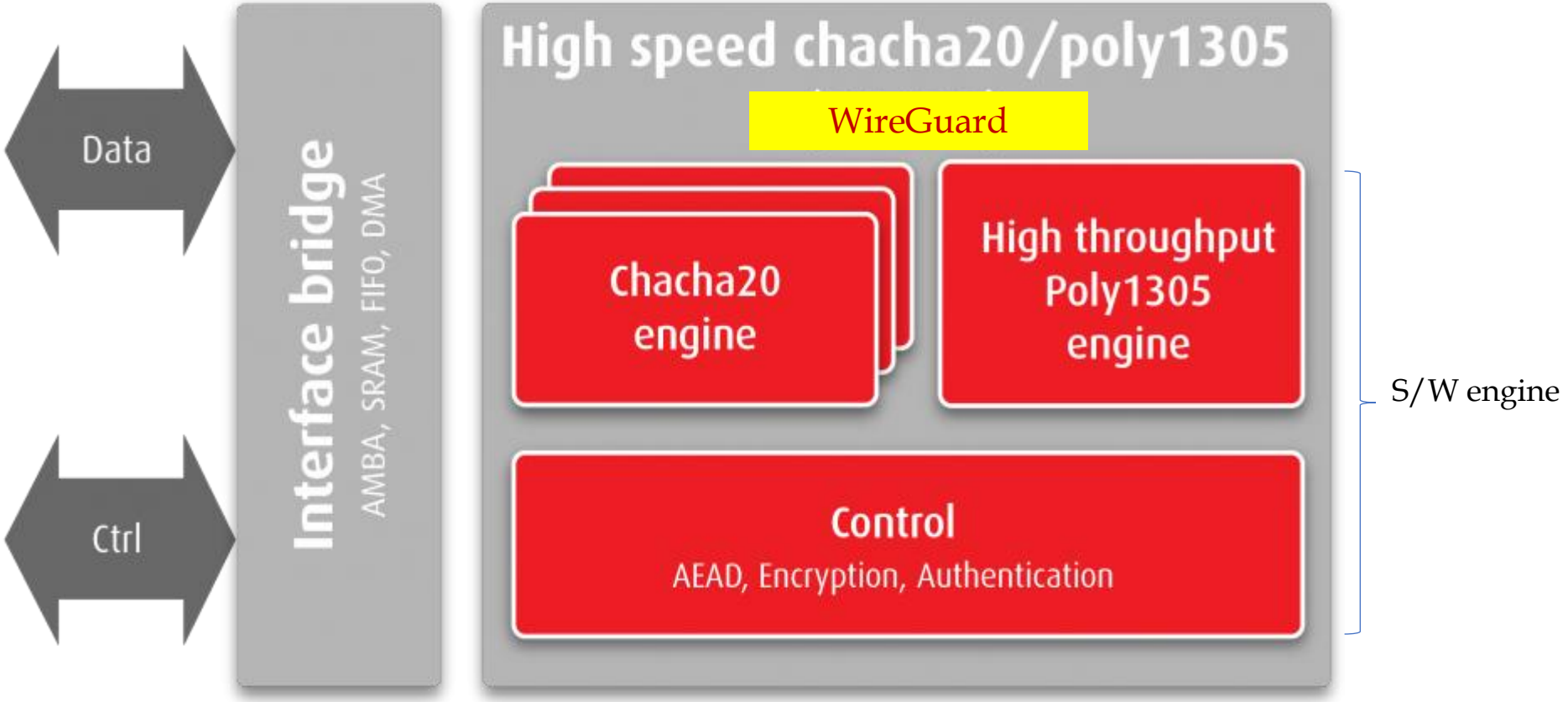
Blake2s > SHA Series

Speed

WireGuard는 ECC 알고리즘 기반의 ECDH 기법을 사용하여 상호인증 및 키 교환을 하게 됩니다.

1. WireGuard Overview(4) – Crypto Algorithms(3)

2단계: Authenticated Encryption



WireGuard는 Kernel 수준에서 Assembly 언어로 최적화된 ChaPoly 알고리즘을 사용하므로, 빠른 속도를 자랑합니다.

1. WireGuard Overview(5) – Roaming(1)

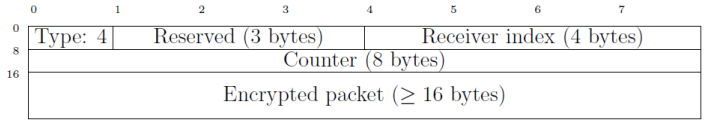
WireGuard는 상대방의 IP가 변해도 연결을 유지할 수 있습니다.

IP changed

WireGuard Tunnel
- ✓ Peer's public key로 관리됨
- ✓Peer's public key | ip address

Public key가 동일할 경우, src ip가 변경되더라도 이를 update해 주게 됨. 즉 같은 tunnel로 인식 !

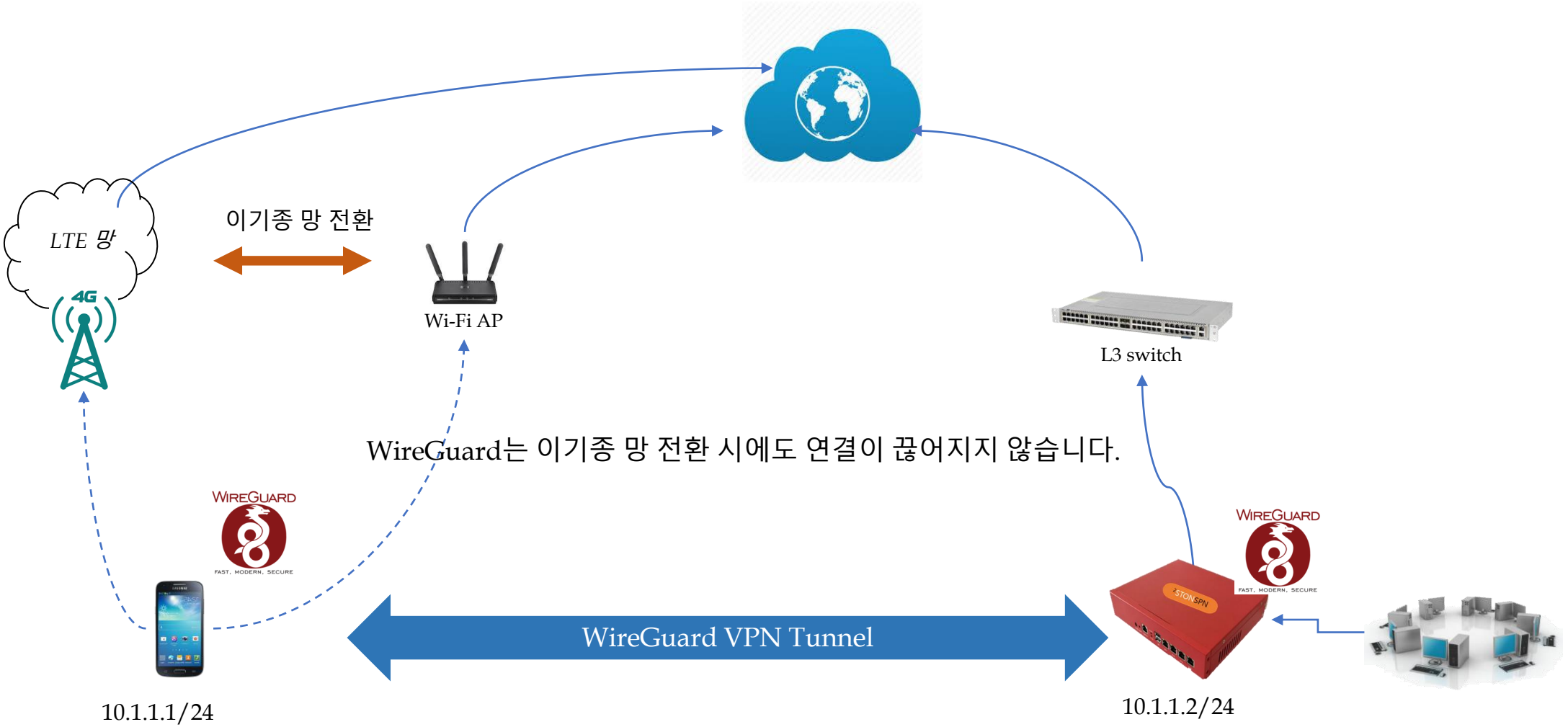
Index for Public key in WireGuard header



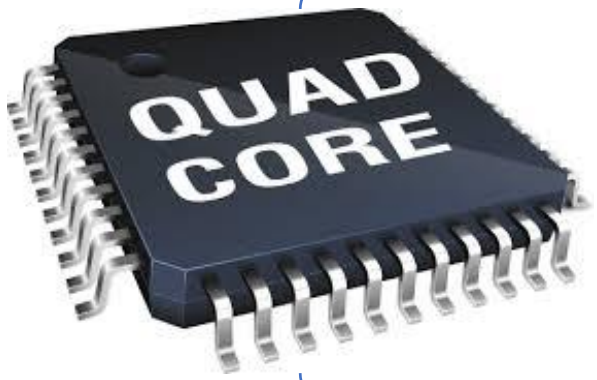
Receiver Index for public key in WireGuard header

WireGuard VPN Tunnel

1. WireGuard Overview(5) – Roaming(2)



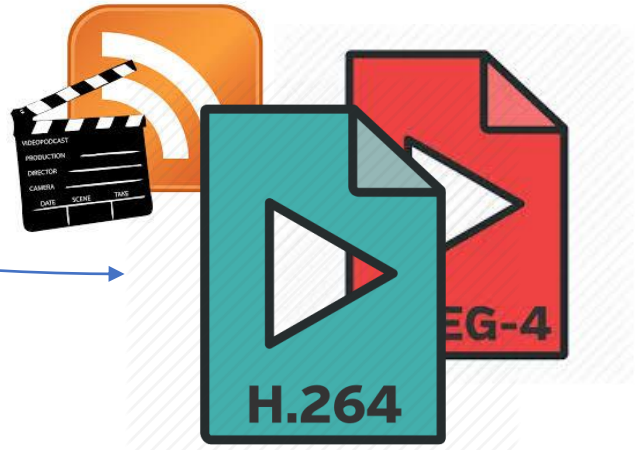
1. WireGuard Overview(6) – CPU Optimization



