nQUIC: Noise-Based Packet Protection

Mathias Hall-Andersen[*]
University of Copenhagen

David Wong[*]
Facebook

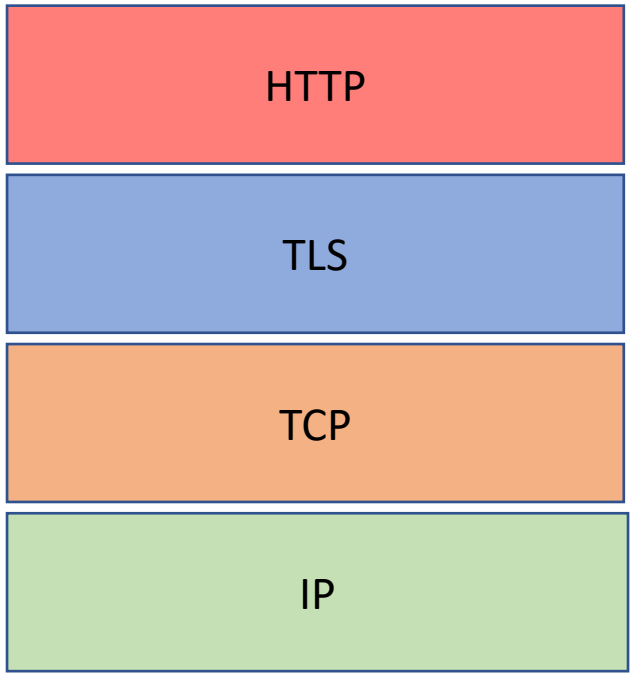
Nick Sullivan
Cloudflare

Alishah Chator[†]
Johns Hopkins University

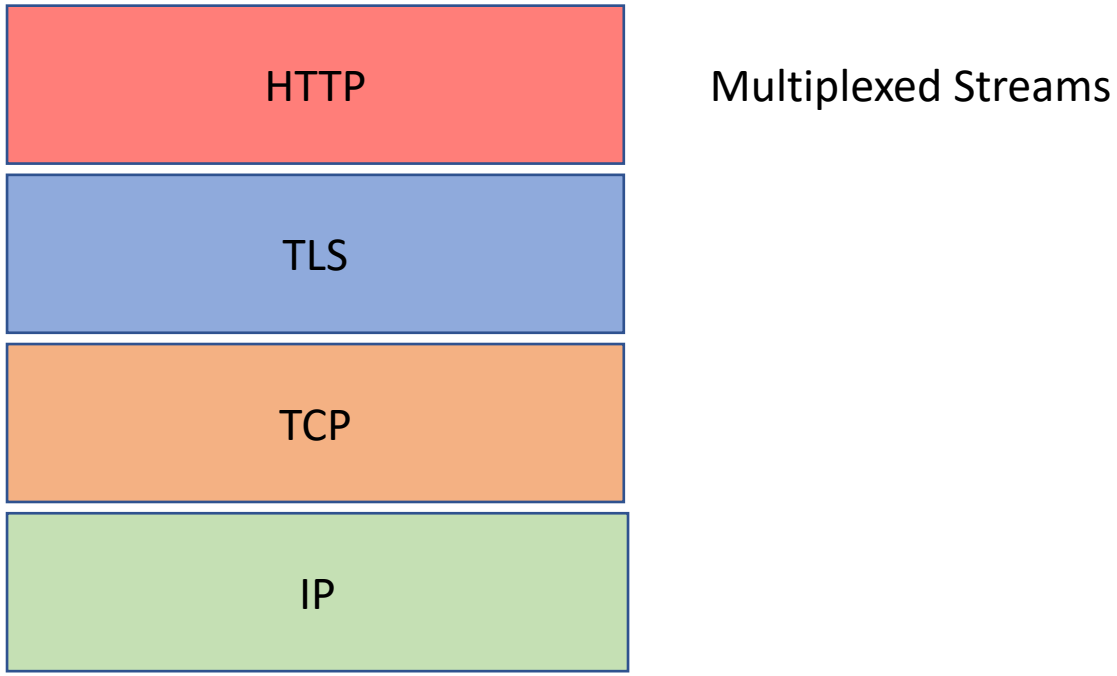
[*]Work done while at NCC Group

[†]Work done while at Cloudflare

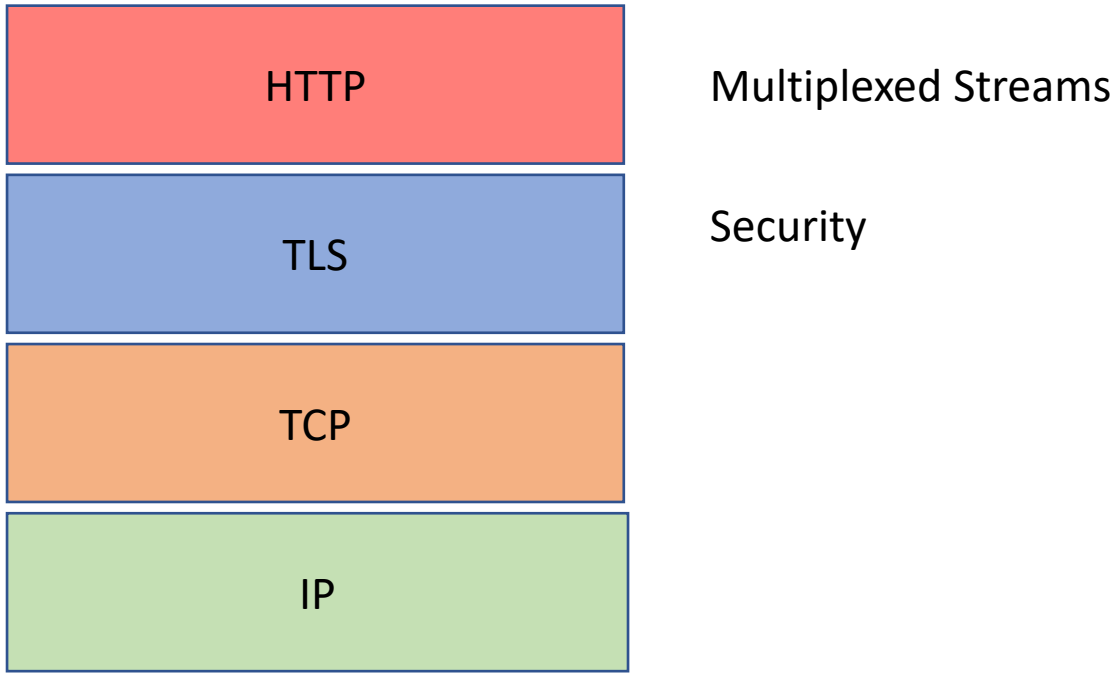
The Traditional HTTPS Stack



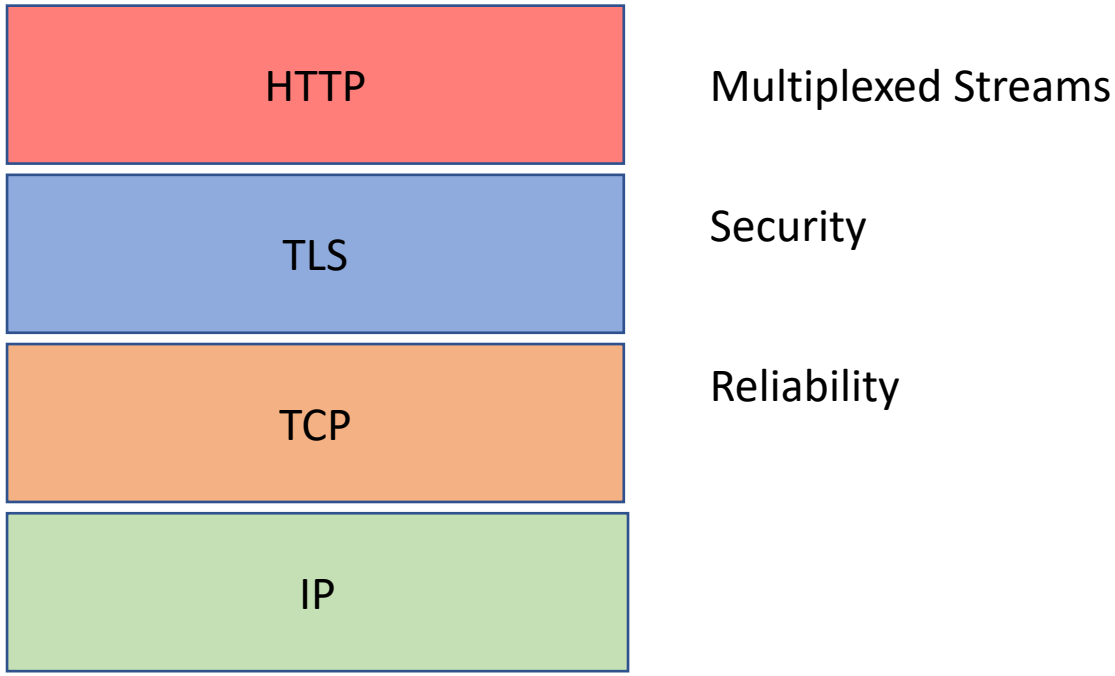
The Traditional HTTPS Stack



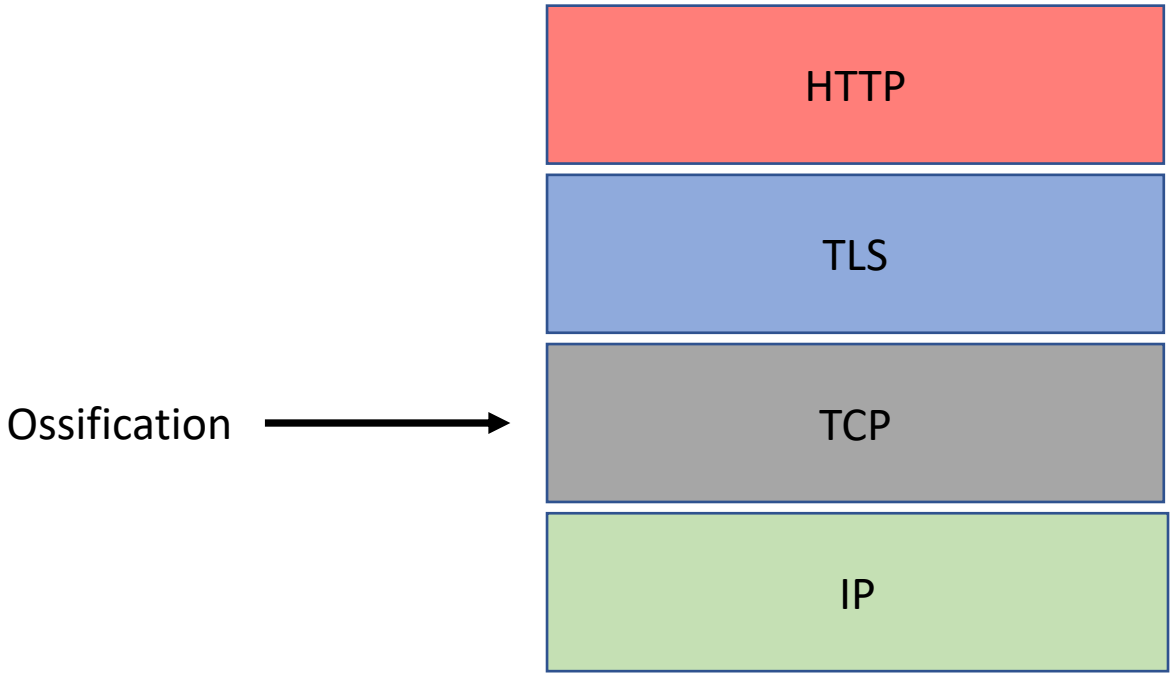
The Traditional HTTPS Stack



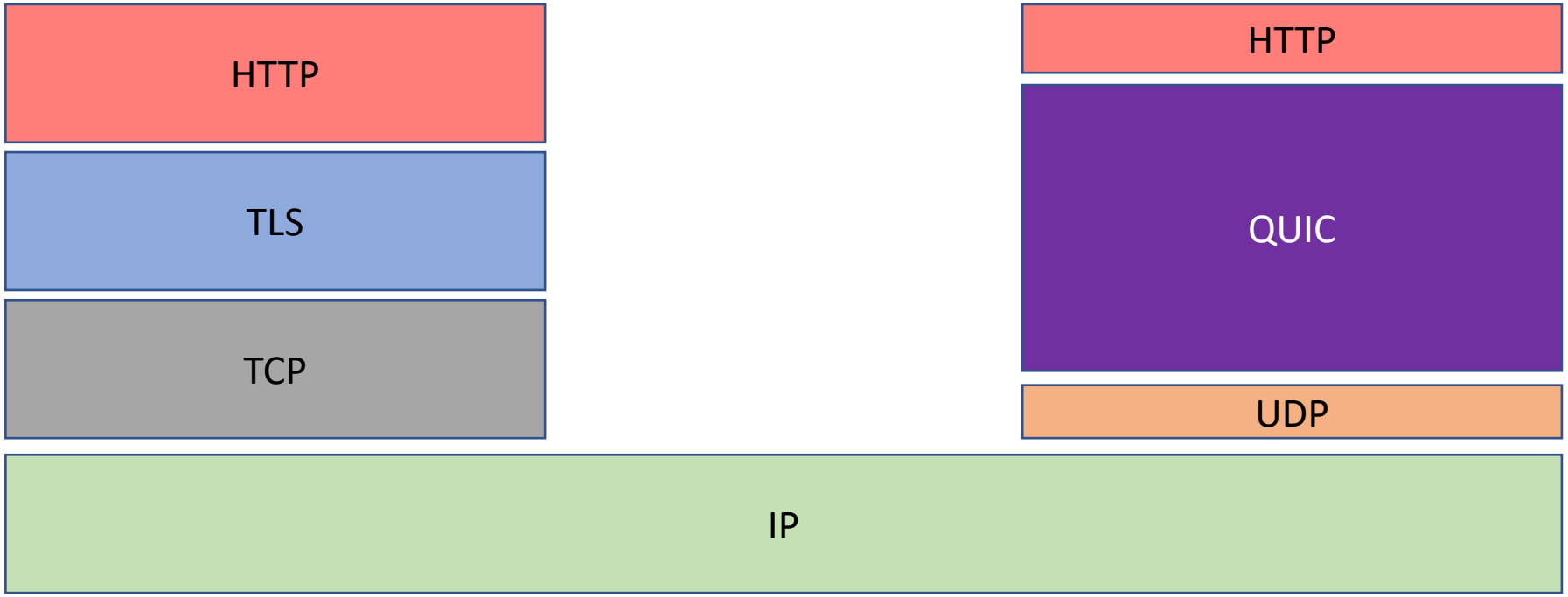
The Traditional HTTPS Stack



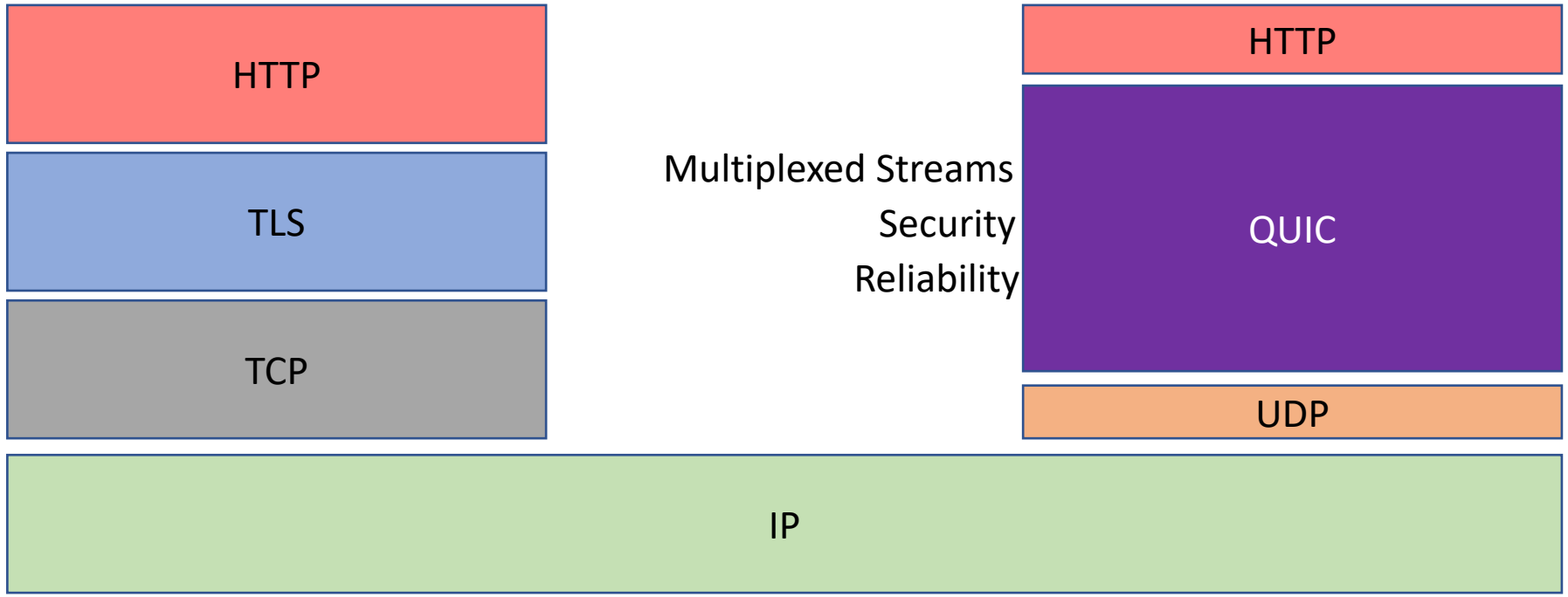
The Traditional HTTPS Stack



The QUIC way of doing things

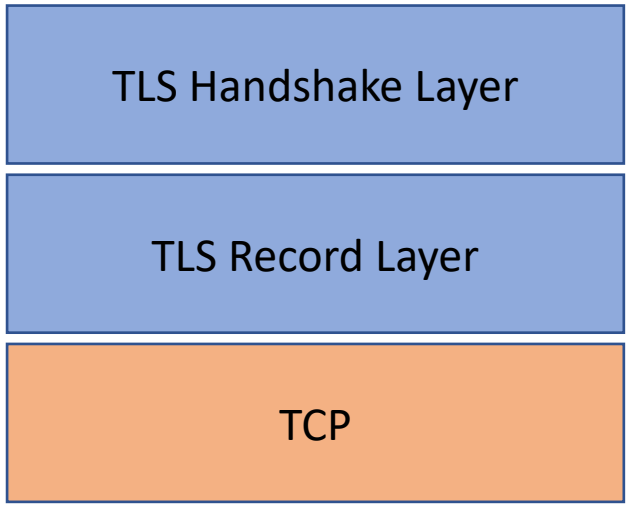