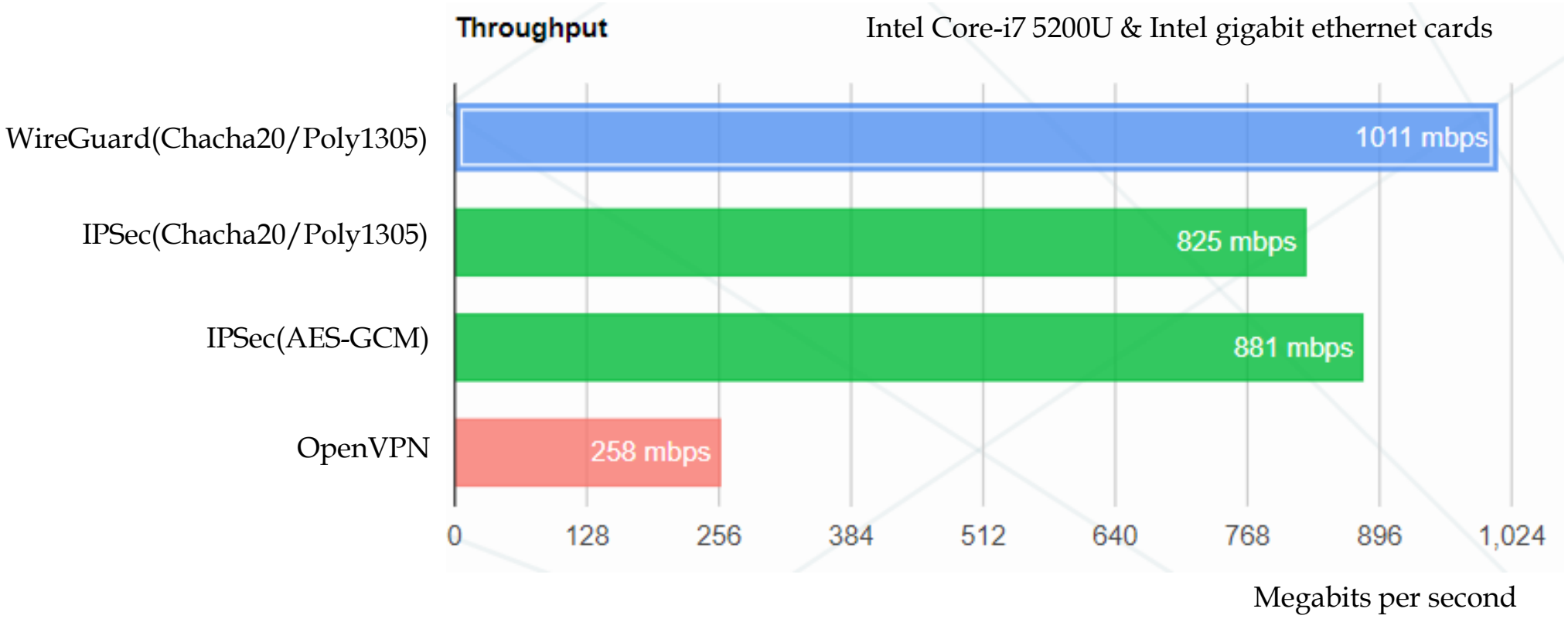
Encryption
Decryption
(ChaCha20)

동시에 복수개의 Encryption과 Decryption을 수행함.

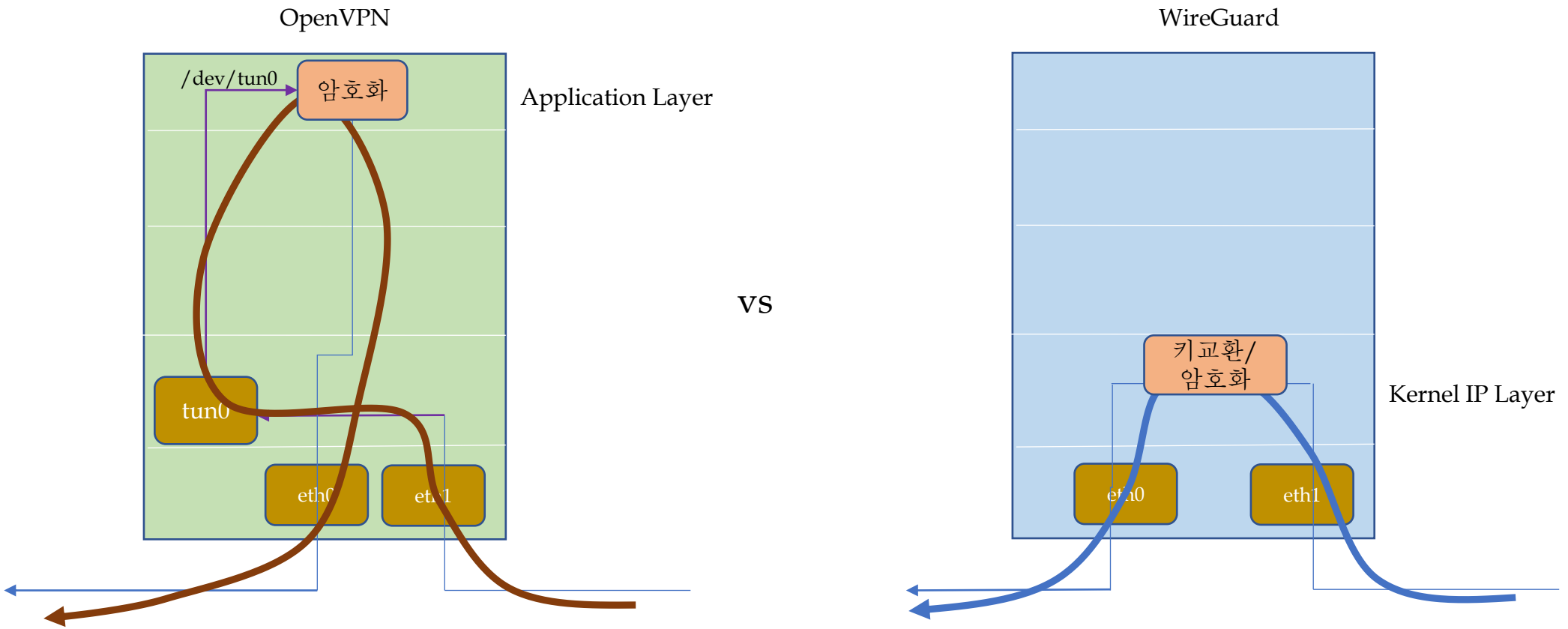
WireGuard는 multi-core CPU에 최적화되어 있습니다.



1. WireGuard Overview(7) – Fast Speed(1)

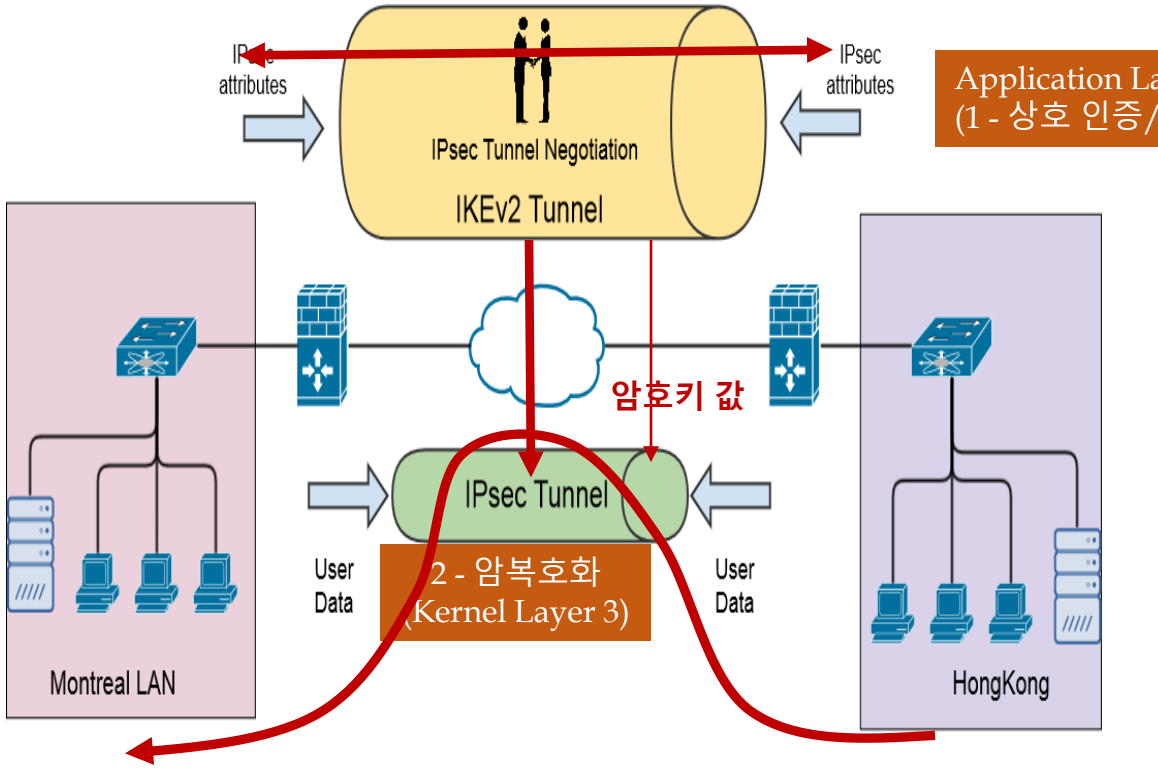


1. WireGuard Overview(7) – Fast Speed(2)

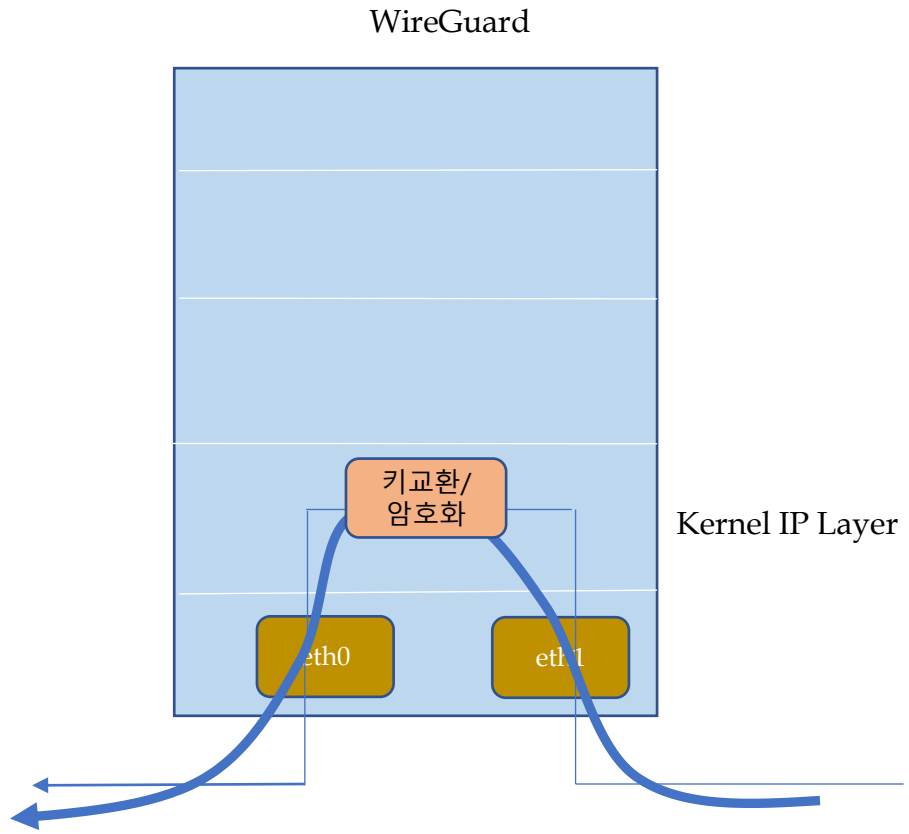


WireGuard가 OpenVPN 보다 왜 빠른가 ?

1. WireGuard Overview(7) – Fast Speed(3)



vs

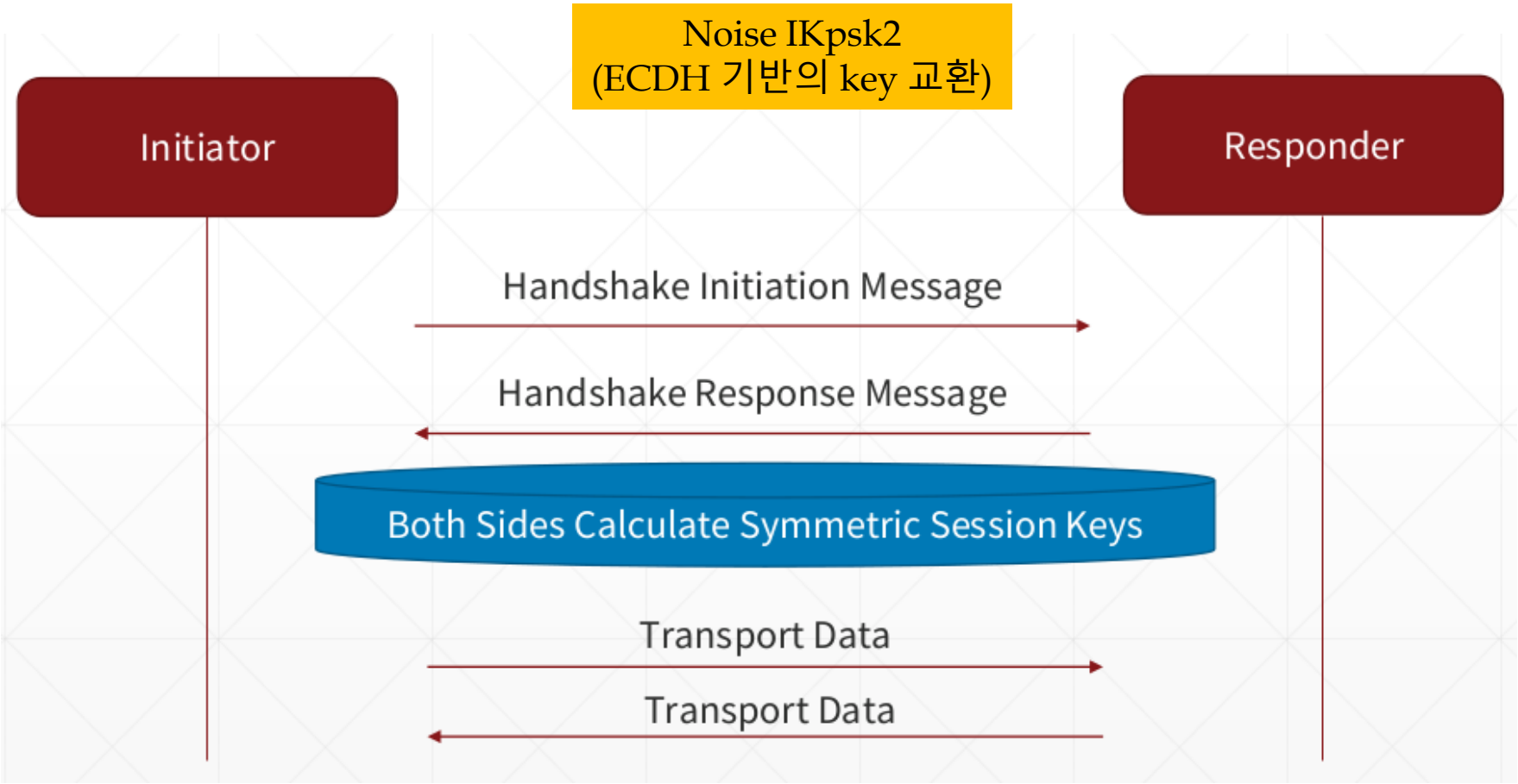


왜 WireGuard가 IPsec 보다 빠르고 간결한가 ?
(IKEv2 < Noise_IKpsk2)

1. WireGuard Overview(8) – How to setup

- https://slowbootkernelhacks.blogspot.com/2020/09/wireguard-vpn.html
- https://slowbootkernelhacks.blogspot.com/2020/04/espressobin-wireguard-vpn.html
- https://slowbootkernelhacks.blogspot.com/2020/05/openwrt-gainstrong-minibox3-wireguard.html

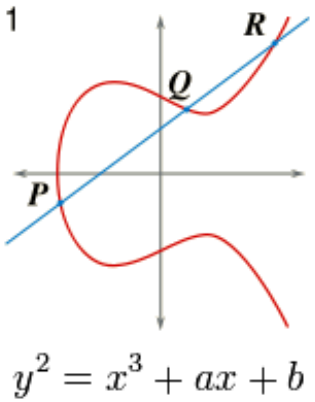
2. WireGuard Protocol(1) – Key Exchange(1)



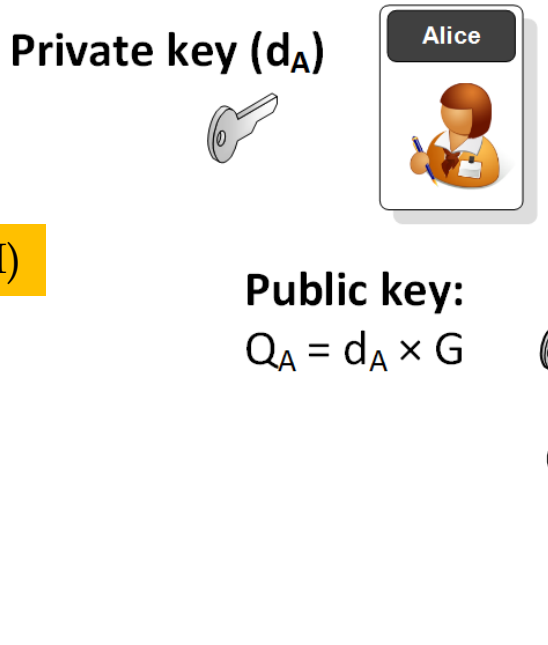
출처: 참고문헌 3

2. WireGuard Protocol(1) – Key Exchange(2)

Curve25519(ECDH)