The QUIC way of doing things

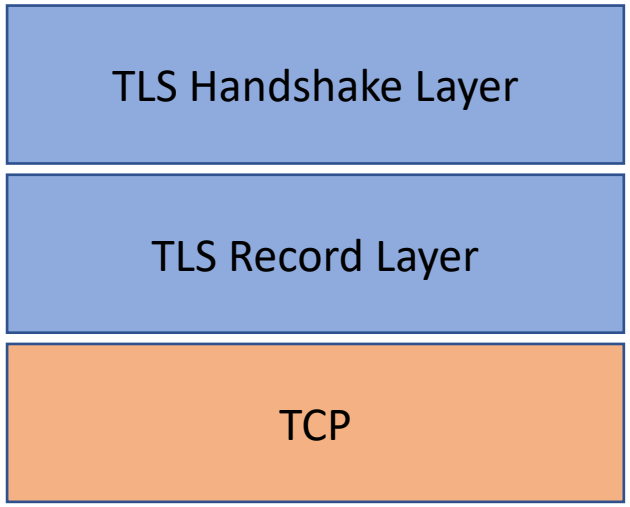


Securing Protocols with TLS

Securing Protocols with TLS: TCP

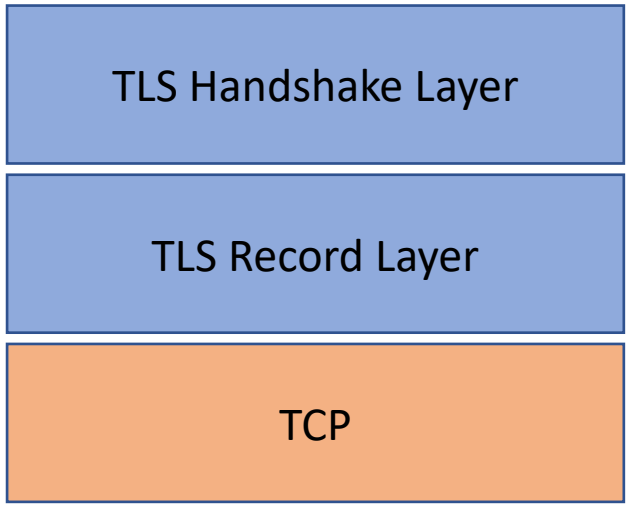


Securing Protocols with TLS: TCP



Handshake messages, Application data, TLS alerts

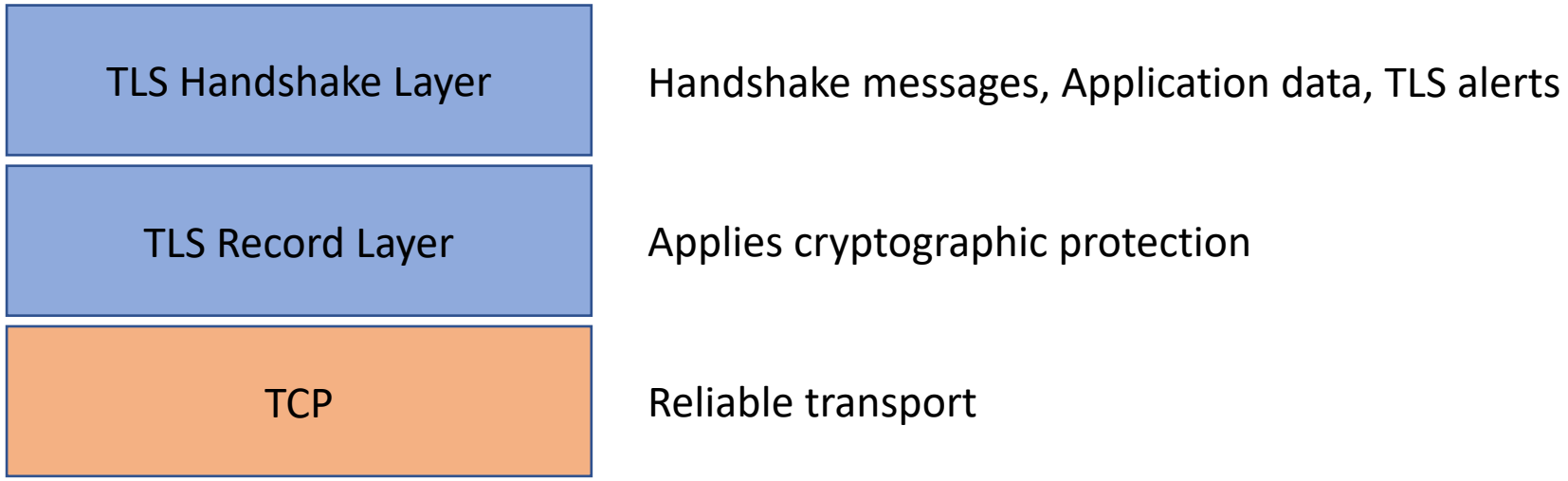
Securing Protocols with TLS: TCP



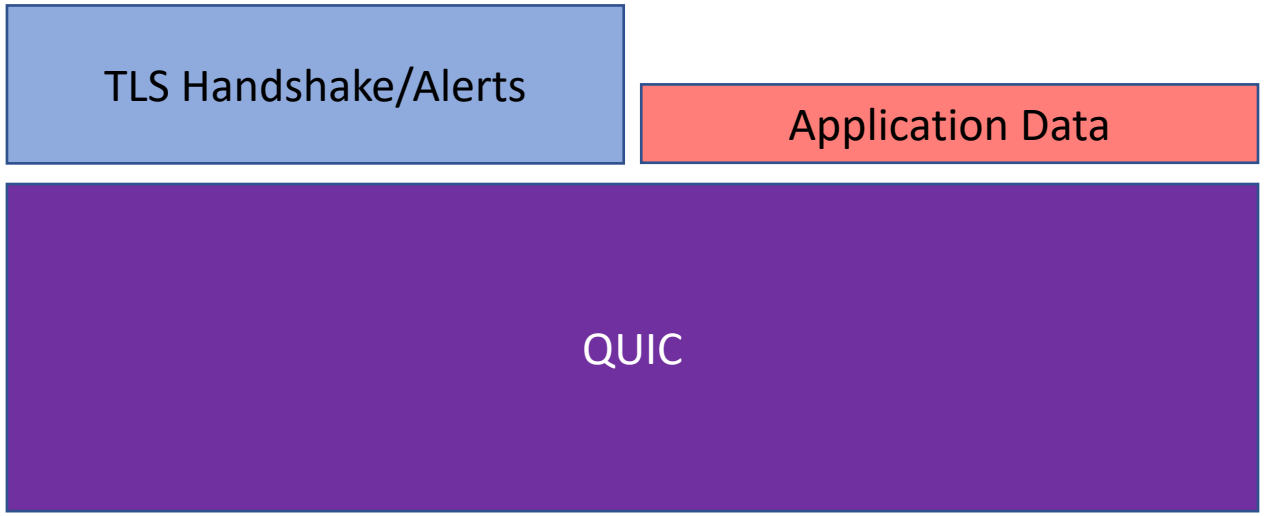
Handshake messages, Application data, TLS alerts

Applies cryptographic protection

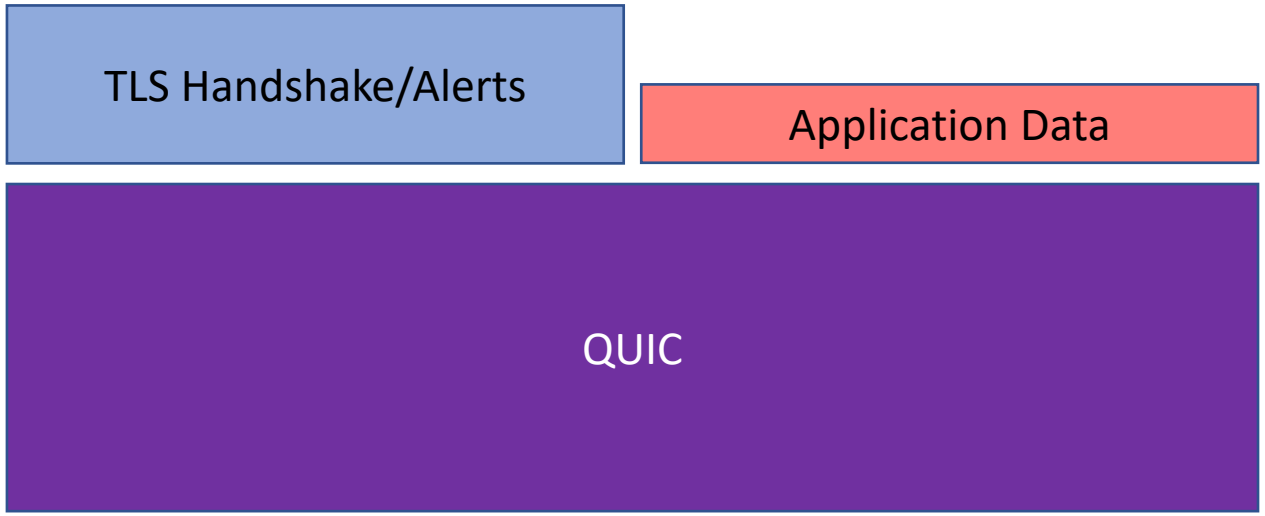
Securing Protocols with TLS: TCP



Securing Protocols with TLS: QUIC



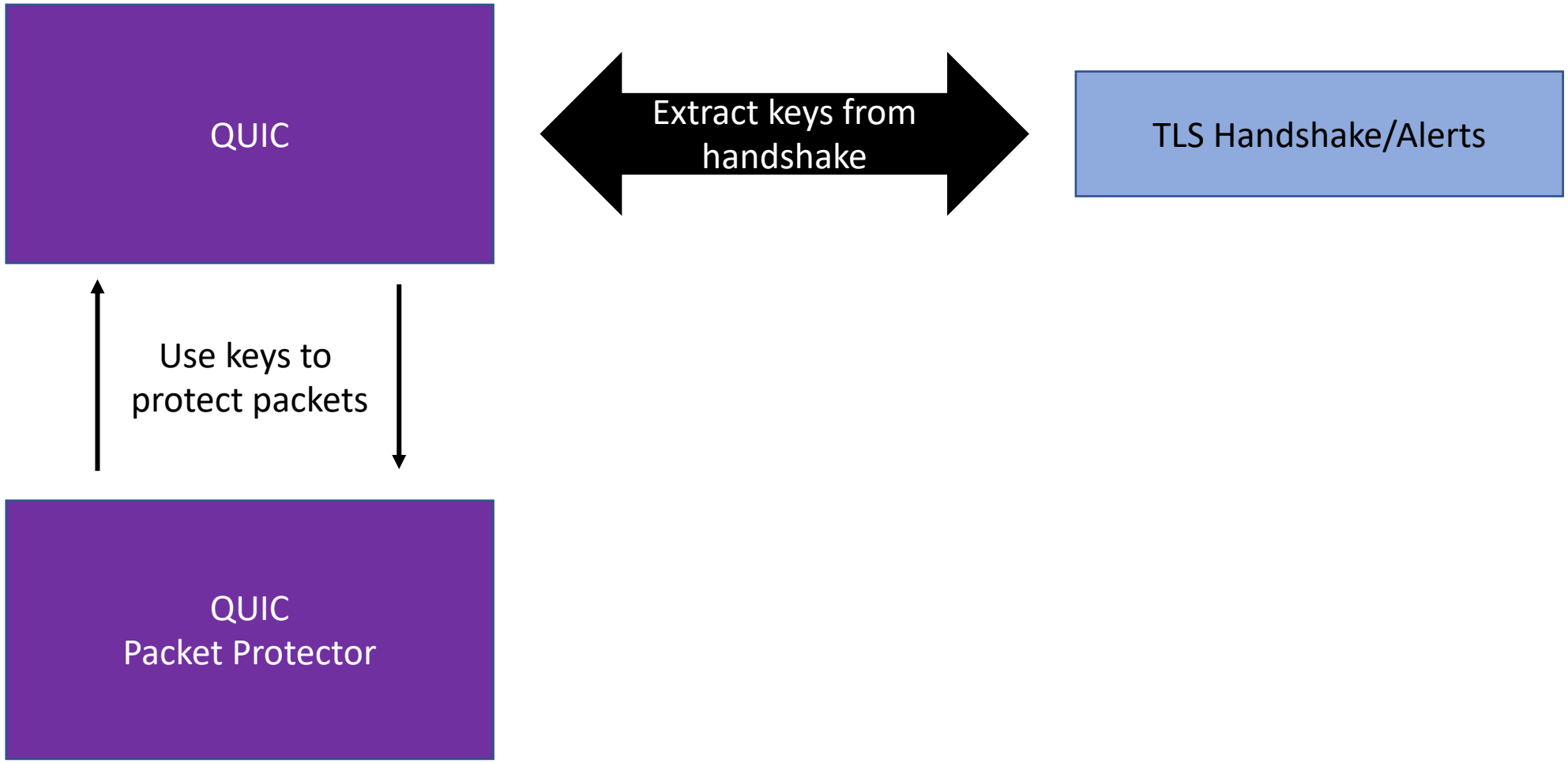
Securing Protocols with TLS: QUIC



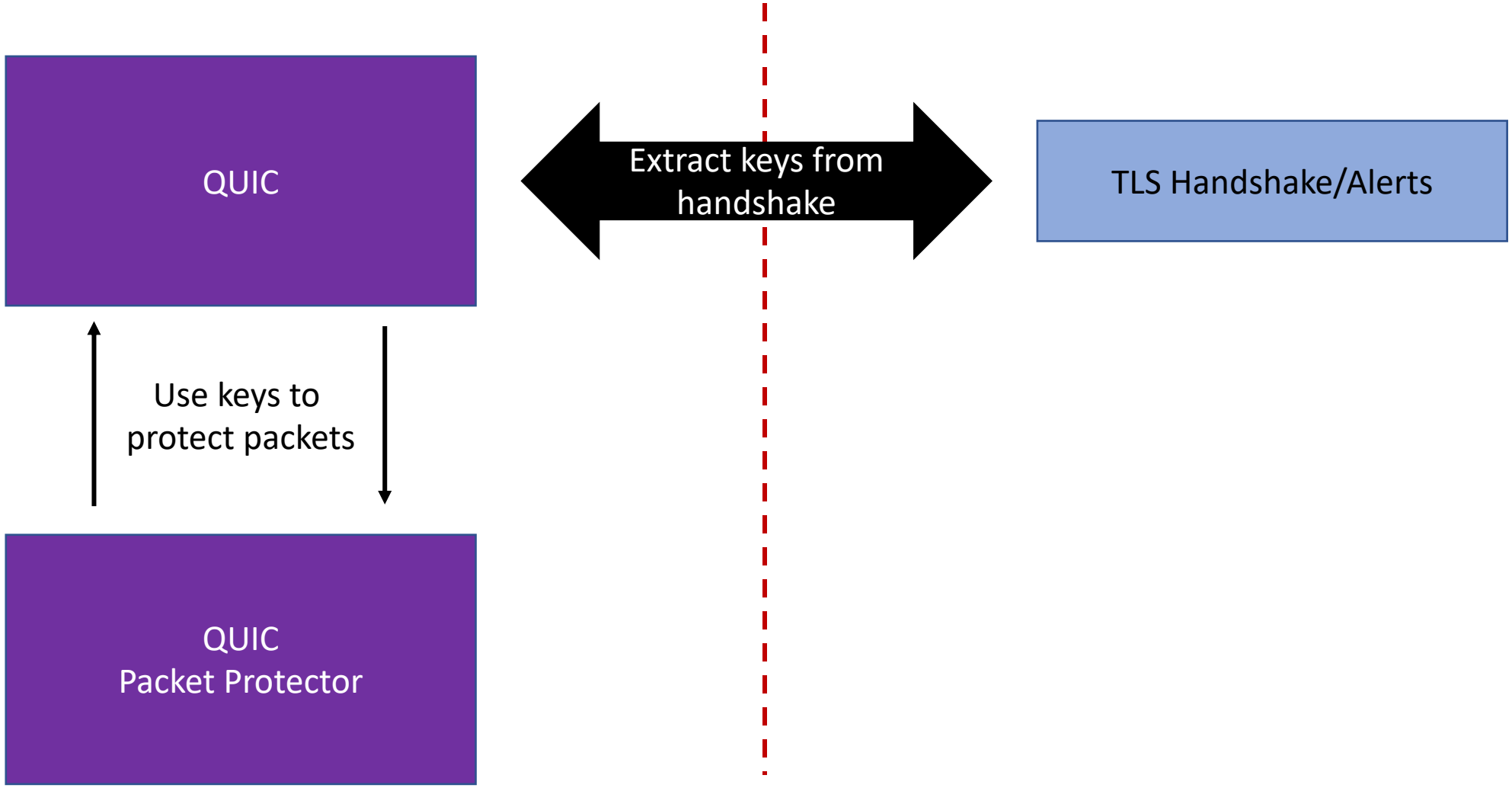
Supplies both packet protection and reliable transport

Handshake Modularity

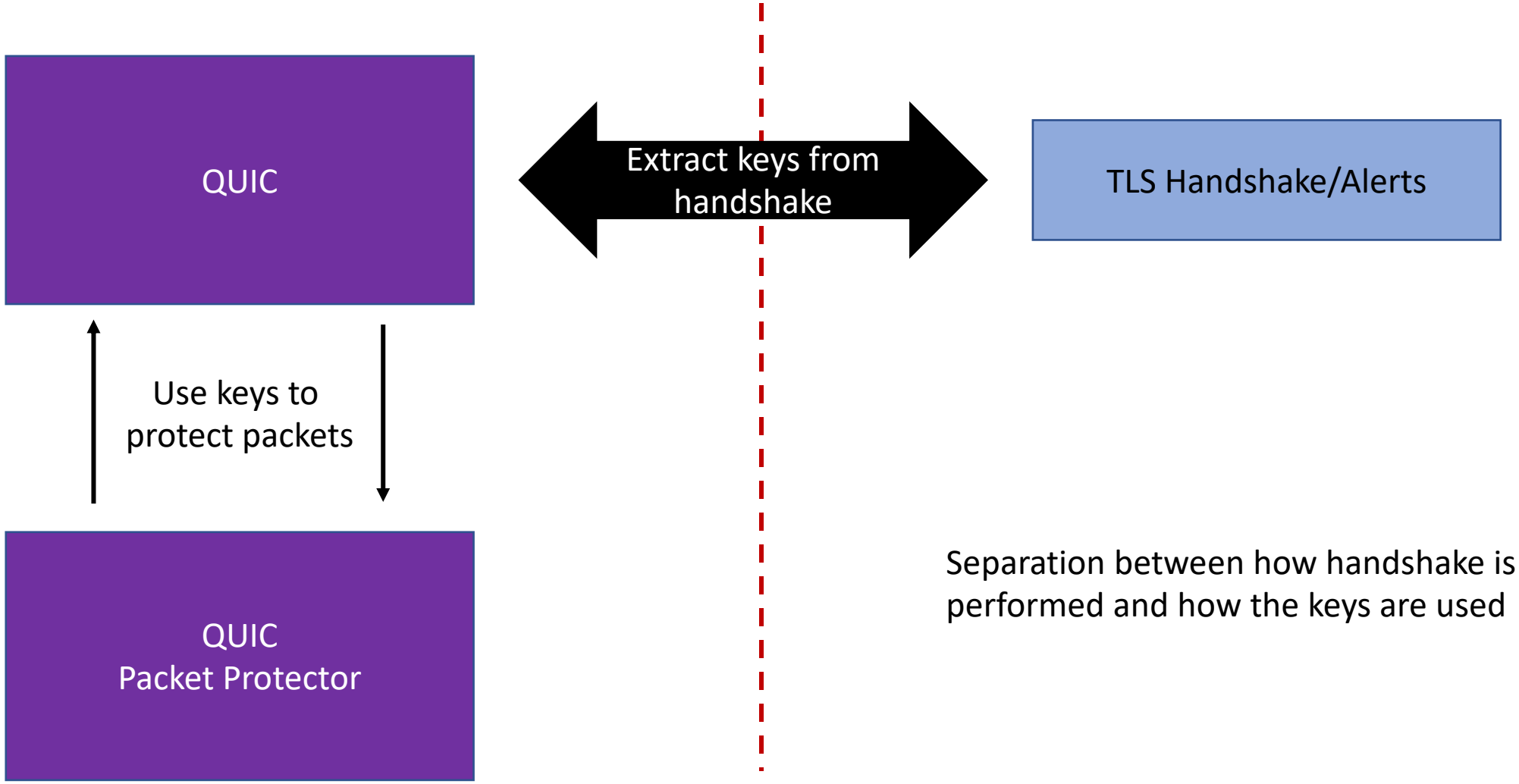
Handshake Modularity



Handshake Modularity

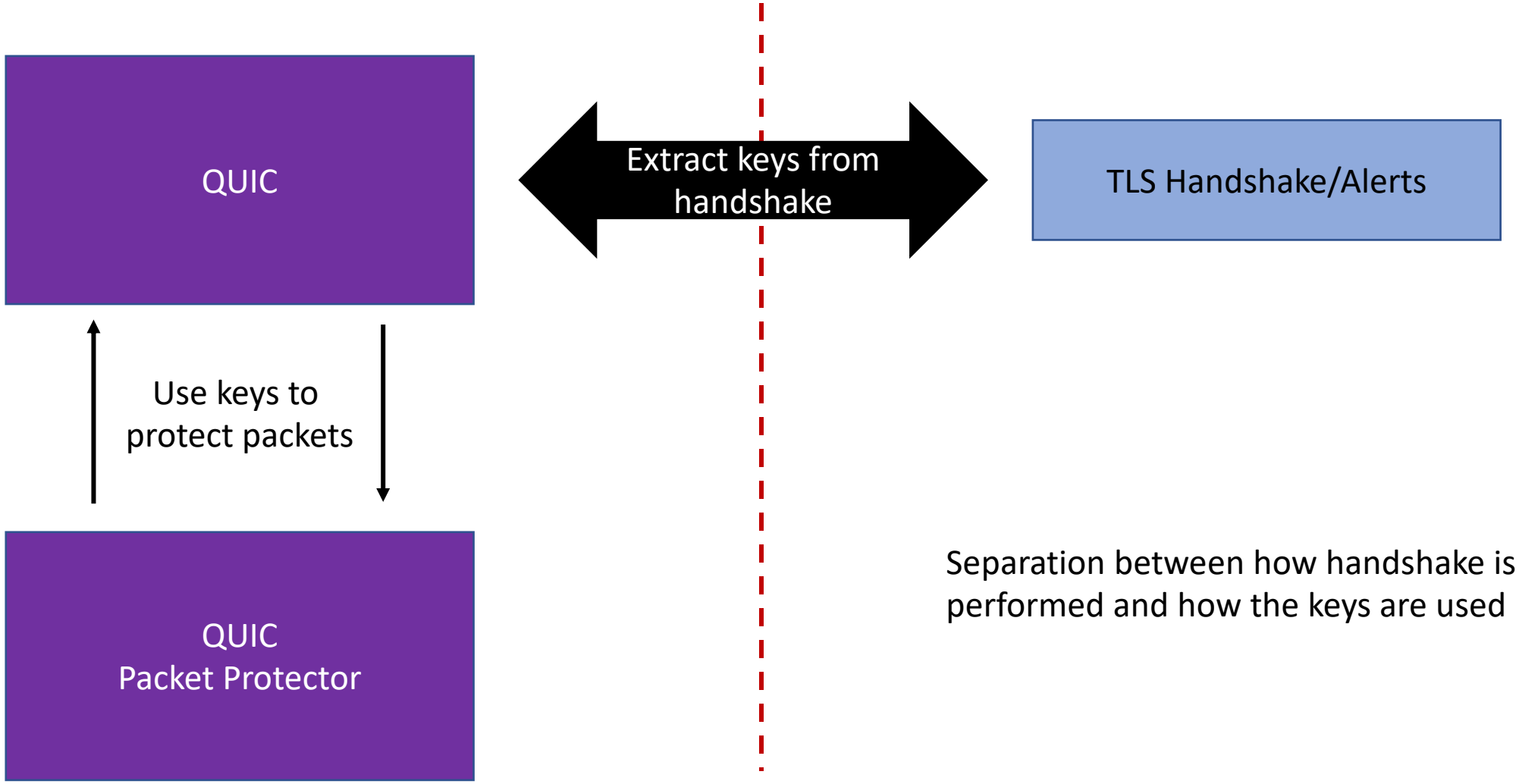


Handshake Modularity



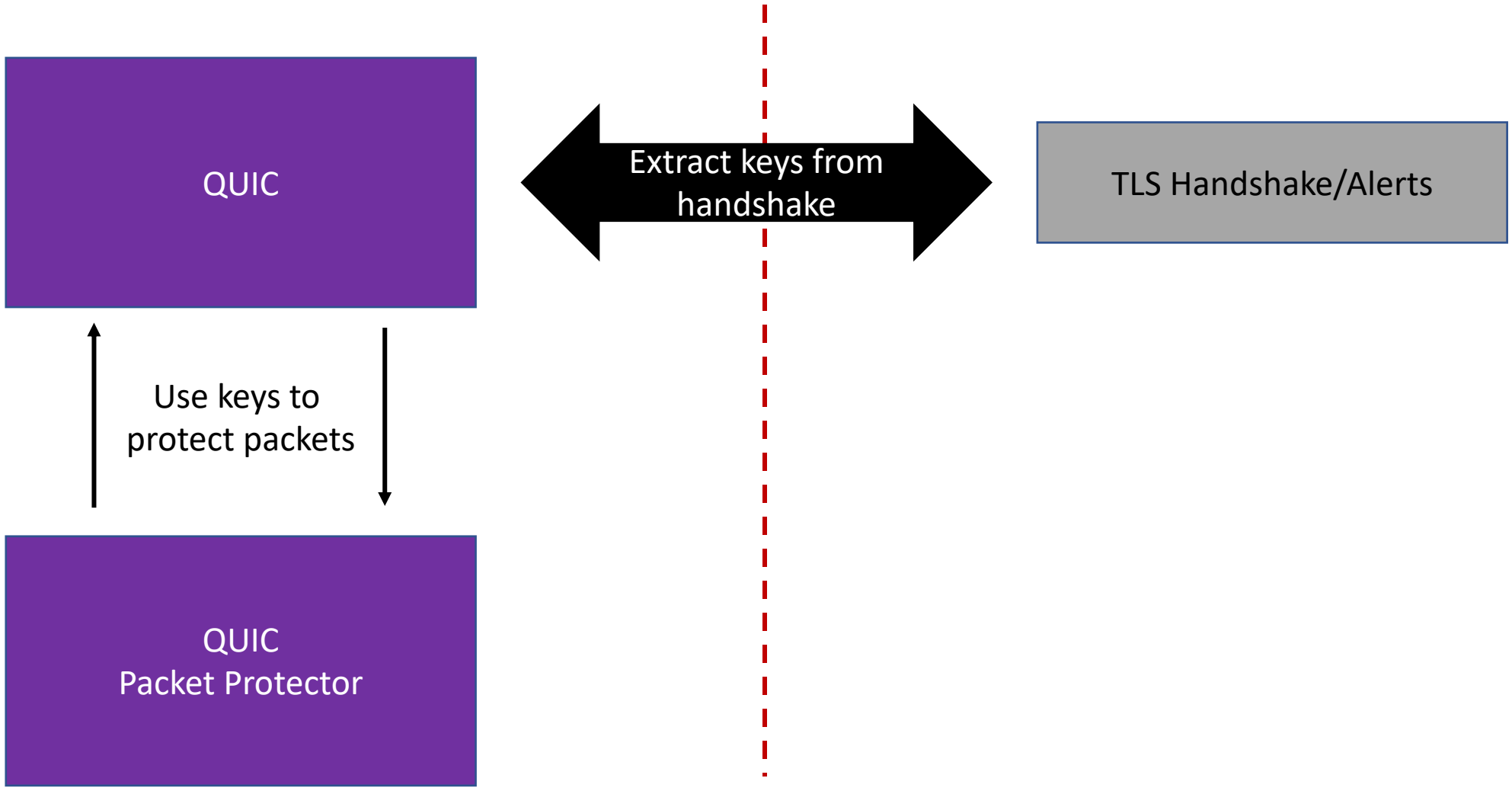
Separation between how handshake is performed and how the keys are used

Handshake Modularity

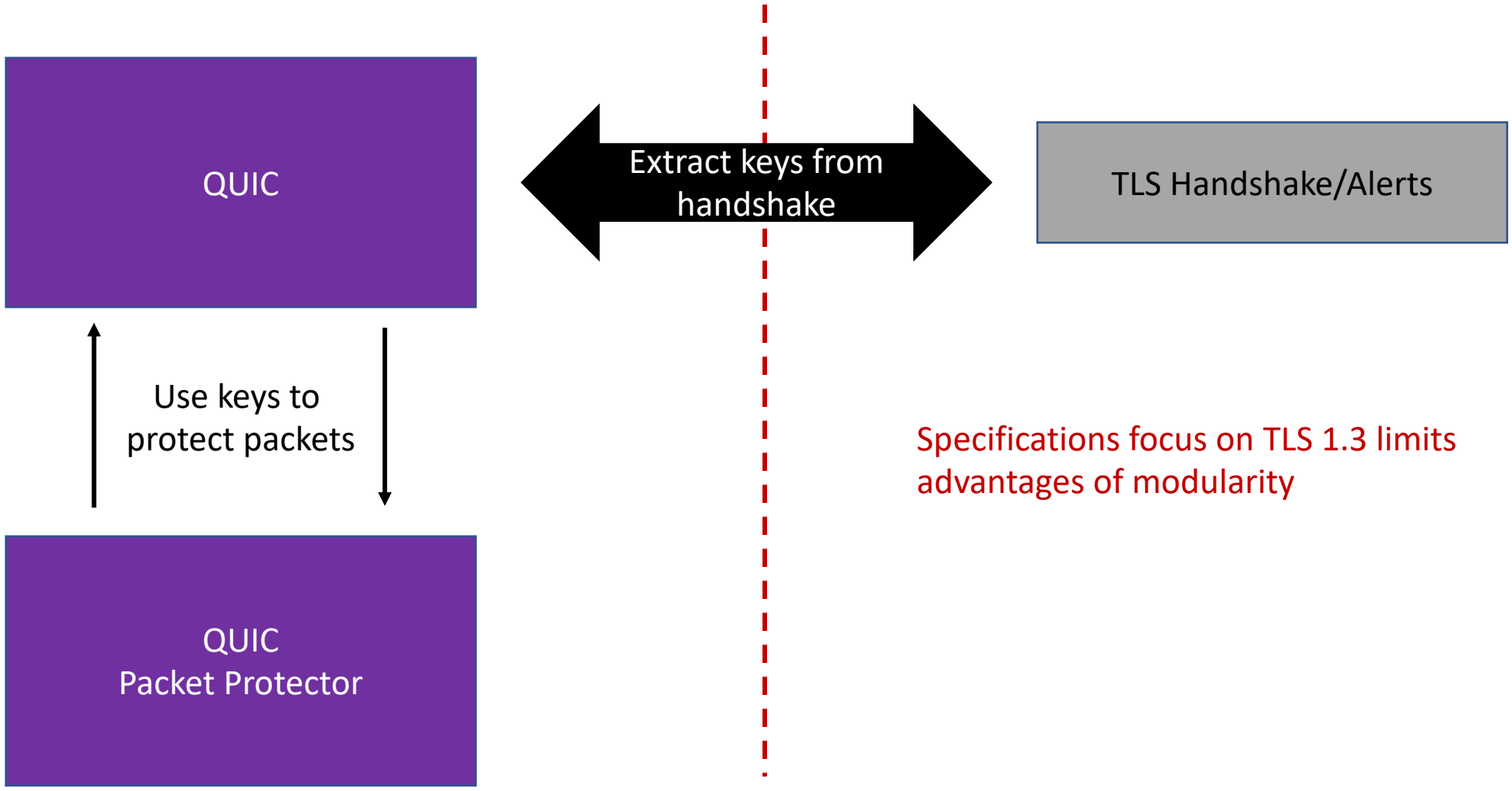


Separation between how handshake is performed and how the keys are used

Handshake Modularity?