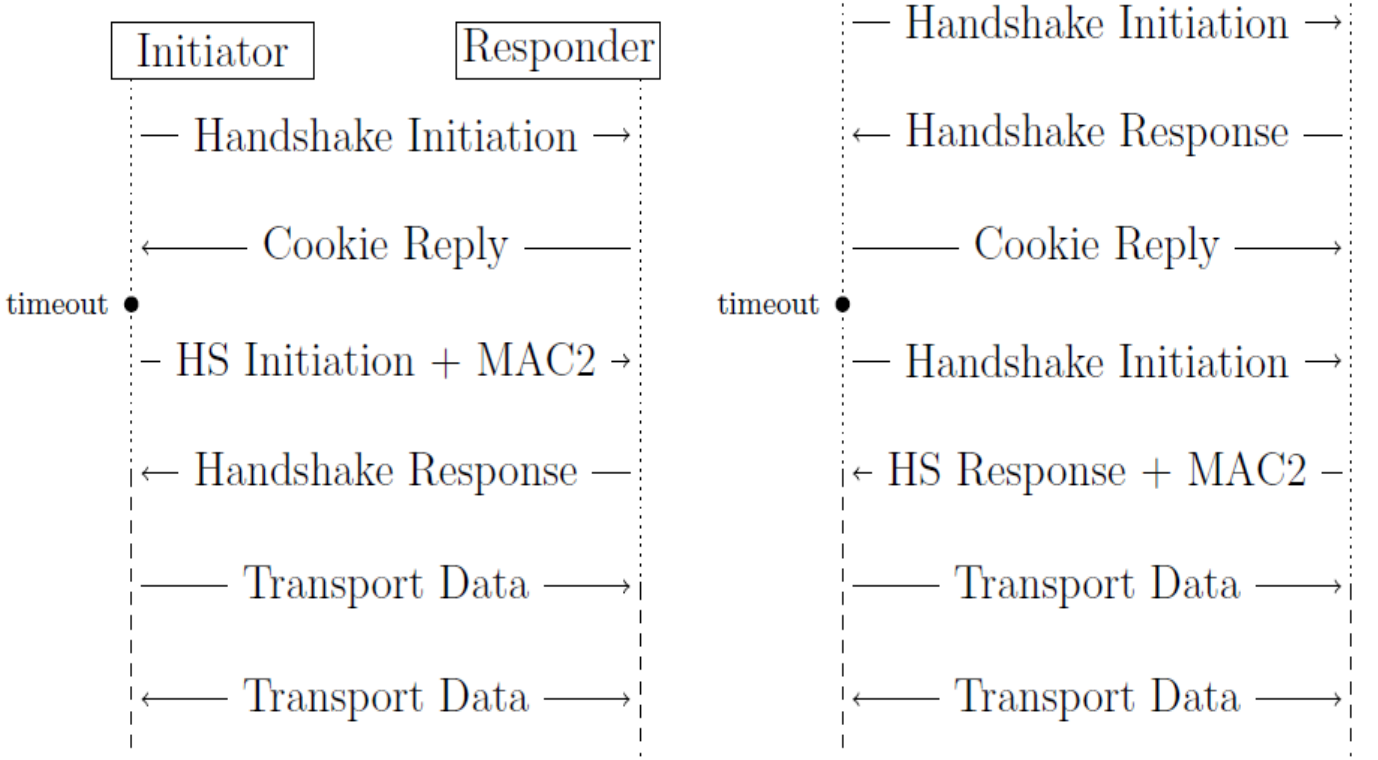


$$y^2 = x^3 + ax + b$$



2. WireGuard Protocol(2) – Cookie

DoS Attack Mitigation



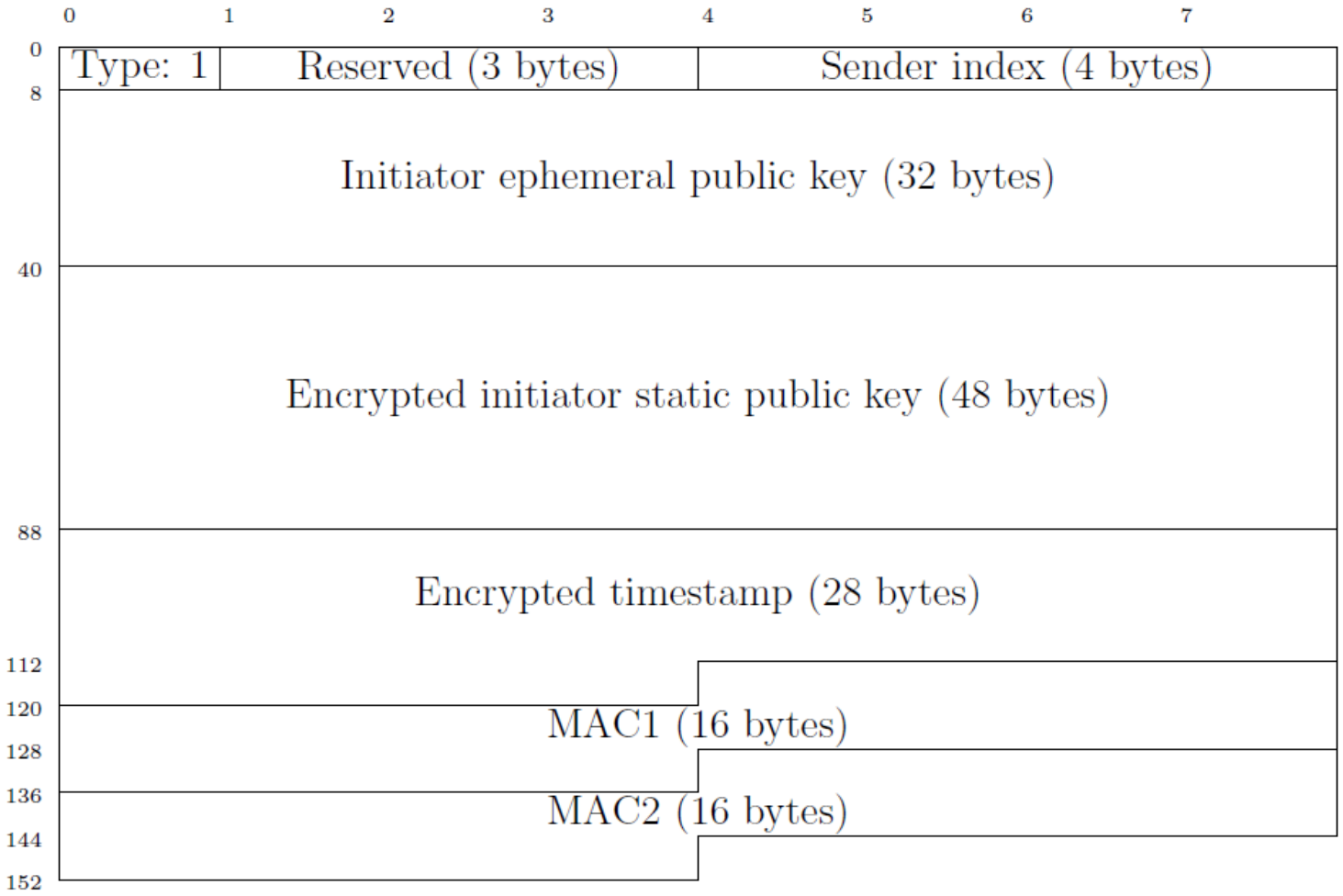
(a) Responder under load.

(b) Initiator under load.