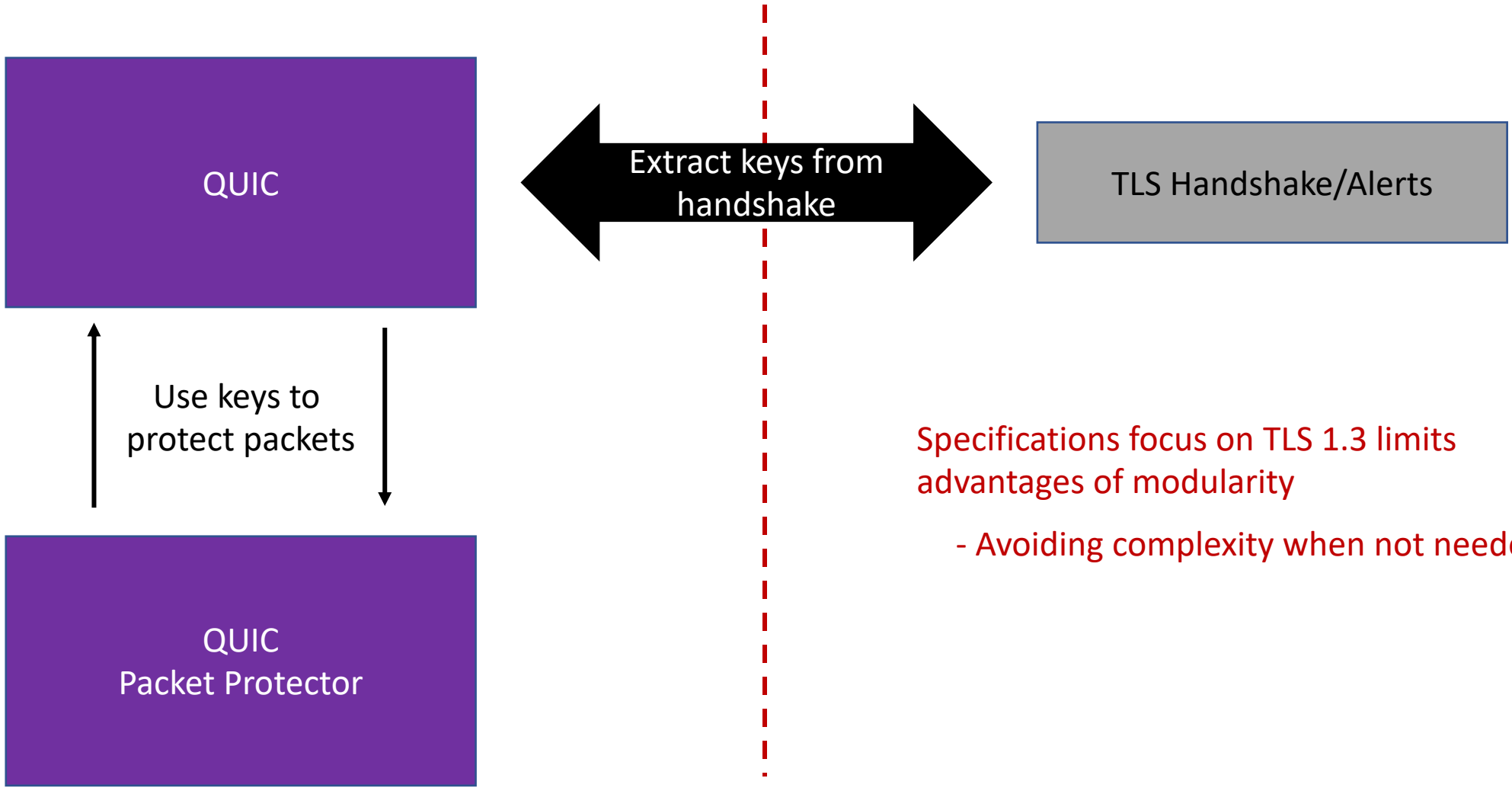


Handshake Modularity?



Specifications focus on TLS 1.3 limits advantages of modularity

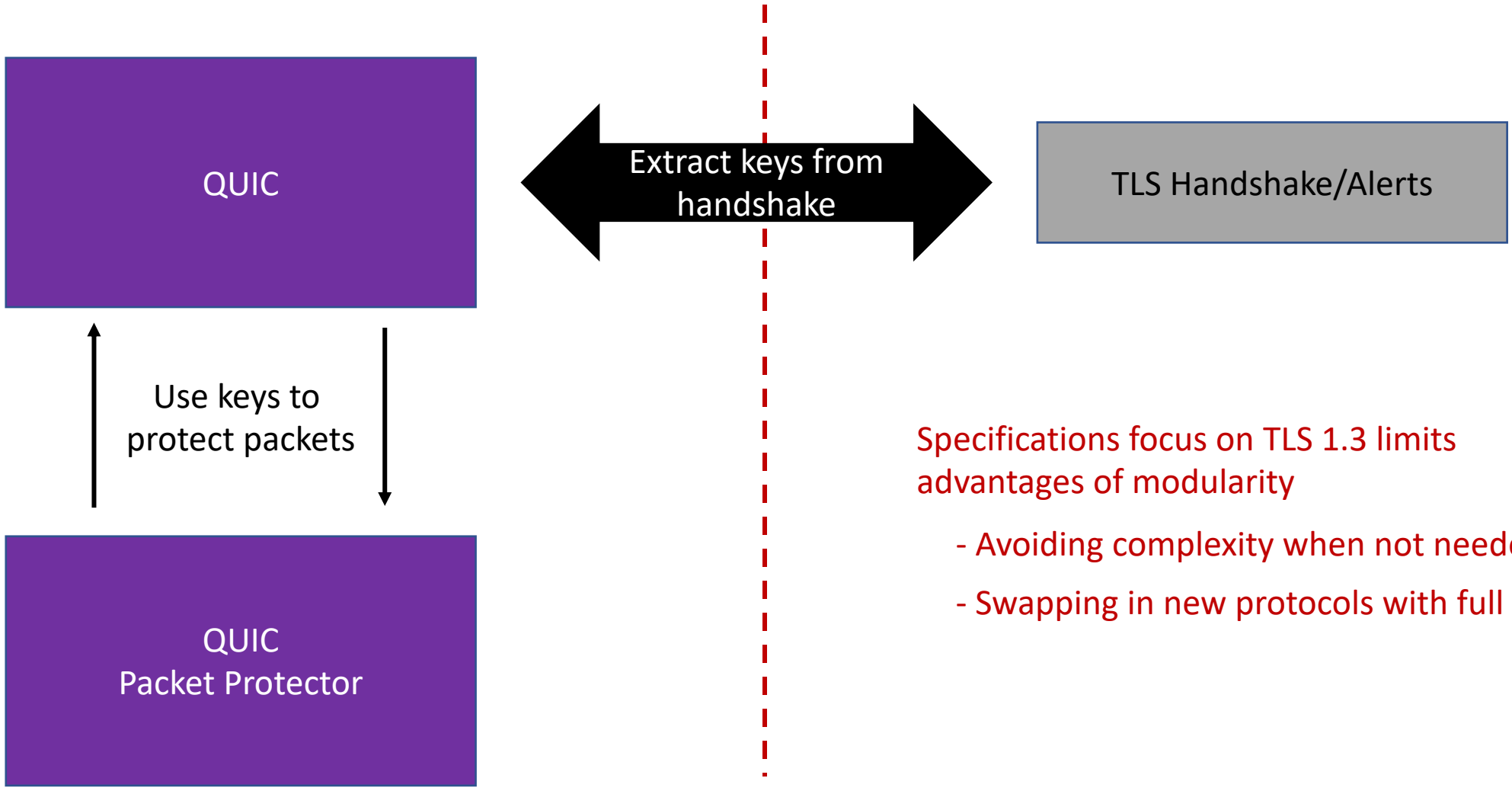
Handshake Modularity?



Specifications focus on TLS 1.3 limits advantages of modularity

- Avoiding complexity when not needed

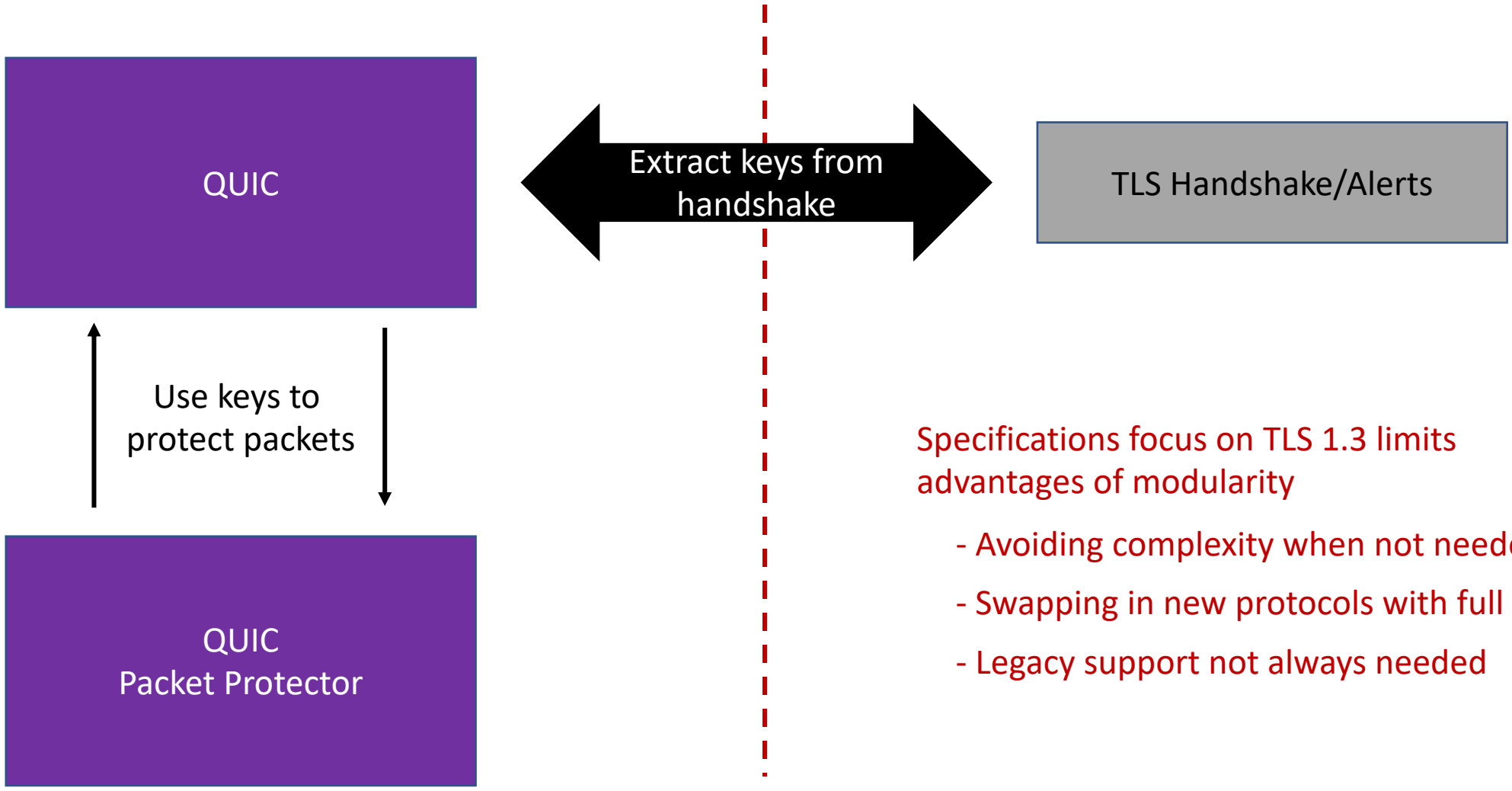
Handshake Modularity?



Specifications focus on TLS 1.3 limits advantages of modularity

- Avoiding complexity when not needed
- Swapping in new protocols with full security proofs

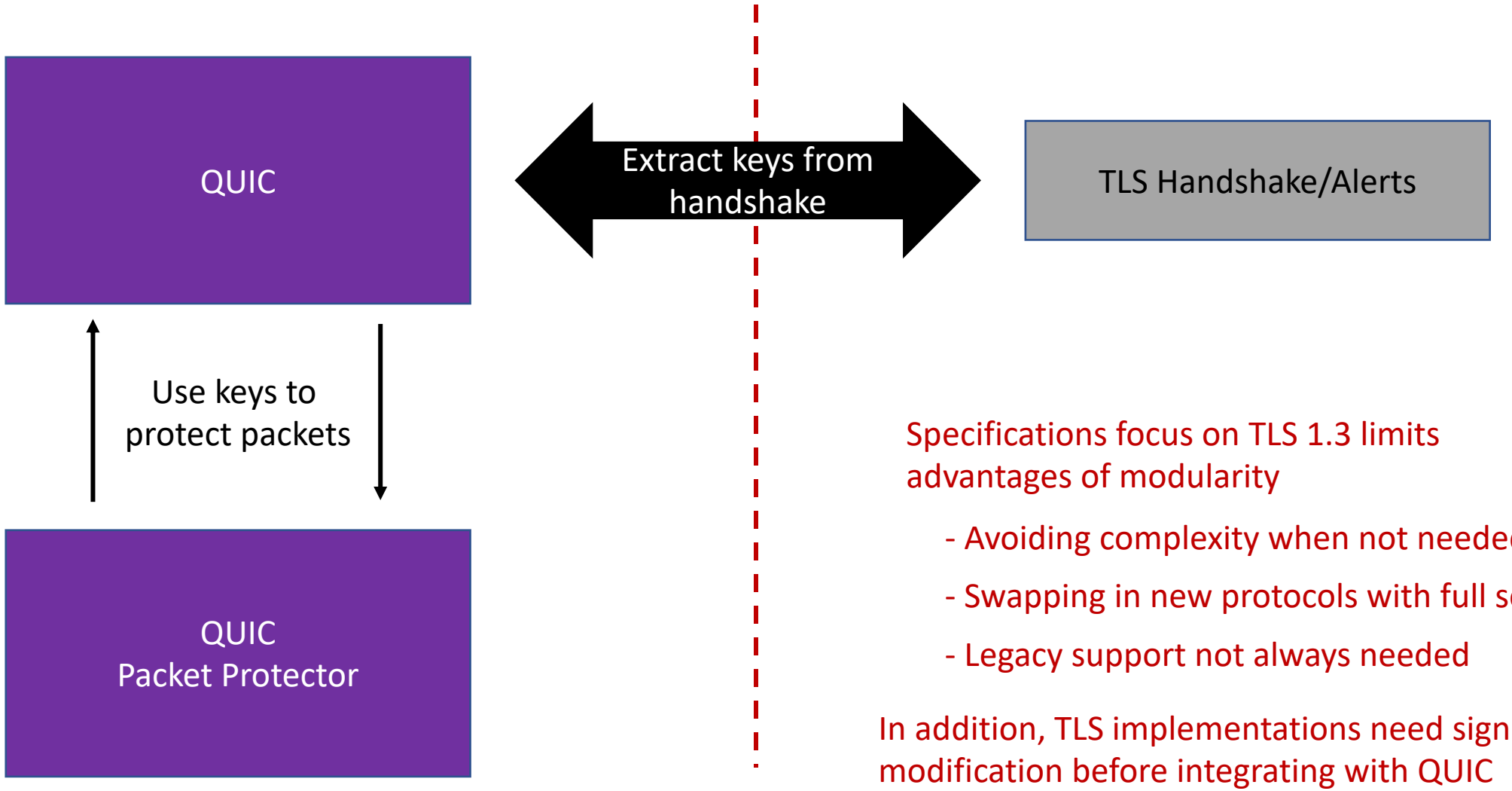
Handshake Modularity?



Specifications focus on TLS 1.3 limits advantages of modularity

- Avoiding complexity when not needed
- Swapping in new protocols with full security proofs
- Legacy support not always needed

Handshake Modularity?



Specifications focus on TLS 1.3 limits advantages of modularity

- Avoiding complexity when not needed
- Swapping in new protocols with full security proofs
- Legacy support not always needed

In addition, TLS implementations need significant modification before integrating with QUIC

Are there circumstances we can do better than TLS 1.3?