출처: 참고문헌 4

2. WireGuard Protocol(3) – Message Type 1

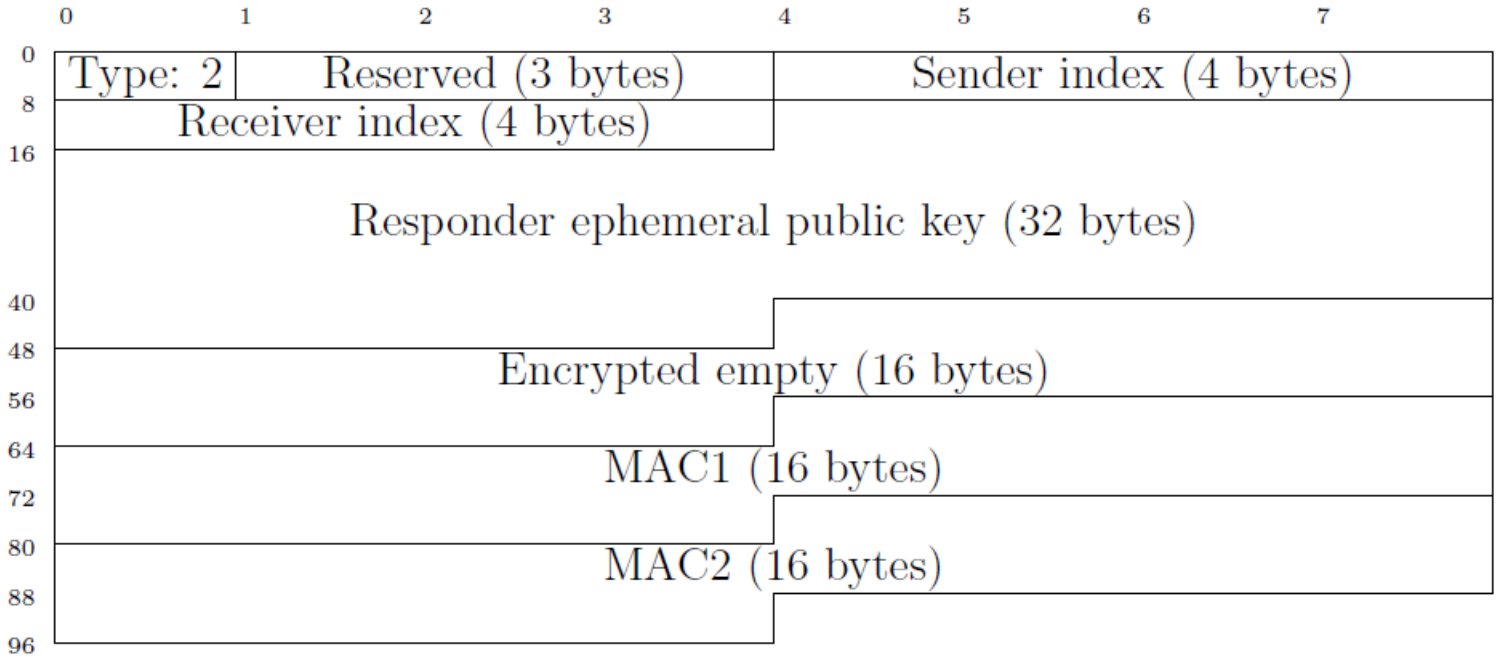
Initiation Message



출처: 참고문헌 4

2. WireGuard Protocol(3) - Message Type 2

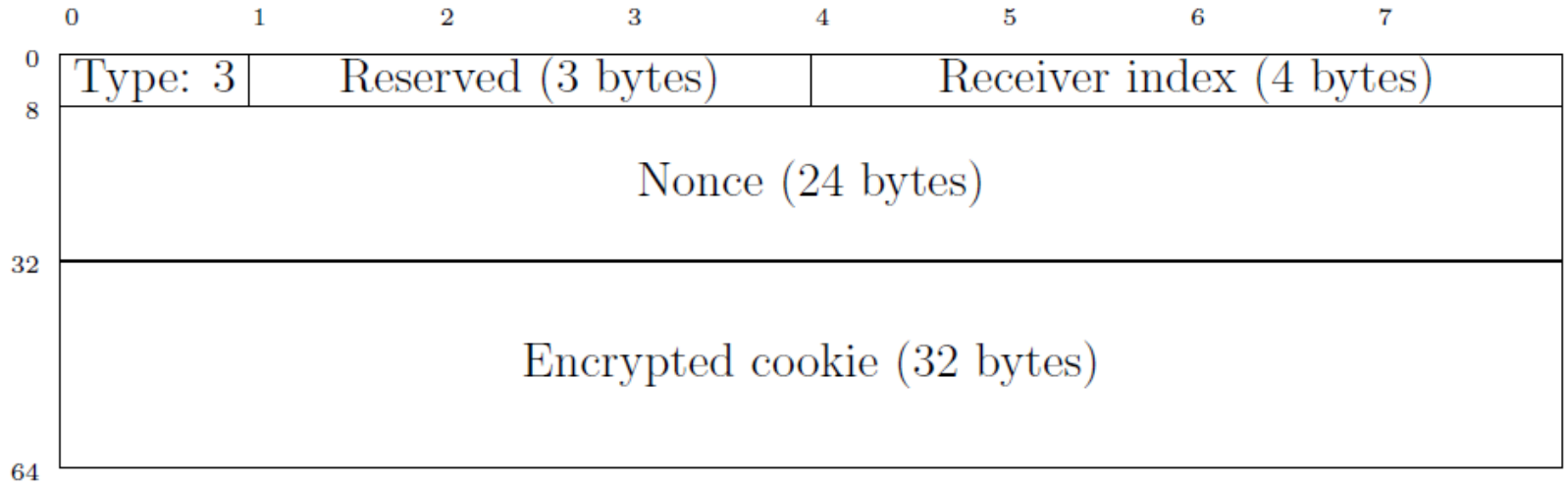
Response Message



출처: 참고문헌 4

2. WireGuard Protocol(3) – Message Type 3

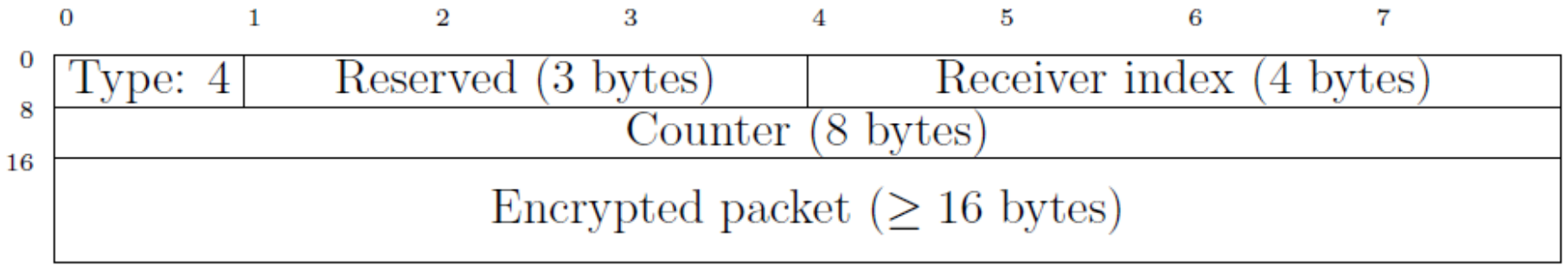
Cookie Message



출처: 참고문헌 4

2. WireGuard Protocol(3) – Message Type 4

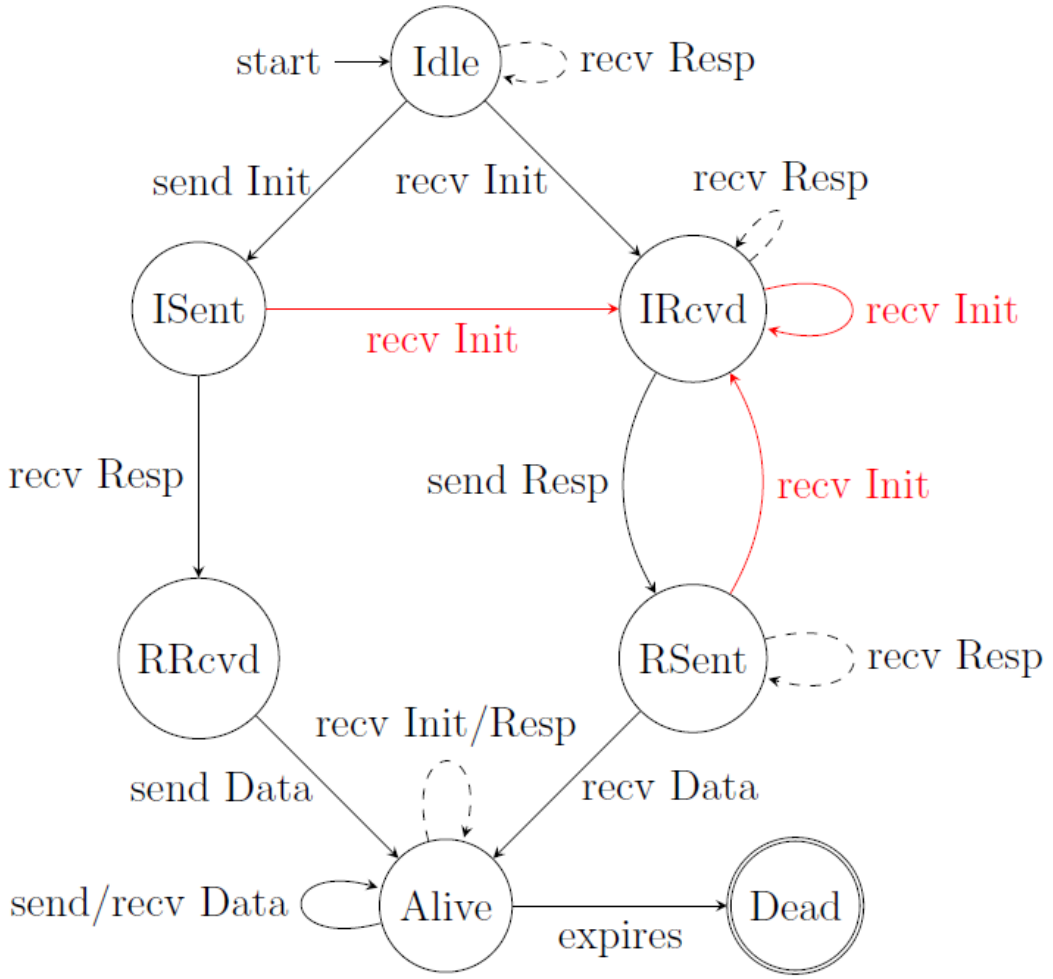
Data Message



출처: 참고문헌 4

2. WireGuard Protocol(4) – Timers & State Machine

Symbol	Value
Rekey-After-Messages	$2^{64} - 2^{16} - 1$ messages
Reject-After-Messages	$2^{64} - 2^{4} - 1$ messages
Rekey-After-Time	120 seconds
Reject-After-Time	180 seconds
Rekey-Attempt-Time	90 seconds
Rekey-Timeout	5 seconds
Keepalive-Timeout	10 seconds



출처: 참고문헌 4

2. WireGuard Protocol(5) – Handshake Summary(1)

1. Initial handshake message creation

1 :	**Initiator**	**Responder**
2 :	$\mathsf{pk}_I, \mathsf{sk}_I, \mathsf{tsp}$, initial data P	$\mathsf{pk}_R, \mathsf{sk}_R$
	..... Out-of-band key exchange: $\mathsf{pk}_I, \mathsf{pk}_R, Q$	
3 :	$\mathsf{epk}_I, \mathsf{esk}_I = \mathsf{DHGen}()$	
4 :	$\mathsf{ck}_1, k_1 = \mathsf{HKDF}_2(\mathsf{epk}_I, \mathsf{DH}(\mathsf{esk}_I, \mathsf{pk}_R))$	
5 :	$h_1 = H(\mathsf{pk}_R \| \mathsf{epk}_I)$	
6 :	$\text{enc-id} = \text{aead-enc}(k_1, 0, h_1, \mathsf{pk}_I)$	
7 :	$\mathsf{ck}_2, k_2 = \mathsf{HKDF}_2(\mathsf{ck}_1, \mathsf{DH}(\mathsf{sk}_I, \mathsf{pk}_R))$	
8 :	$h_2 = H(h_1 \| \text{enc-id})$	
9 :	$\text{enc-tsp} = \text{aead-enc}(k_2, 0, h_2, \mathsf{tsp})$	
10 :	$\mathsf{pkt} = \mathsf{epk}_I \| \text{enc-id} \| \text{enc-tsp}$	
11 :	$\text{mac1} = \text{BLAKE2s}(\mathsf{pk}_R, \mathsf{pkt})$	
12 :		$\xrightarrow[\text{Initiator packet}]{\mathsf{epk}_I, \text{enc-id}, \text{enc-tsp}, \text{mac1}}$

출처: 참고문헌 4

2. WireGuard Protocol(5) - Handshake Summary(2)

2. Initial handshake message processing

12 : $\mathsf{epk}_I, \text{enc-id}, \text{enc-tsp}, \text{mac1} \xrightarrow[\text{Initiator packet}]{}$

..... Responder receives initiator packet .....

13 : $\text{pkt'} = \mathsf{epk}_I || \text{enc-id} || \text{enc-tsp}$

14 : Verify $\mathsf{BLAKE2s}(\mathsf{pk}_R, \text{pkt'}, \text{mac1})$

15 : $\mathsf{ck}'_1, k'_1 = \mathsf{HKDF}_2(\mathsf{epk}_I, \mathsf{DH}(\mathsf{sk}_R, \mathsf{epk}_I))$

16 : $\mathsf{id} = \text{aead-dec}(k'_1, 0, h'_1, \text{enc-id})$

17 : $\mathsf{pk}_I, Q = \mathsf{LookupPeerKeys}(\mathsf{id})$

18 : $\mathsf{ck}'_2, k'_2 = \mathsf{HKDF}_2(\mathsf{ck}'_1, \mathsf{DH}(\mathsf{sk}_R, \mathsf{pk}_I))$

19 : $h'_2 = H(h'_1 || \text{enc-id})$

20 : $\text{tsp}' = \text{aead-dec}(\mathsf{k}'_2, 0, \mathsf{h}'_2, \text{enc-tsp})$

21 : $\mathsf{VerifyAntiReplay}(\text{tsp}')$

출처: 참고문헌 4

2. WireGuard Protocol(5) - Handshake Summary(3)

3. Response handshake message creation

22 : $\mathsf{epk}_R, \mathsf{esk}_R = \mathsf{DHGen}()$

23 : $\mathsf{ck}_3 = \mathsf{HKDF}_1(\mathsf{ck}'_2, \mathsf{epk}_R)$

24 : $\mathsf{ck}_4 = \mathsf{HKDF}_1(\mathsf{ck}_3, \mathsf{DH}(\mathsf{esk}_R, \mathsf{epk}_I))$

25 : $\mathsf{ck}_5 = \mathsf{HKDF}_1(\mathsf{ck}_4, \mathsf{DH}(\mathsf{esk}_R, \mathsf{pk}_I))$

26 : $\mathsf{ck}_6, t, k_3 = \mathsf{HKDF}_3(\mathsf{ck}_5, Q)$

27 : $h_3 = H(h'_2||t)$

28 : $\text{enc-e} = \text{aead-enc}(k_3, 0, h_3, \epsilon)$

29 : $\mathsf{mac1}_r = \mathsf{BLAKE2s}(\mathsf{pk}_I, \mathsf{epk}_R||\text{enc-e})$

30 : $\xleftarrow[\text{Responder packet}]{\mathsf{epk}_R, \text{enc-e}, \mathsf{mac1_r}}$

출처: 참고문헌 4

2. WireGuard Protocol(5) - Handshake Summary(4)

4. Response handshake message processing

30 : $\xleftarrow[\text{Responder packet}]{\mathsf{epk}_R, \text{enc-e}, \mathsf{mac1_r}}$

..... Initiator receives responder packet .....

31 : Verify $\mathsf{BLAKE2s}(\mathsf{pk}_I, \mathsf{epk}_R || \text{enc-e}, \mathsf{mac1}_r)$

32 : $\mathsf{ck}'_3 = \mathsf{HKDF}_1(\mathsf{ck}_2, \mathsf{epk}_R)$

33 : $\mathsf{ck}'_4 = \mathsf{HKDF}_1(\mathsf{ck}'_3, \mathsf{DH}(\mathsf{esk}_I, \mathsf{epk}_R))$

34 : $\mathsf{ck}'_5 = \mathsf{HKDF}_1(\mathsf{ck}'_4, \mathsf{DH}(\mathsf{sk}_I, \mathsf{epk}_R))$

35 : $\mathsf{ck}'_6, t', k'_3 = \mathsf{HKDF}_3(\mathsf{ck}'_5, Q)$

36 : $h'_3 = H(h_2 || t')$

37 : Verify $\text{aead-dec}(k'_3, 0, h'_3, \text{enc-e}) = \epsilon$

출처: 참고문헌 4

2. WireGuard Protocol(5) - Handshake Summary(5)

5. Handshake confirmation

38 : $T_i^{send}, T_i^{recv} = \mathsf{HKDF}_2(\mathsf{ck}'_6, \epsilon)$

39 : $\mathsf{ctr}_i = 0$

40 : $\mathsf{enc\text{-}P} = \mathsf{aead\text{-}enc}(T_i^{send}, \mathsf{ctr}_i, \epsilon, \mathsf{P})$

41 : $\mathsf{ctr}_i,\ \mathsf{enc\text{-}P} \xrightarrow[\text{Initial data packet}]{}$

..... Responder receives confirmation .....

42 : $T_r^{recv}, T_r^{send} = \mathsf{HKDF}_2(\mathsf{ck}_6, \epsilon)$

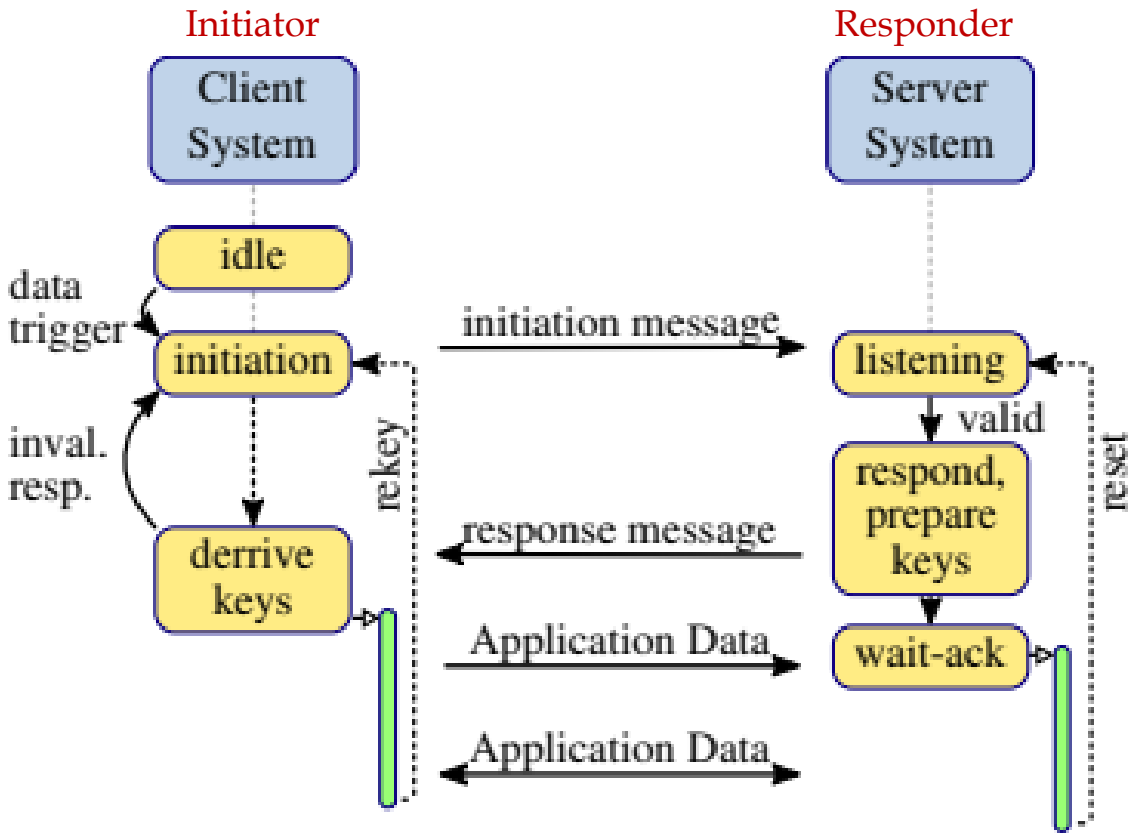
43 : $\mathsf{P'} = \mathsf{aead\text{-}dec}(T_r^{recv}, \mathsf{ctr}_i, \epsilon, \mathsf{enc\text{-}P})$

..... Session established .....

44 :