What is Noise

A *framework* for specifying Cryptographic Handshakes

What is Noise

A *framework* for specifying Cryptographic Handshakes

A variety of protocols can be specified using the simple Noise language

What is Noise

A *framework* for specifying Cryptographic Handshakes

A variety of protocols can be specified using the simple Noise language

These protocols can vary in their guarantees and complexity

What is Noise

A *framework* for specifying Cryptographic Handshakes

A variety of protocols can be specified using the simple Noise language

These protocols can vary in their guarantees and complexity

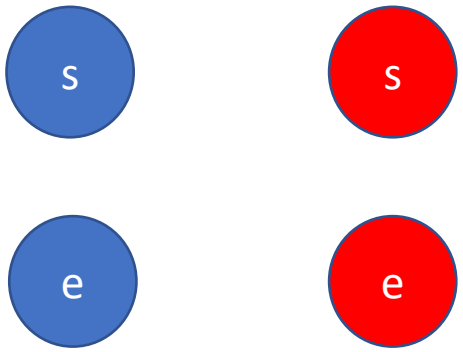
However, once a protocol is selected, the handshake proceeds in a straightforward fashion

What is Noise

The Noise language consists of tokens, which combine into message patterns, when combine into handshake patterns

What is Noise

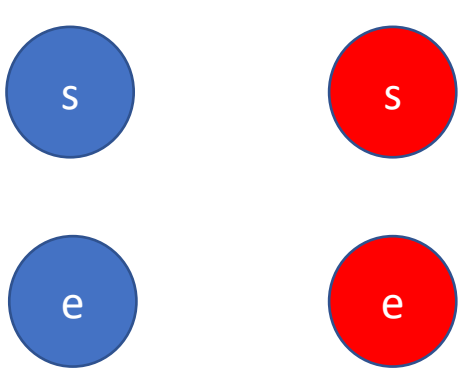
The Noise language consists of tokens, which combine into message patterns, when combine into handshake patterns



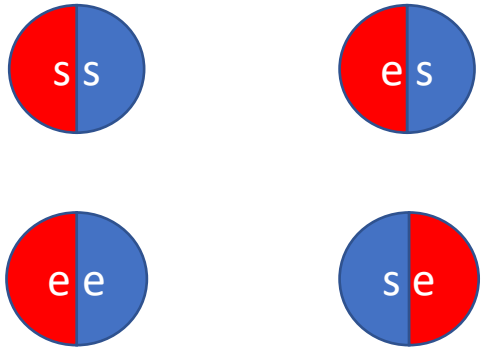
Public Key Tokens

What is Noise

The Noise language consists of tokens, which combine into message patterns, when combine into handshake patterns



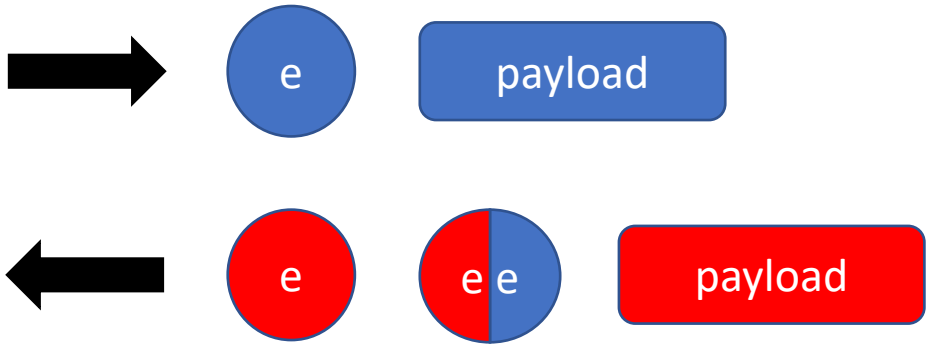
Public Key Tokens



DH Tokens

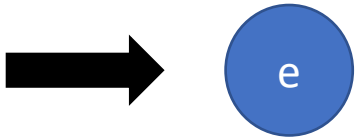
What is Noise

Here is a basic example handshake pattern



What is Noise

Here is a basic example handshake pattern



Initiator sends a public ephemeral DH share g^a

What is Noise

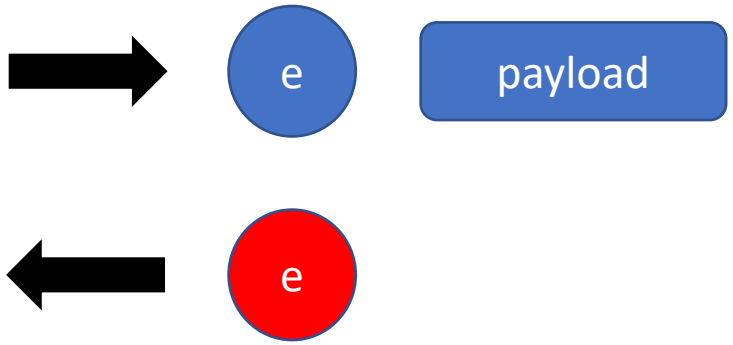
Here is a basic example handshake pattern

Initiator sends a public ephemeral DH share g^a

A cleartext payload is also sent over

What is Noise

Here is a basic example handshake pattern



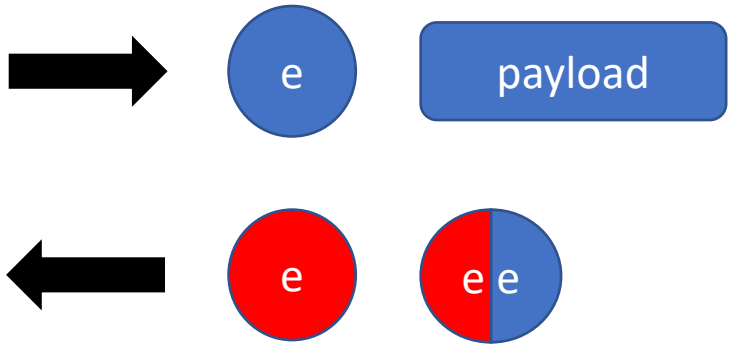
Initiator sends a public ephemeral DH share g^a

A cleartext payload is also sent over

Responder sends a public ephemeral DH share g^b

What is Noise

Here is a basic example handshake pattern



Initiator sends a public ephemeral DH share g^a

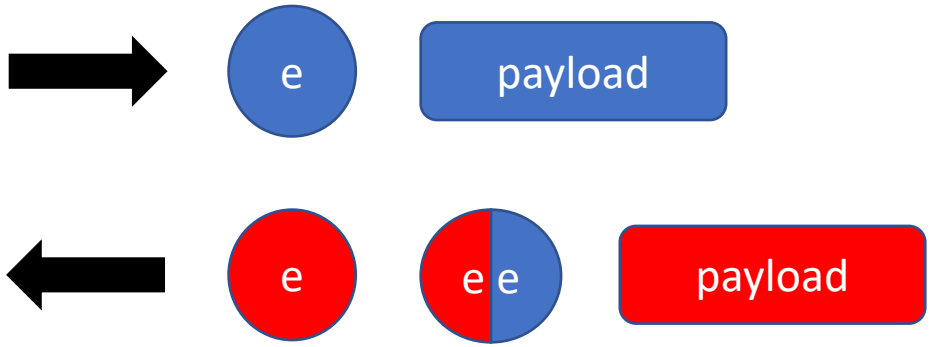
A cleartext payload is also sent over

Responder sends a public ephemeral DH share g^b

A DHKE is performed using these keys to obtain g^{ab}

What is Noise

Here is a basic example handshake pattern



Initiator sends a public ephemeral DH share g^a

A cleartext payload is also sent over

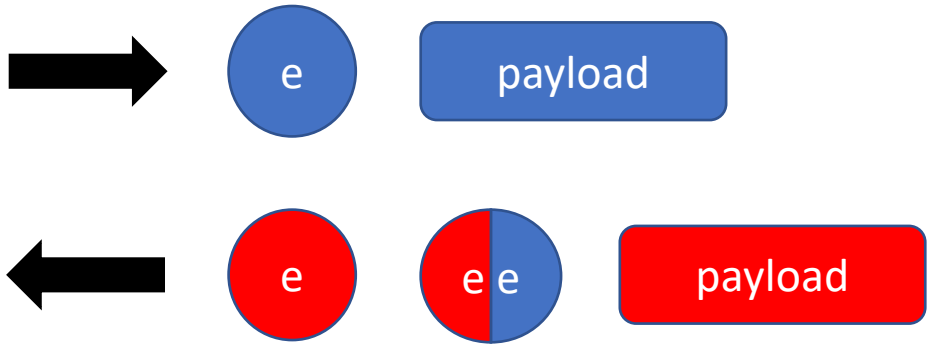
Responder sends a public ephemeral DH share g^b

A DHKE is performed using these keys to obtain g^{ab}

Responder sends payload encrypted under a derived key

What is Noise

Here is a basic example handshake pattern



Initiator sends a public ephemeral DH share g^a

A cleartext payload is also sent over

Responder sends a public ephemeral DH share g^b

A DHKE is performed using these keys to obtain g^{ab}

Responder sends payload encrypted under a derived key

Noise does additional processing to mix all handshake data into the derived key

Noise vs TLS

- Once a handshake pattern is selected, noise follows a simple linear state machine

Noise vs TLS

- Once a handshake pattern is selected, Noise follows a simple linear state machine

- Noise is easy to prove secure

Noise vs TLS

- Once a handshake pattern is selected, Noise follows a simple linear state machine
- Noise is easy to prove secure
- Noise is generally implemented as a “build your own protocol” library

Noise vs TLS

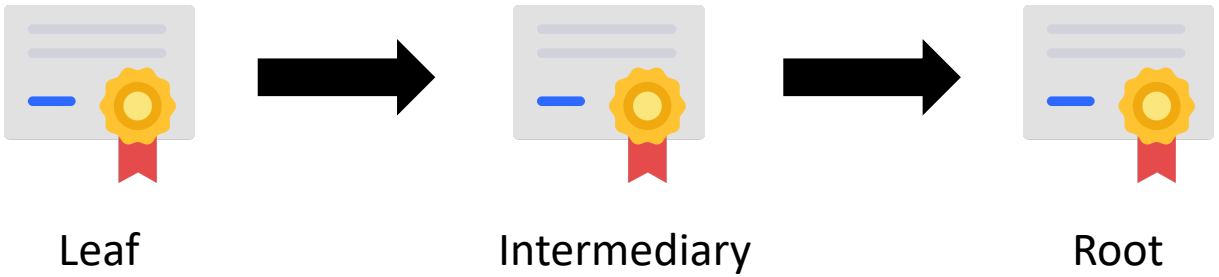
- Once a handshake pattern is selected, Noise follows a simple linear state machine
- Noise is easy to prove secure
- Noise is generally implemented as a “build your own protocol” library
- Noise lacks cryptographic agility

Peer Authentication and Pinning

Traditionally, Authentication of peers in TLS involves a PKI

Peer Authentication and Pinning

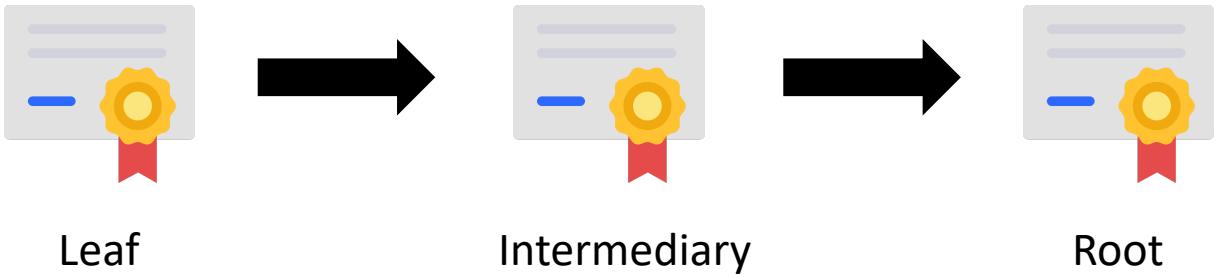
Traditionally, Authentication of peers in TLS involves a PKI



Chain of Trust

Peer Authentication and Pinning

Traditionally, Authentication of peers in TLS involves a PKI



Chain of Trust

However this is not necessary in a centrally managed setting

Peer Authentication and Pinning

Pinning instructs a peer to expect a specific key

Peer Authentication and Pinning

Pinning instructs a peer to expect a specific key

This is similar to the Preshared Symmetric Keys (PSKs) setting

Peer Authentication and Pinning

Pinning instructs a peer to expect a specific key

This is similar to the Preshared Symmetric Keys (PSKs) setting

However, PSKs require many more keys, since every pair of endpoints must have its own unique key

Peer Authentication and Pinning

Pinning instructs a peer to expect a specific key

This is similar to the Preshared Symmetric Keys (PSKs) setting

However, PSKs require many more keys, since every pair of endpoints must have its own unique key

$$n \quad \text{vs} \quad \frac{n!}{2 \cdot (n-2)!}$$

Peer Authentication and Pinning

nQUIC is designed for the public key pinning setting

Peer Authentication and Pinning

nQUIC is designed for the public key pinning setting

This applies to cases where:

- Public keys or Certificate Chains are obtained out-of-band

Peer Authentication and Pinning

nQUIC is designed for the public key pinning setting

This applies to cases where:

- Public keys or Certificate Chains are obtained out-of-band

- Peers are bootstrapped with keys

Peer Authentication and Pinning

nQUIC is designed for the public key pinning setting

This applies to cases where:

- Public keys or Certificate Chains are obtained out-of-band
- Peers are bootstrapped with keys
- Public keys are managed by a trusted key management service

nQUIC

Motivated by simplicity while still satisfying the following requirements:

1. Authenticated Key Exchange
2. Authentication of Transport Parameters
3. Authenticated Version Negotiation
4. Authenticated Negotiation of Application Protocol
5. Address Validation

nQUIC

Motivated by simplicity while still satisfying the following requirements:

1. Authenticated Key Exchange
   Feature of Noise
2. Authentication of Transport Parameters
   Can be placed in the payload field
3. Authenticated Version Negotiation
   Can be placed in the payload field
4. Authenticated Negotiation of Application Protocol
   ALPN data can be placed in transport parameters
5. Address Validation
   Handled by QUIC address validation tokens

nQUIC’s Noise Pattern

We needed a handshake that:

nQUIC’s Noise Pattern

We needed a handshake that:

Authenticates the server

nQUIC’s Noise Pattern

We needed a handshake that:

Authenticates the server

Optionally authenticates the client

nQUIC's Noise Pattern

We needed a handshake that:

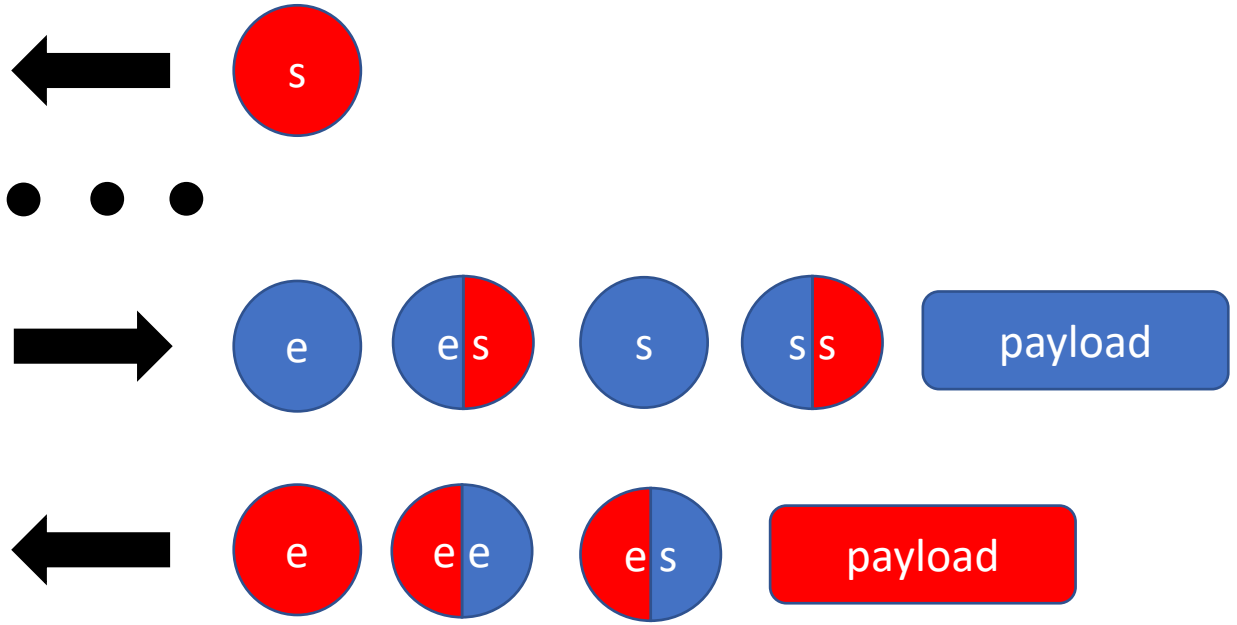
Authenticates the server

Optionally authenticates the client

Encrypts transport parameters

nQUIC's Noise Pattern

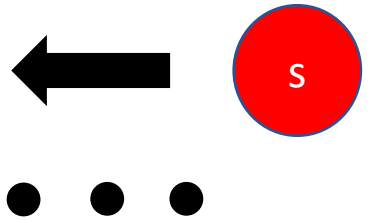
In order to achieve this we selected the IK pattern:



nQUIC’s Noise Pattern

In order to achieve this we selected the IK pattern:

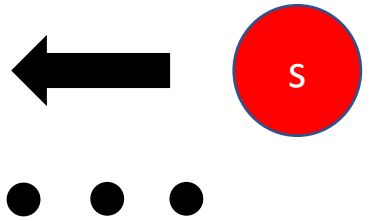
Server shares static key in advance



nQUIC's Noise Pattern

In order to achieve this we selected the IK pattern:

Server shares static key in advance

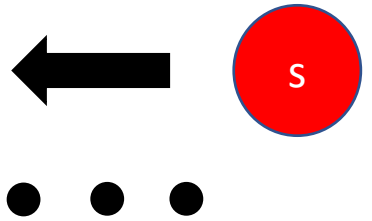


Client payload secured by client ephemeral and server static keys

nQUIC's Noise Pattern

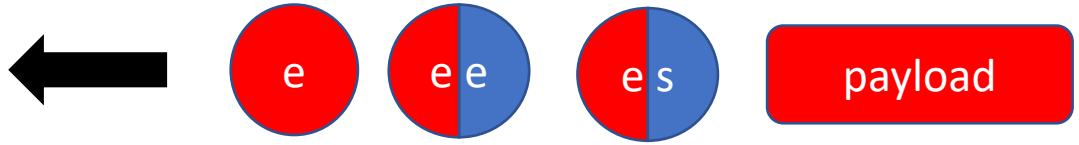
In order to achieve this we selected the IK pattern:

Server shares static key in advance



Client payload secured by client ephemeral and server static keys

Server payload secured by both parties ephemeral keys



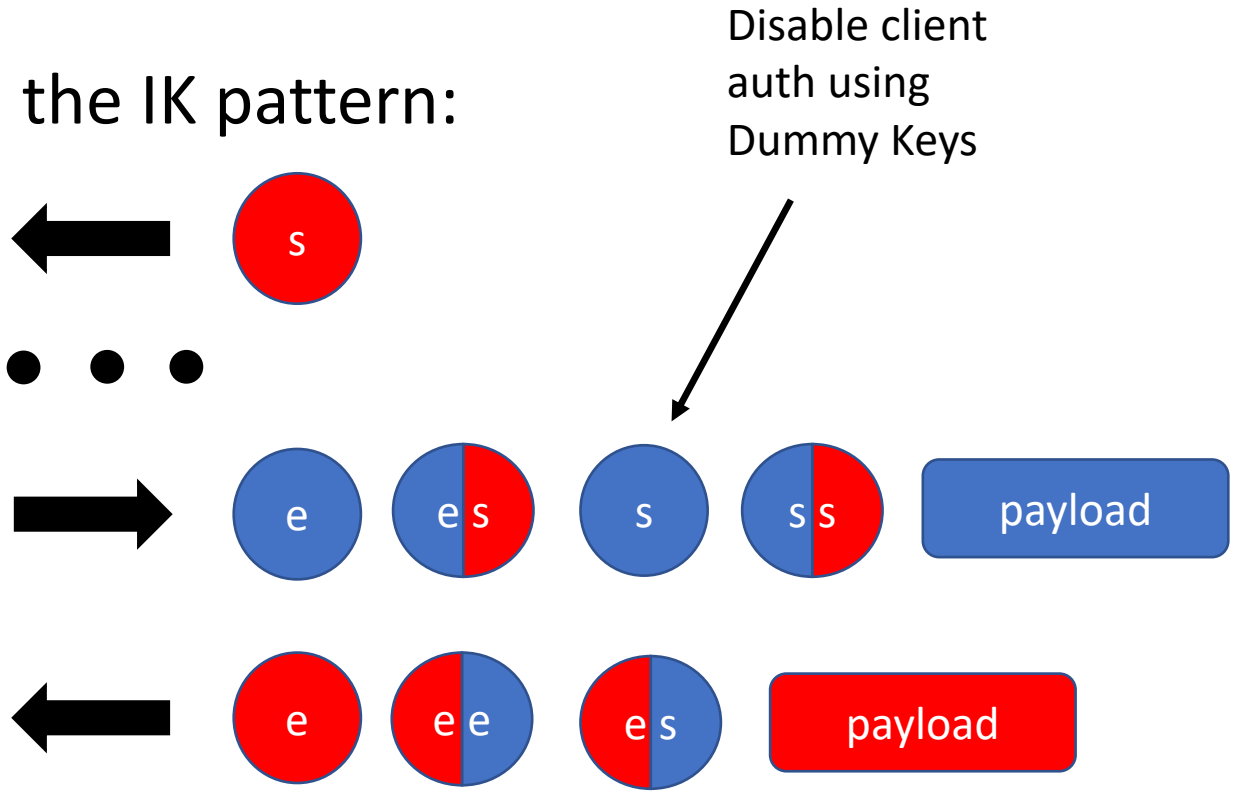
nQUIC's Noise Pattern

In order to achieve this we selected the IK pattern:

Server shares static key in advance

Client payload secured by client ephemeral and server static keys

Server payload secured by both parties ephemeral keys



Why IK?

There were other Noise patterns that offered similar guarantees

Why IK?

There were other Noise patterns that offered similar guarantees

Some patterns did not require server to share static key beforehand

Why IK?

There were other Noise patterns that offered similar guarantees

Some patterns did not require server to share static key beforehand

- Supports more settings, but also may require a PKI

Why IK?

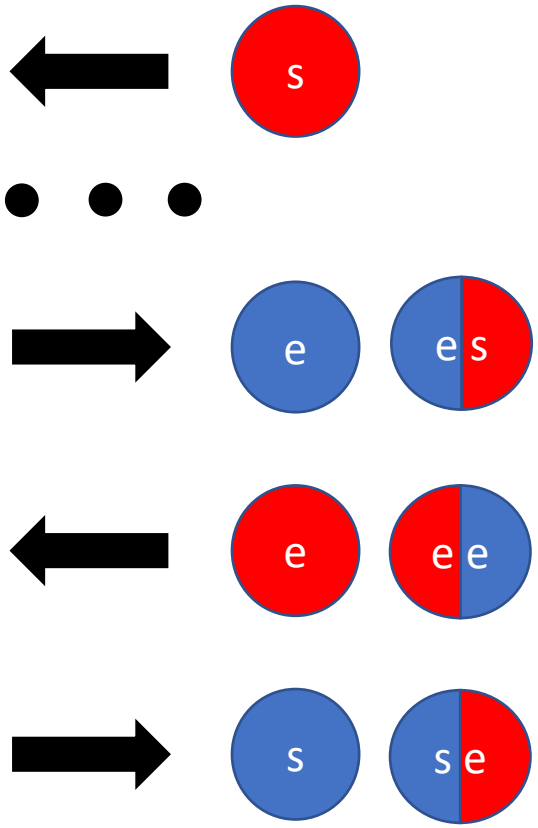
There were other Noise patterns that offered similar guarantees

Some patterns did not require server to share static key beforehand

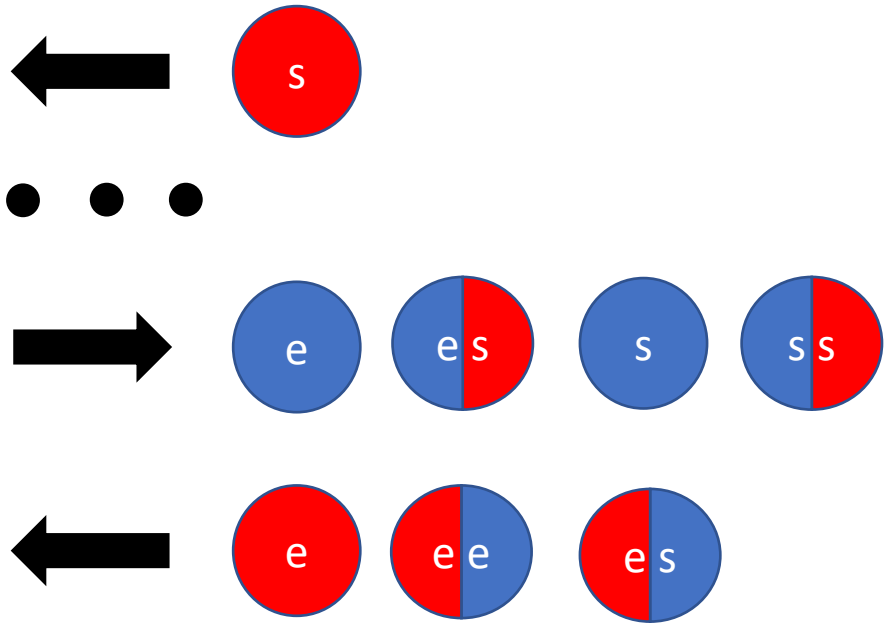
- Supports more settings, but also may require a PKI