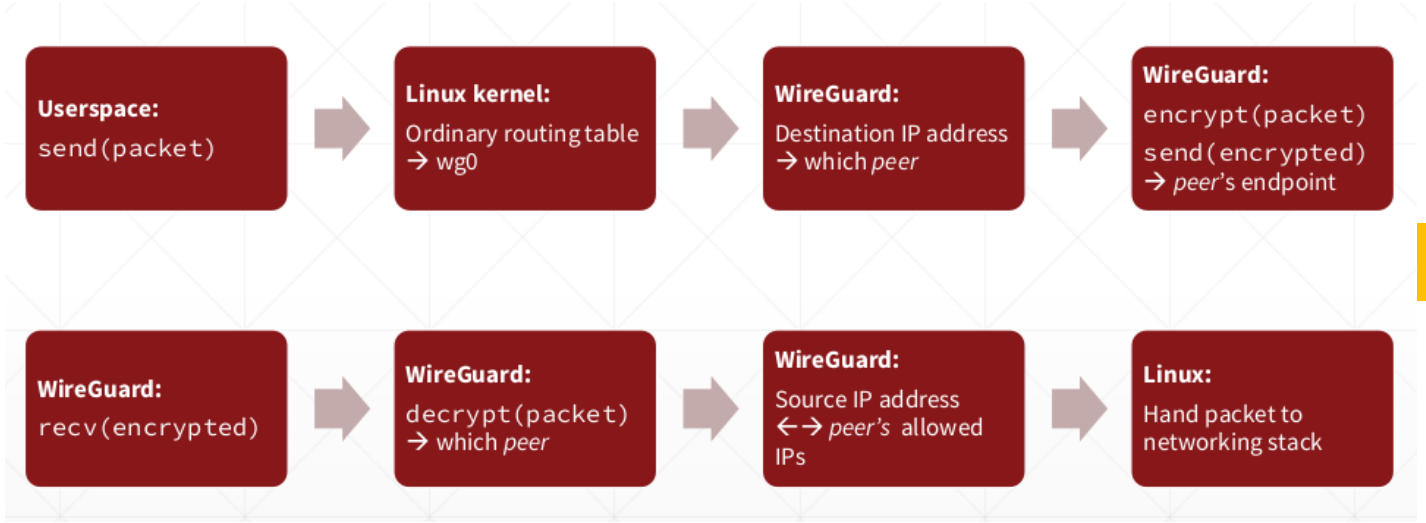
출처: 참고문헌 4

3. WireGuard Kernel Code 분석(1)



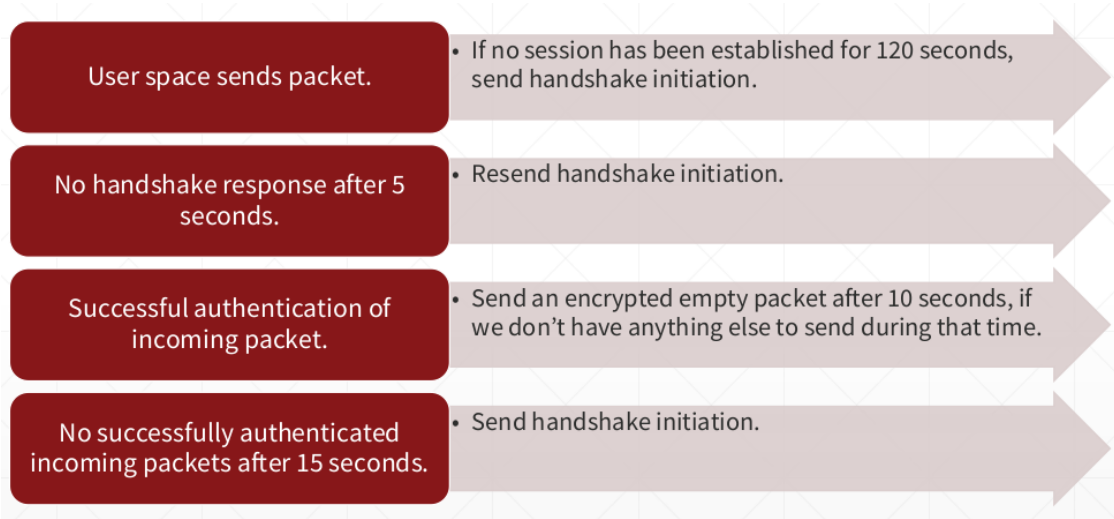
출처: 참고문헌 6

3. WireGuard Kernel Code 분석(2)



Packet send/recv

Timers



출처: 참고문헌 3

3. WireGuard Kernel Code 분석(3) – Handshake Tx Flow

```
<handshake tx>
---------------
wg_open() | wg_xmit() | set_peer() | wg_packet_consume_data_done() | wg_packet_send_keepalive()
|
V
wg_packet_send_staged_packets()
|
V
wg_packet_send_queued_handshake_initiation()
|
V
queue_work(peer->device->handshake_send_wq, &peer->transmit_handshake_work)
        => [wg_packet_handshake_send_worker]

[wg_packet_handshake_send_worker]
        -> Initiation packet 만들어 전송
        |
        V
        wg_packet_send_handshake_initiation()
        |
        V
        wg_socket_send_buffer_to_peer()
        |
        V
        send4() or send6()
        |
        V
        peer에게로 packet 전송
```

3. WireGuard Kernel Code 분석(4) – Handshake Rx Flow

```
<handshake rx>
---------------
tcp/ip stack udp tunnel packet recv
|
V
wg_receive()
|
V
wg_packet_receive()
        : MESSAGE_HANDSHAKE_INITIATION/MESSAGE_HANDSHAKE_RESPONSE/MESSAGE_HANDSHAKE_COOKIE
|
V
skb_queue_tail()
|
V
[wg_packet_handshake_receive_worker]
        |
        V
        skb_dequeue()
           : MESSAGE_HANDSHAKE_INITIATION
                |
                V
                wg_noise_handshake_consume_initiation()
                wg_packet_send_handshake_response() -> wg_socket_send_buffer_to_peer() -> send4() or send6()

           : MESSAGE_HANDSHAKE_RESPONSE
                |
                V
                wg_noise_handshake_consume_response()
```

3. WireGuard Kernel Code 분석(5) – Data Tx Flow

```
<packet tx flow>
------------------
        -> ip encapsulation(tunneling) & encrpytion

wg_xmit()
|
V
enqueue to ptr_ring buffer
|
V
[wg_packet_encrypt_worker]
        |
        V
        dequeue ptr_ring buffer
        |
        V
        encrypt_packet()
        |
        V
        [wg_packet_tx_worker]
                |
                V
                dequeue ptr_ring buffer
                |
                V
                send4() or send6()
                |
                V
                peer에게로 packet 전송
```

3. WireGuard Kernel Code 분석(6) – Data Rx Flow

```
<packet rx flow>
----------------
        -> ip decapsulation & decryption

tcp/ip stack udp tunnel packet recv
|
V
wg_receive()
|
V
wg_packet_receive()
    : MESSAGE_HANDSHAKE_INITIATION/MESSAGE_HANDSHAKE_RESPONSE/MESSAGE_HANDSHAKE_COOKIE
        => skb_queue_tail() -> wg_packet_handshake_receive_worker

    : MESSAGE_DATA
        |
        V
        enqueue to ptr_ring buffer(produce)
        |
        V
        [wg_packet_decrypt_worker]
                |
                V
                dequeue from ptr_ring buffer(consume)
                |
                V
                decrypt_packet()
                |
                V
                napi_schedule()
                |
                V
                tcp/ip stack에서 이후 처리
```

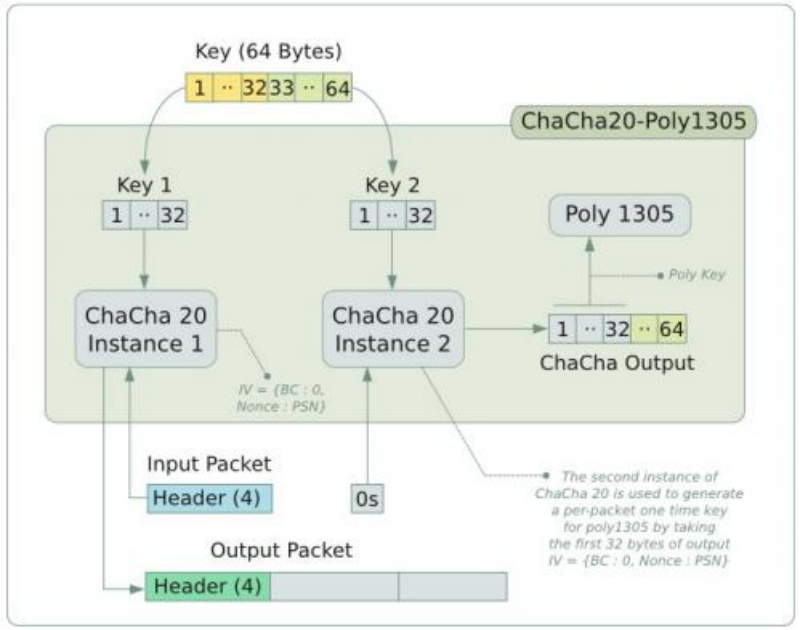
4. References

- [1] https://www.wireguard.com/
- [2] https://www.wireguard.com/papers/wireguard.pdf
- [3] WireGuard - Fast, Modern, Secure VPN Tunnel, Jason A. Donenfeld
- [4] Master's Thesis - Analysis of the WireGuard protocol, Peter Wu
- [5] Tiny WireGuard Tweak, Jacob Appelbaum, Chloe Martindale, and Peter Wu
- [6] In Search for a Simple Secure Protocol for Safety-Critical High-Assurance Applications, Thorsten Schulz, …
- [7] https://slowbootkernelhacks.blogspot.com/2020/09/wireguard-vpn.html
- [8] and, Google~

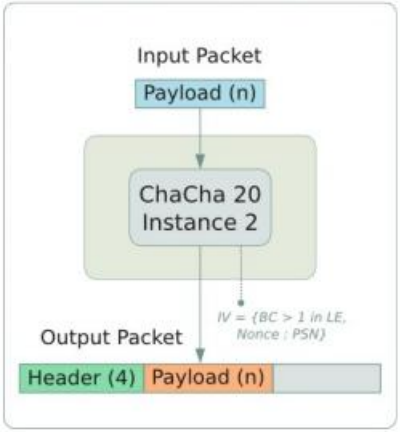
Appendix

A. WireGuard Crypto(1) – ChaCha20/Poly1305

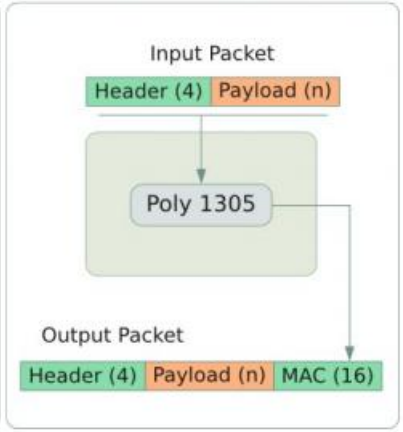
1. Encrypting the Header and Initializing Poly1305