The XK Pattern also sends server static key beforehand, but is 3 rounds

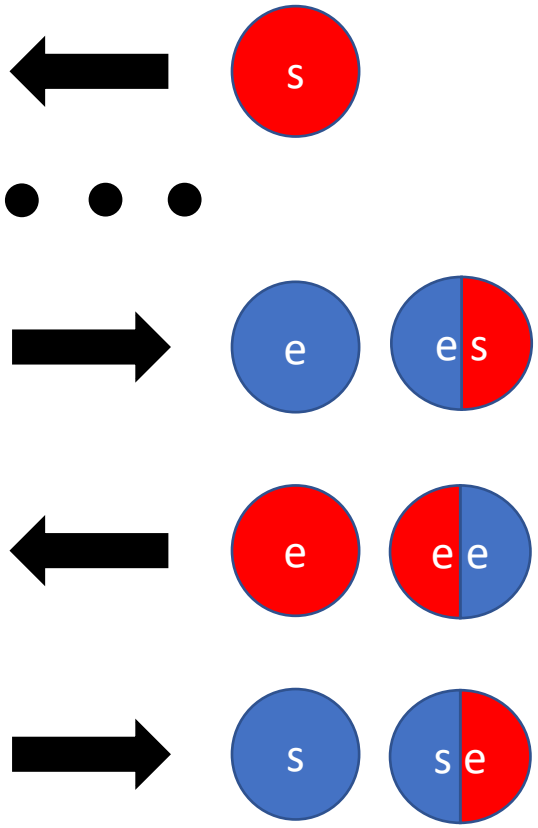
XK



IK

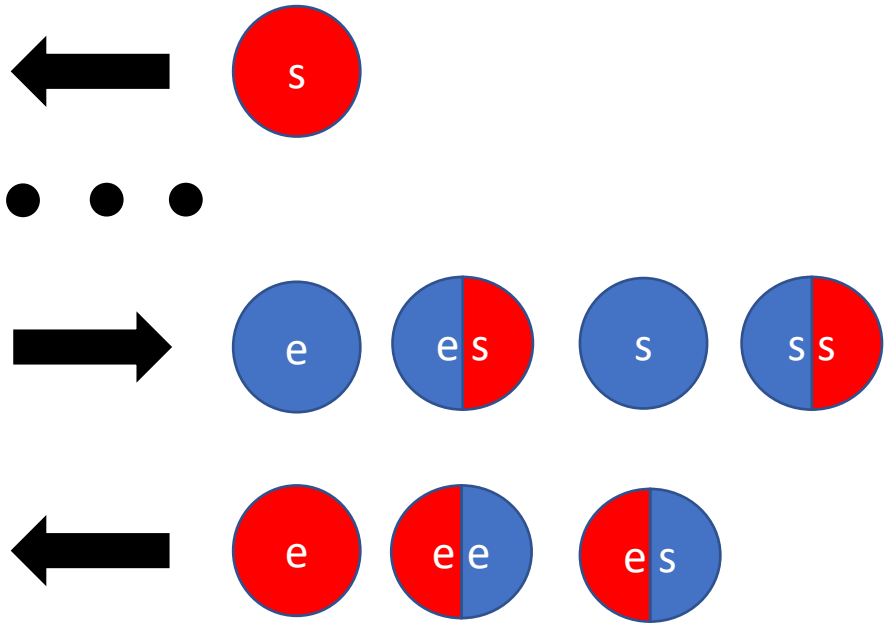


XK

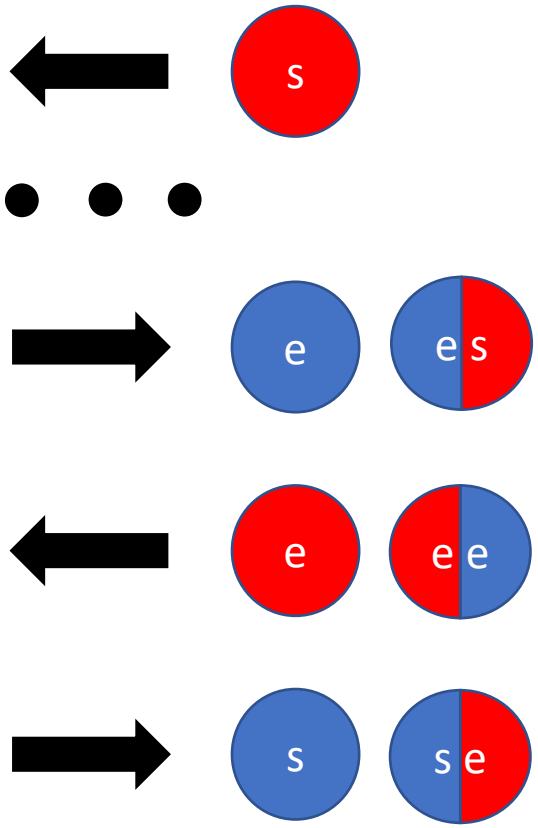


- More rounds

IK

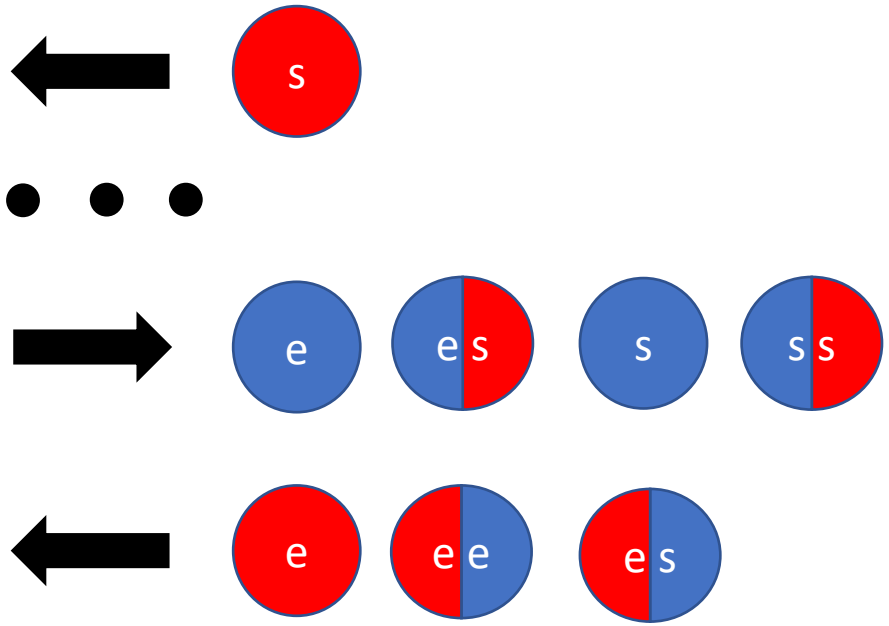


XK



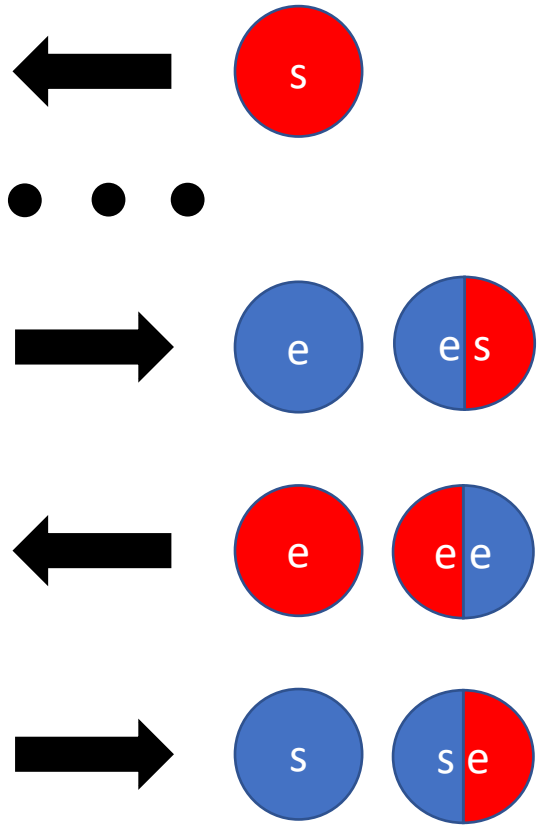
- More rounds

IK



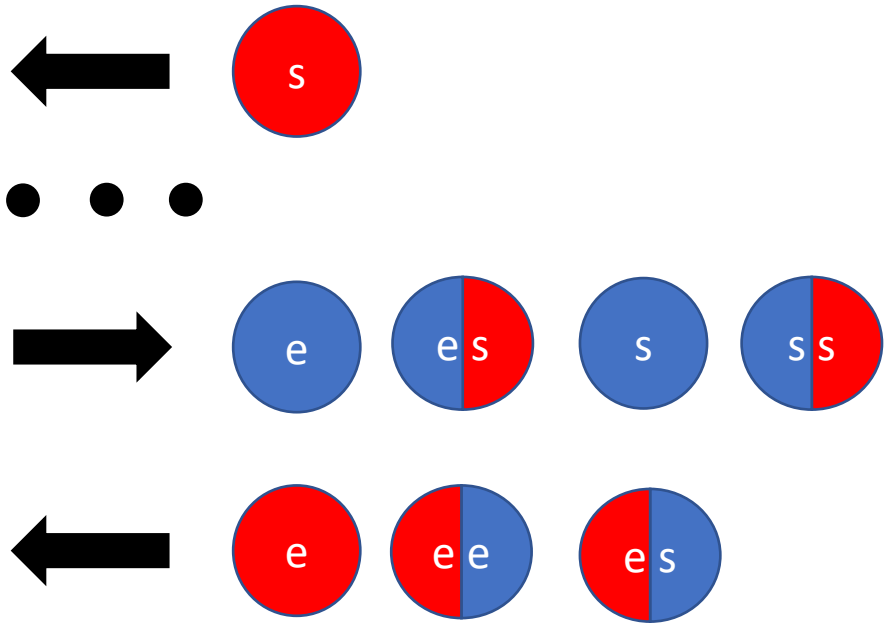
- More DHs

XK



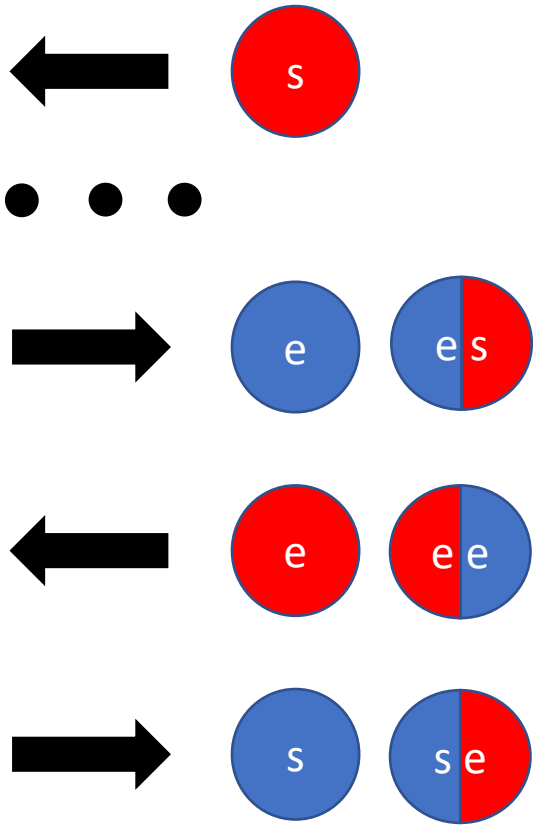
- More rounds

IK



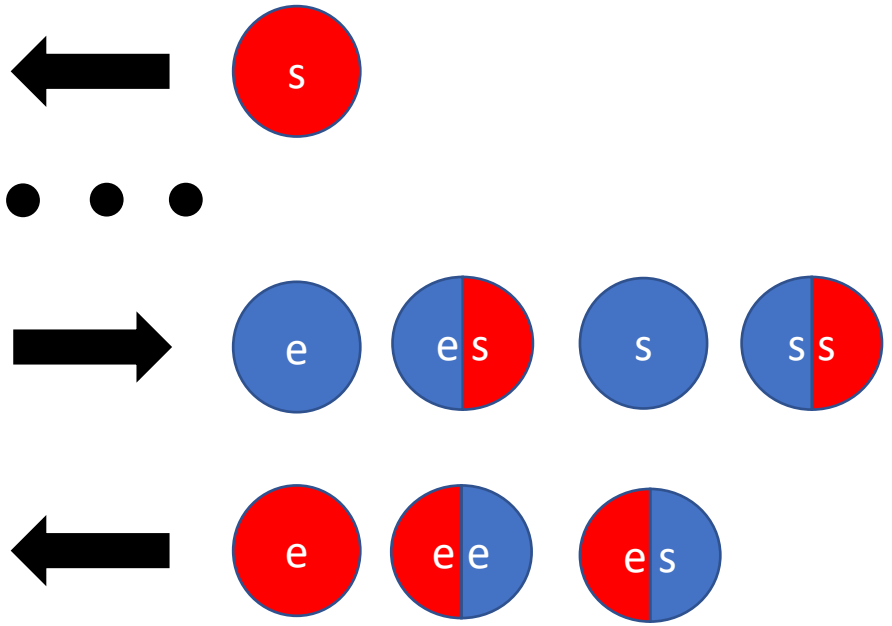
- More DHs
- Weaker Security of first message

XK



- More rounds

IK



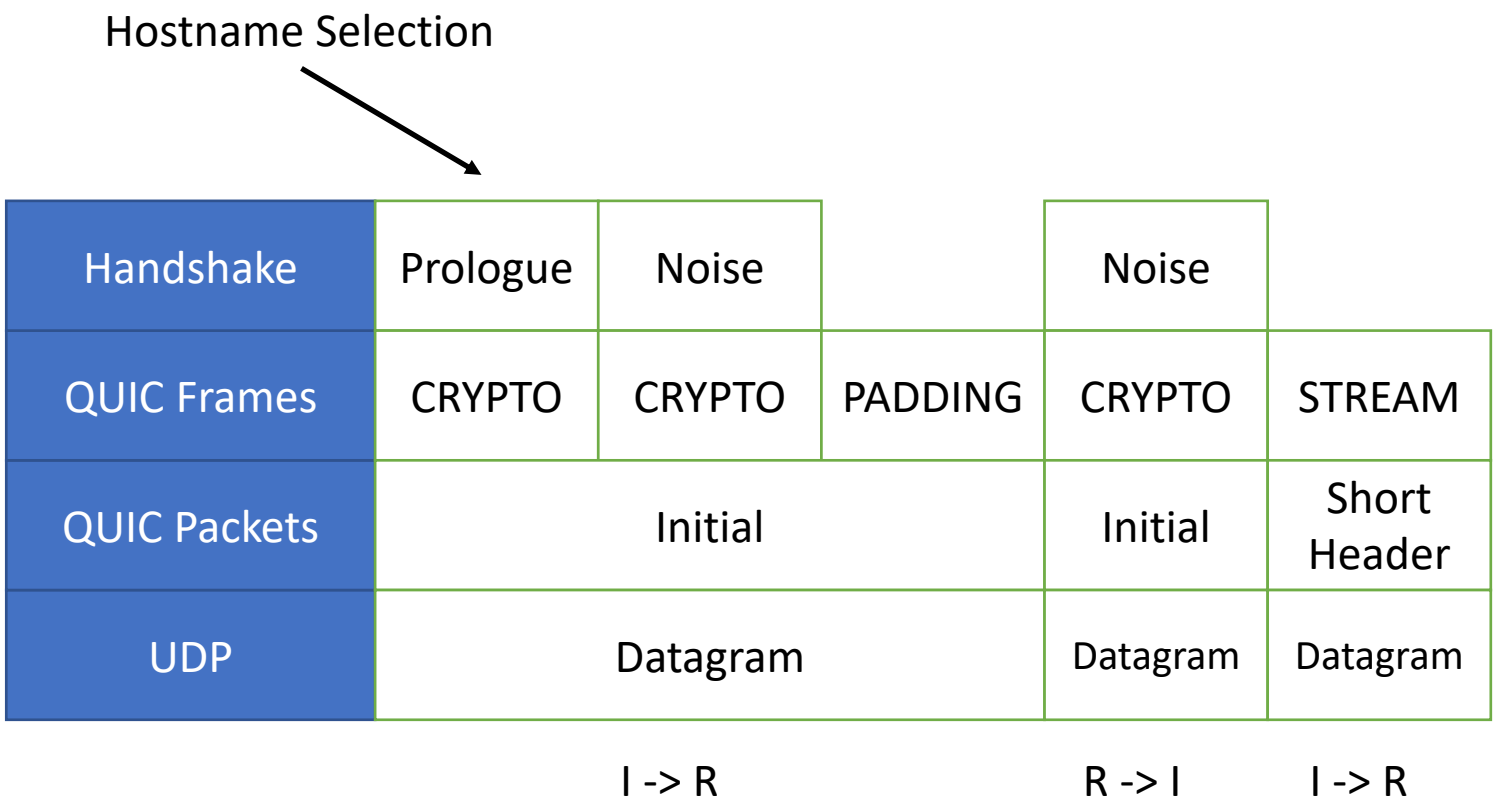
- More DHs
- Weaker Security of first message

Ultimately, we found network latency to be the most serious constraint

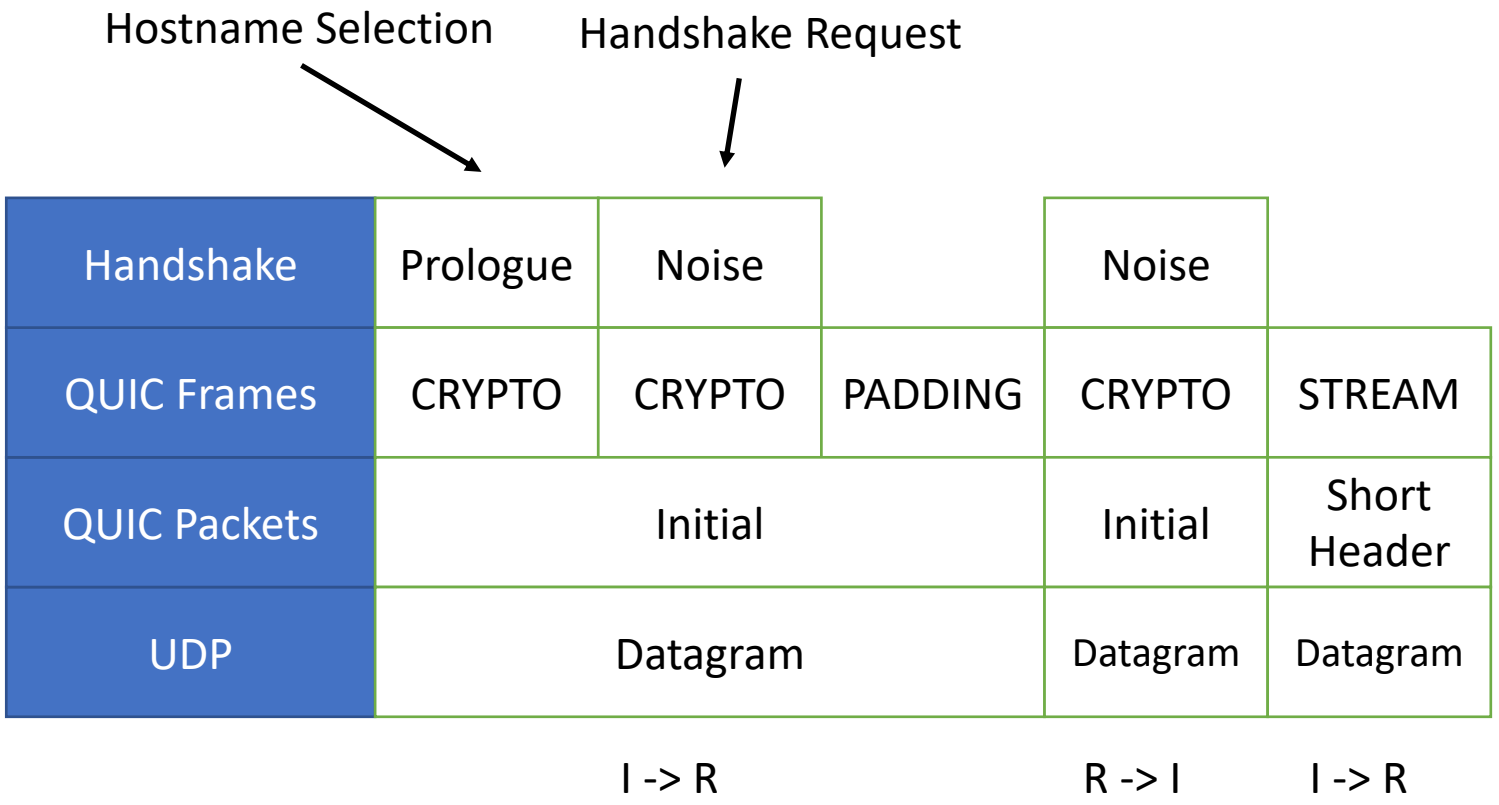
Anatomy of nQUIC

Handshake	Prologue	Noise		Noise	
QUIC Frames	CRYPTO	CRYPTO	PADDING	CRYPTO	STREAM
QUIC Packets	Initial			Initial	Short Header
UDP	Datagram			Datagram	Datagram
	I -> R			R -> I	I -> R