2. Encrypting the Payload



3. Computing the MAC



ChaCha 20 – Poly 1305

인증을 동반한 암호 알고리즘

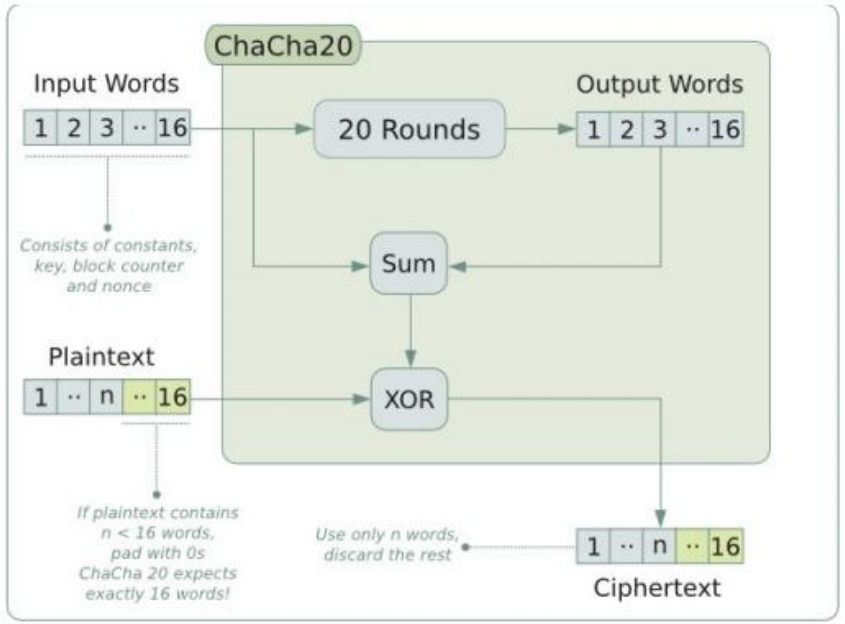
Introduction

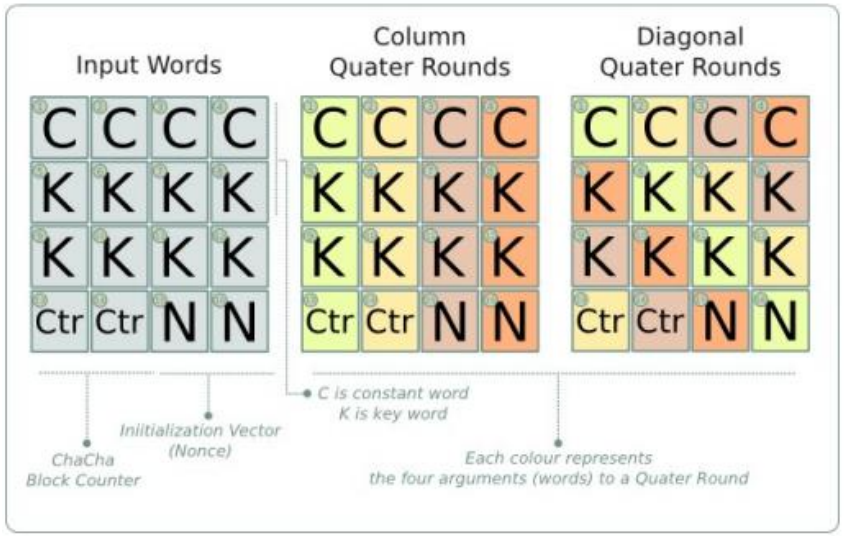
- ChaCha20-Poly1305 is an Authenticated Encryption mechanism which combines two primitives:
  - ChaCha20 for Encryption
  - Poly1305 for Authentication
- ChaCha20-Poly1305 uses a 64 byte symmetric key
- An input packet consists of a 4 byte Header encoding the length of the packet, as well as a variable length payload (and a 16 byte MAC if decrypting)
  - Packet Length = Length of Header + Length of Payload + Length of Message Authentication Code (MAC)

Operation

- A first instance of ChaCha20 is used to encrypt the Header using the first 32 bytes of the key and an Initialization Vector as follows :
  - Block Counter (BC) = 0s
  - Nonce = Packet Sequence Number (PSN)
- A second instance is used to generate a key for Poly1305 by using the last 32 bytes of the key, 0s as input and keeping the first 32 bytes of output
- This second instance is then used with BC = 1 in Little Endian (LE) and Nonce = PSN to encrypt the n-byte payload
- ChaCha20 will increment the BC internally but the ChaCha20-Poly1305 implementation should manage the Nonce (i.e Increment the PSN for every packet)
- Finally Poly1305 is used on the **encrypted** header **and** payload and a MAC is calculated and appended to the output packet
- For decryption, decrypt the header to get the length then verify the MAC and decrypt the rest of the packet (using the same procedure as encryption) only if the MAC is valid

A. WireGuard Crypto(2) – ChaCha20





ChaCha 20

스트림 암호 알고리즘

Introduction

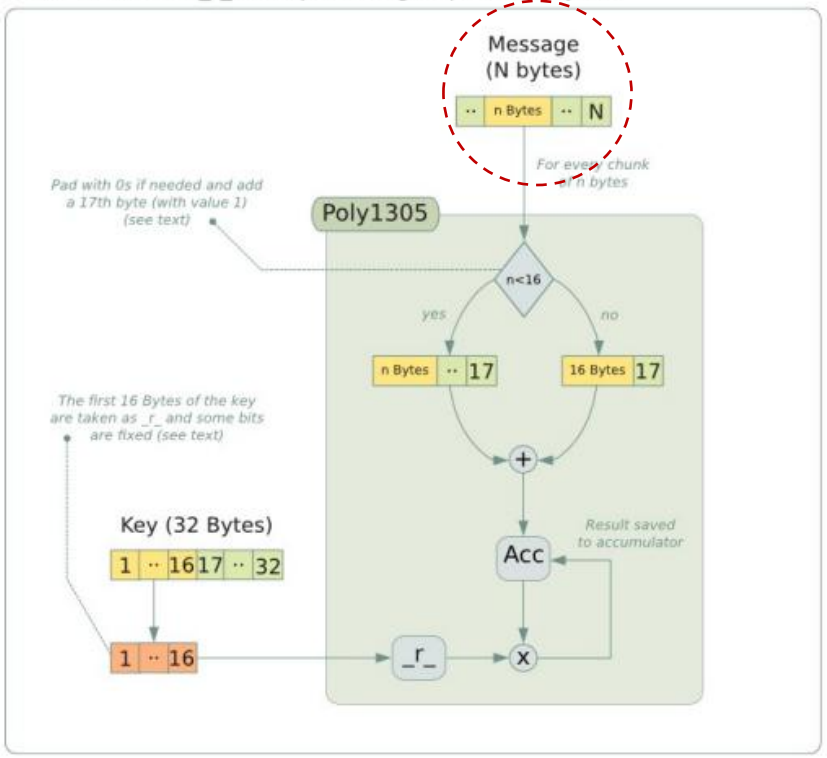
- ChaCha 20 is a stream cipher developed by Daniel Bernstein.
- It is a refinement of the Salsa 20 cipher.
- ChaCha 20 works on 4 byte words, takes 16 words of plaintext and outputs 16 words of ciphertext.
- Operations detailed below are repeated for every 16 words (64 bytes) of a packet

Operation

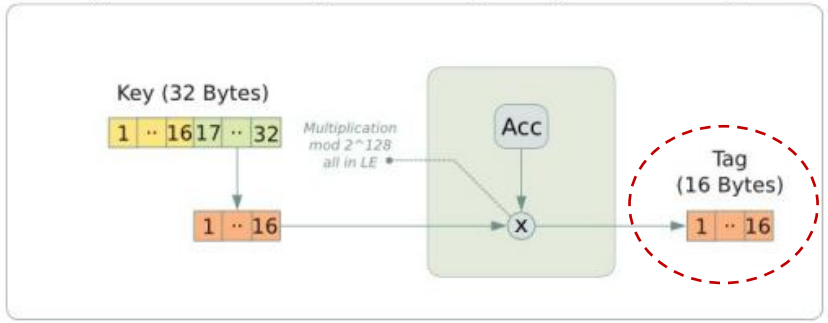
- A 16 word Input Matrix is formed as follows :
  - 4 words of constant value
  - 8 words of key
  - 2 words of block counter (which is incremented by ChaCha 20 after each 20 rounds)
  - 2 words of nonce (which should be managed outside of ChaCha 20)
- For every new 16 words of plaintext, 20 rounds are performed on the original Input Matrix by alternating Column and Diagonal rounds.
- Each round performs 4 quarter rounds on 4 words as follows:
  - a += b; d ^= a; d <<<= 16;
  - c += d; b ^= c; b <<<= 12;
  - a += b; d ^= a; d <<<= 8;
  - c += d; b ^= c; b <<<= 7;
- The output of the 20 rounds is summed to the original input matrix
- This is written out in little endian form and XORed to the 16 words of plaintext to produce 16 words of ciphertext.
- Decryption uses the same procedure as encryption

A. WireGuard Crypto(3) – Poly1305

1. First, initialize _r_ then process groups of 16 bytes



2. Finally, add the last 16 bytes of the key and generate the tag



Poly 1305

인증 알고리즘

Introduction

- Poly1305 is a Wegman-Carter, one-time authenticator designed by D. J. Bernstein
- It is used to calculate a Message Authentication Code (MAC) for a message
- Poly 1305 uses a 32 Byte key and operates on an N byte message

Operation

- The first 16 bytes of the one-time key are interpreted as a number _r_ with the following modifications:
  - The top 4 bits of bytes 3, 7, 11, 15 are set to 0
  - The bottom 2 bits of bytes 4, 8, 12 are set to 0
  - The 16 bytes are interpreted as a little endian value
- The accumulator (Acc in the diagram) is set to 0
- For every n bytes read from the N byte message, if n = 16 then just add a 17th byte having a value of 1 and the 17 bytes are treated as a little endian number
- If n < 16 then pad with 0s until there are 16 bytes and add the 17th byte as in the case when n = 16
- The number is then added to the accumulator which is multiplied by _r_ and the result is saved back to the accumulator
- Note : These operations are all mod 2^130 – 5
- Finally, the last 16 bytes of the key are interpreted as a little endian number and this number is added to the accumulator mod 2^128
- The result is then written out as a little endian number and this is taken as the 16 byte tag

We Secure the Internet of Things with vIoTSec !

Thank You