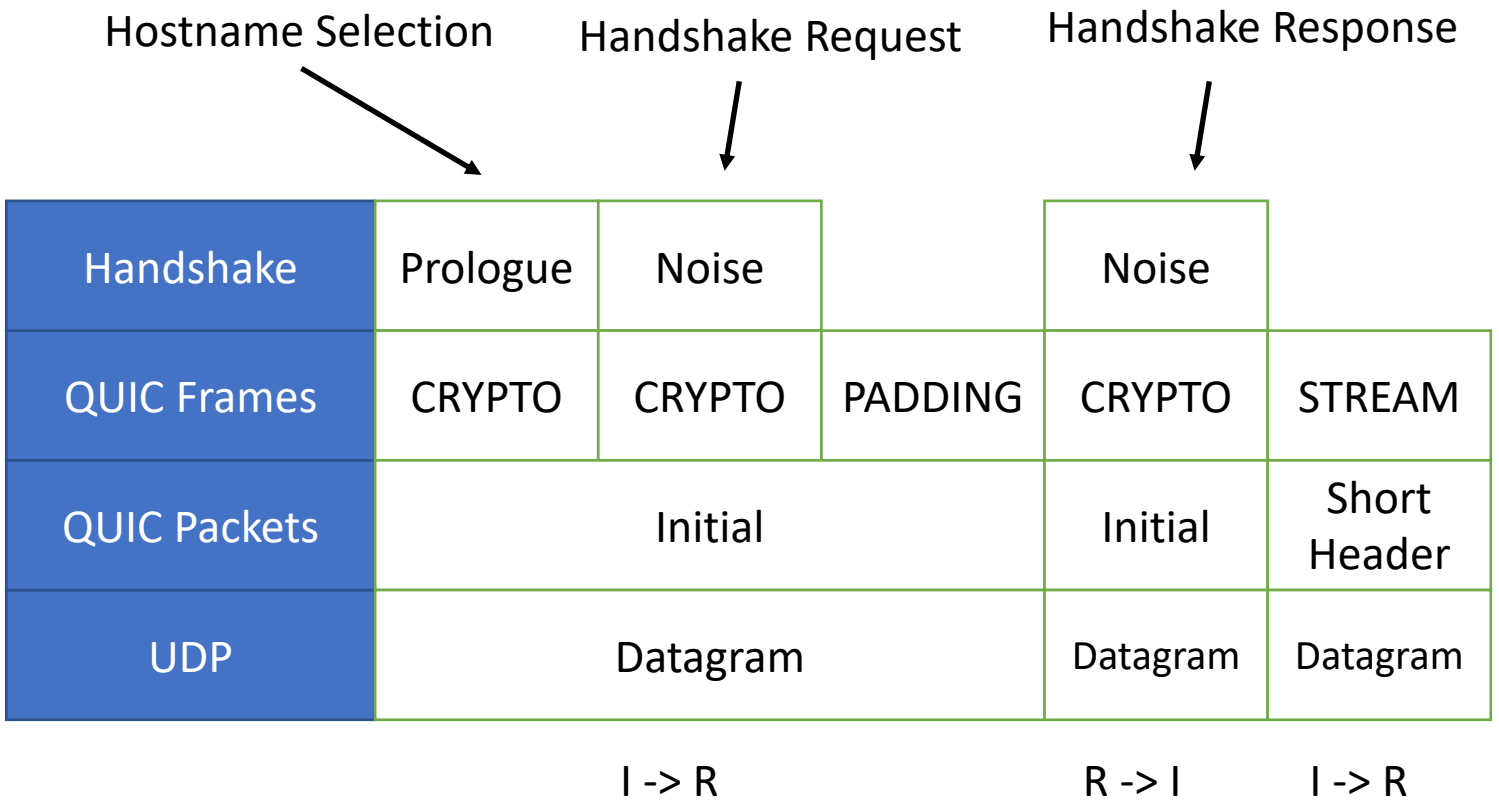
Anatomy of nQUIC



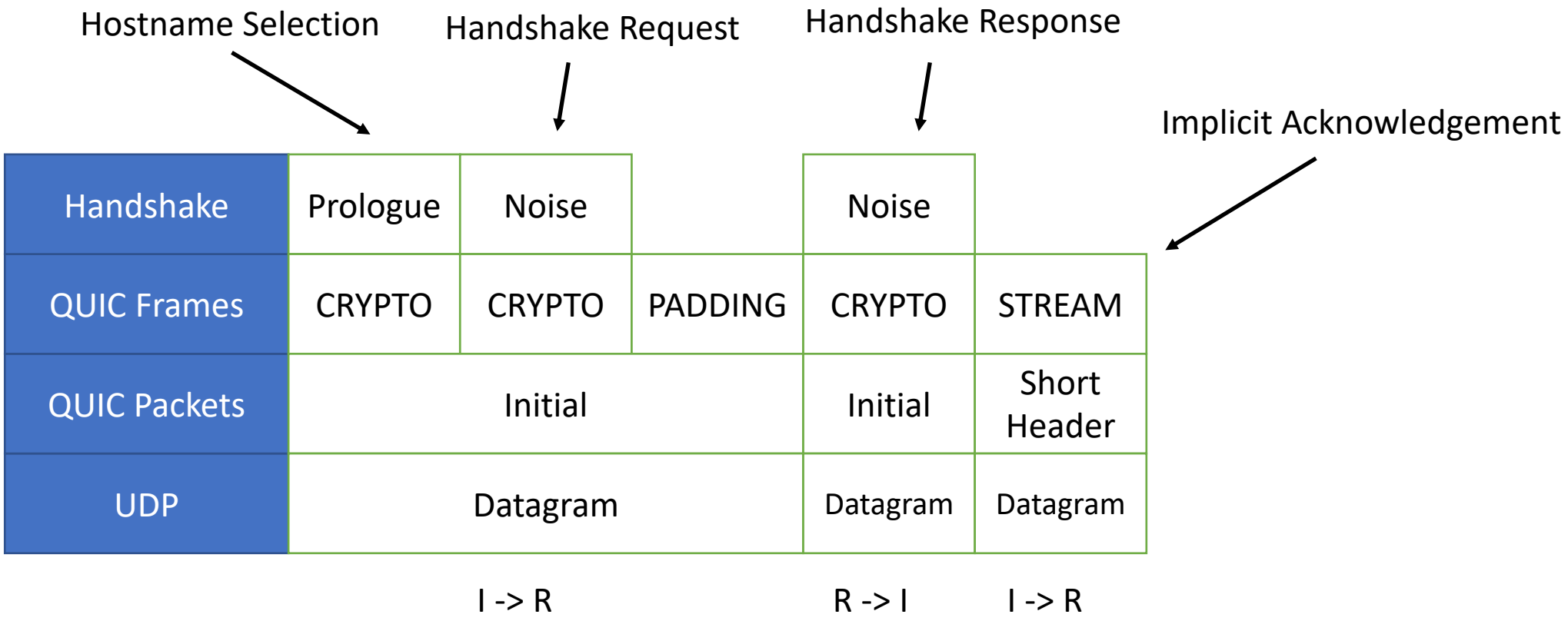
Anatomy of nQUIC



Anatomy of nQUIC



Anatomy of nQUIC

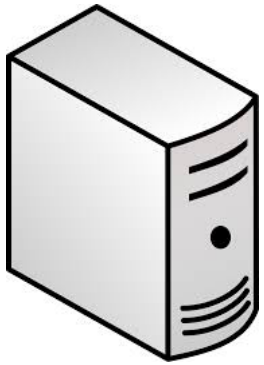


Hostname Selection

Problem: A server may be responsible for multiple endpoints, only one of which has the correct key.

Hostname Selection

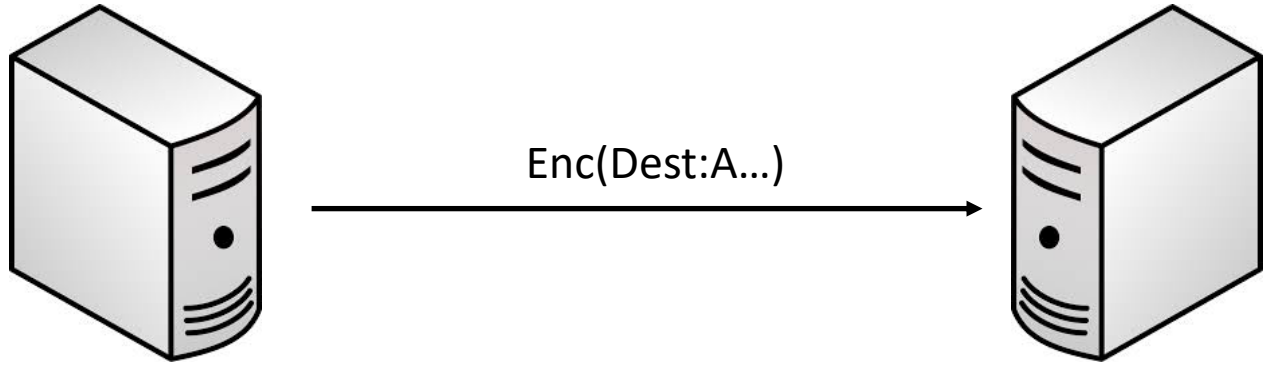
Problem: A server may be responsible for multiple endpoints, only one of which has the correct key.



Has key pinning for A

Hostname Selection

Problem: A server may be responsible for multiple endpoints, only one of which has the correct key.

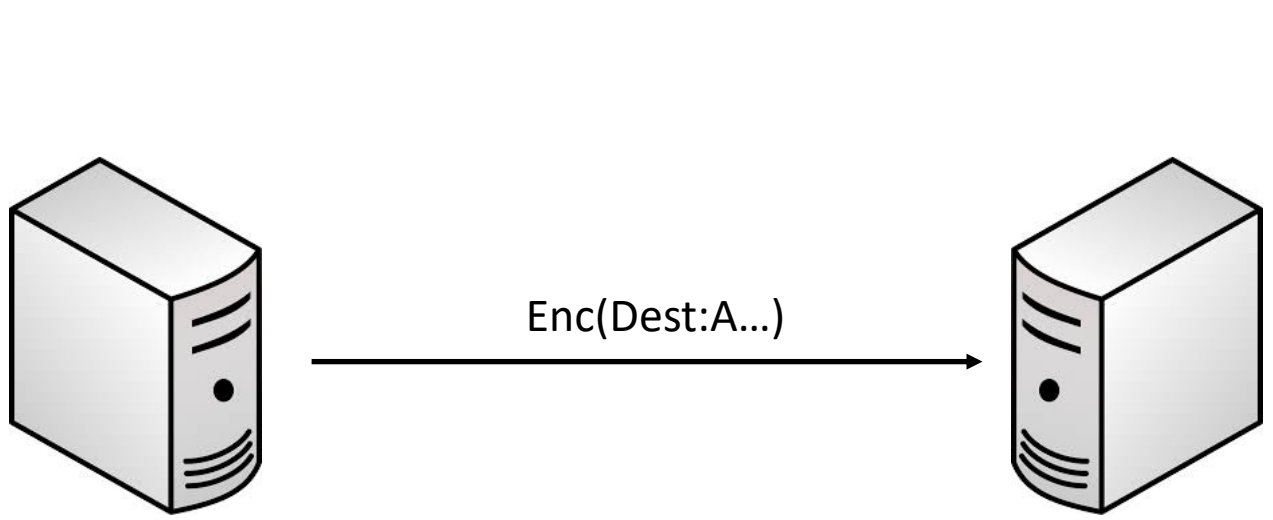


Has key pinning for A

Load Balancer or Server hosting multiple endpoints on same machine

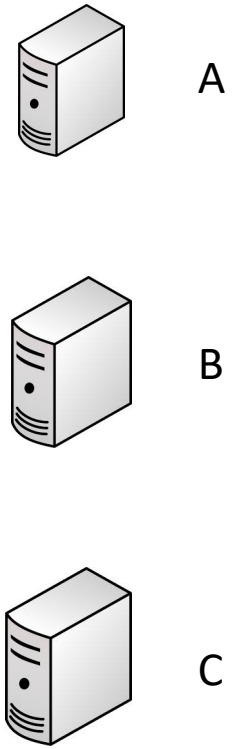
Hostname Selection

Problem: A server may be responsible for multiple endpoints, only one of which has the correct key.



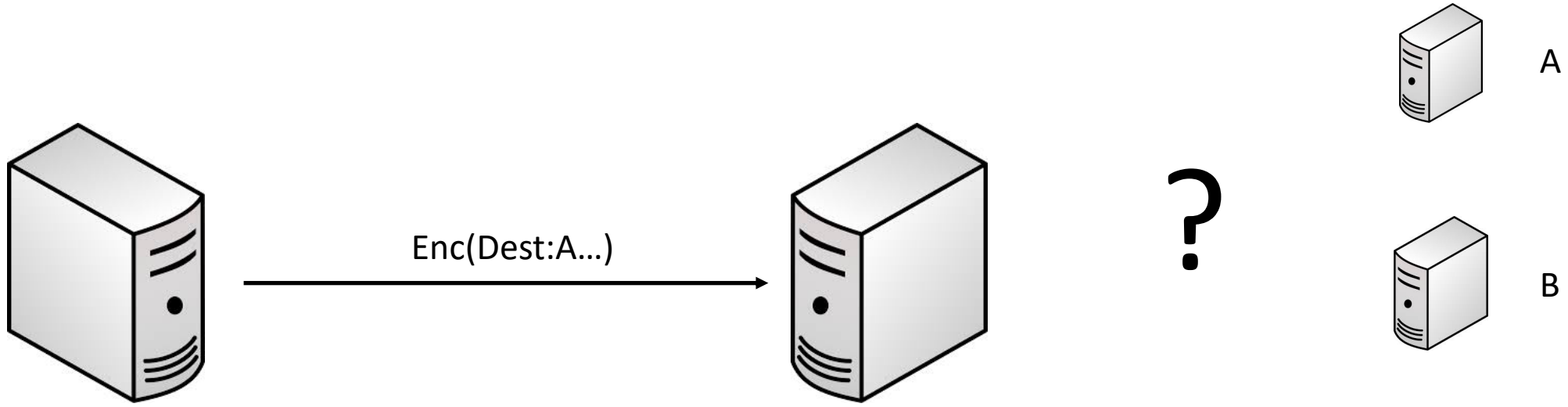
Has key pinning for A

Load Balancer or Server hosting multiple endpoints on same machine



Hostname Selection

Problem: A server may be responsible for multiple endpoints, only one of which has the correct key.

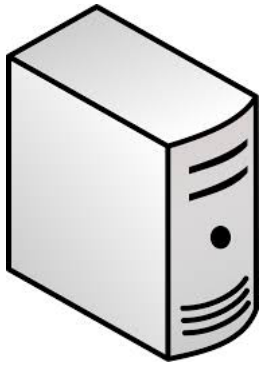


Has key pinning for A

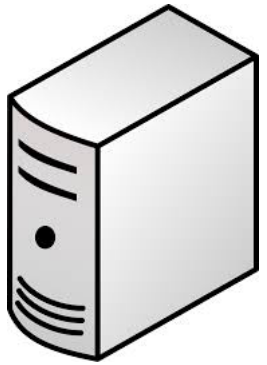
Load Balancer or Server hosting multiple endpoints on same machine

Hostname Selection

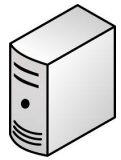
For TLS, SNI solves this by including the server id as part of the TLS negotiation



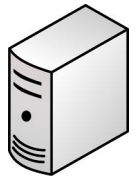
Has key pinning for A



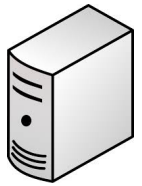
Load Balancer or Server hosting multiple endpoints on same machine



A



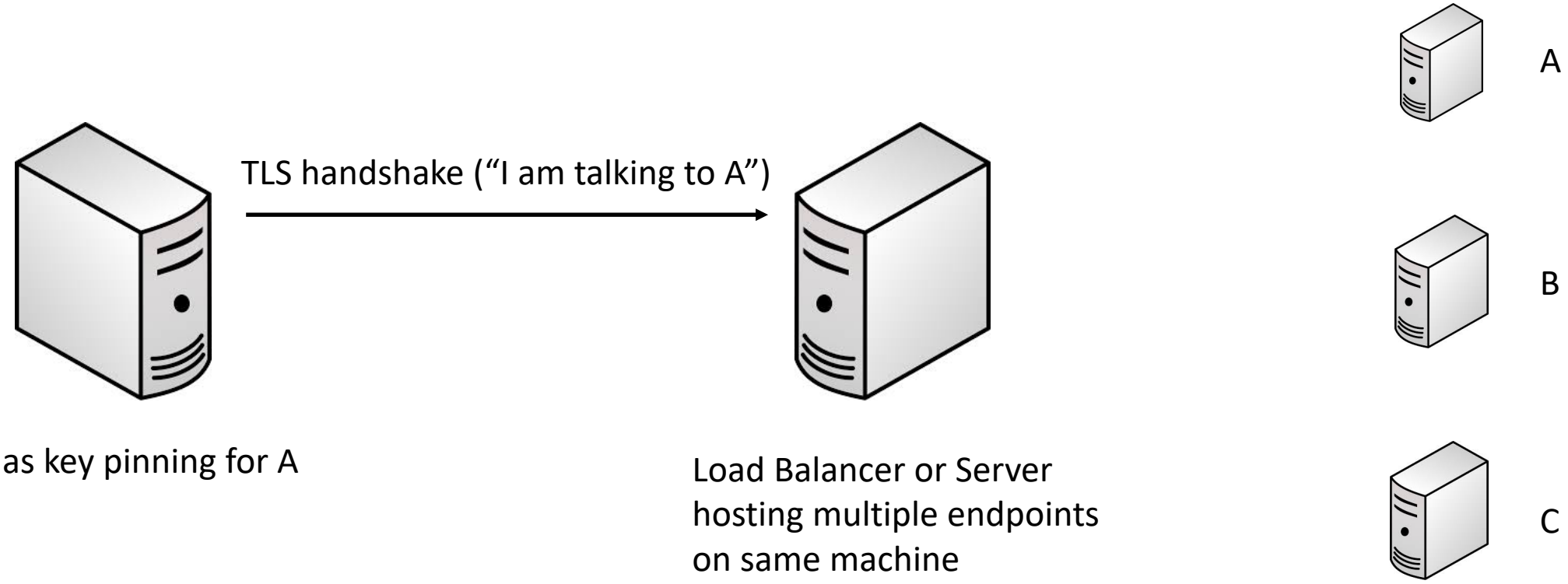
B



C

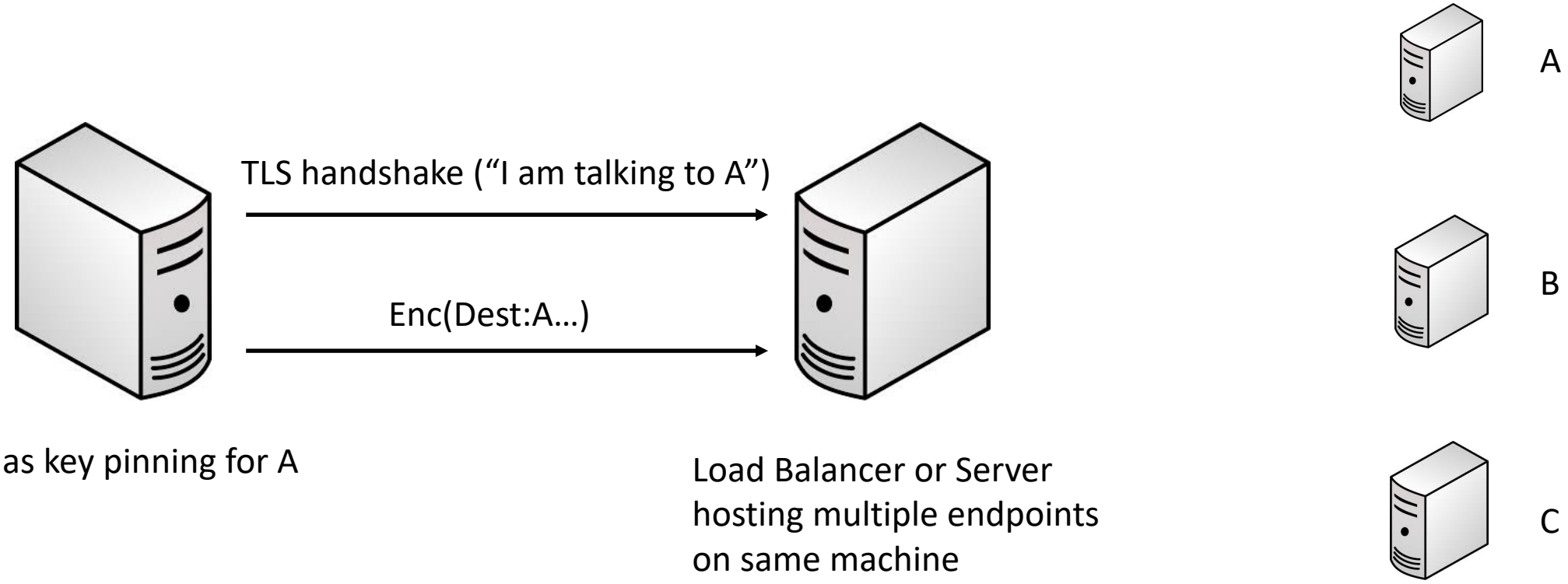
Hostname Selection

For TLS, SNI solves this by including the server id as part of the TLS negotiation



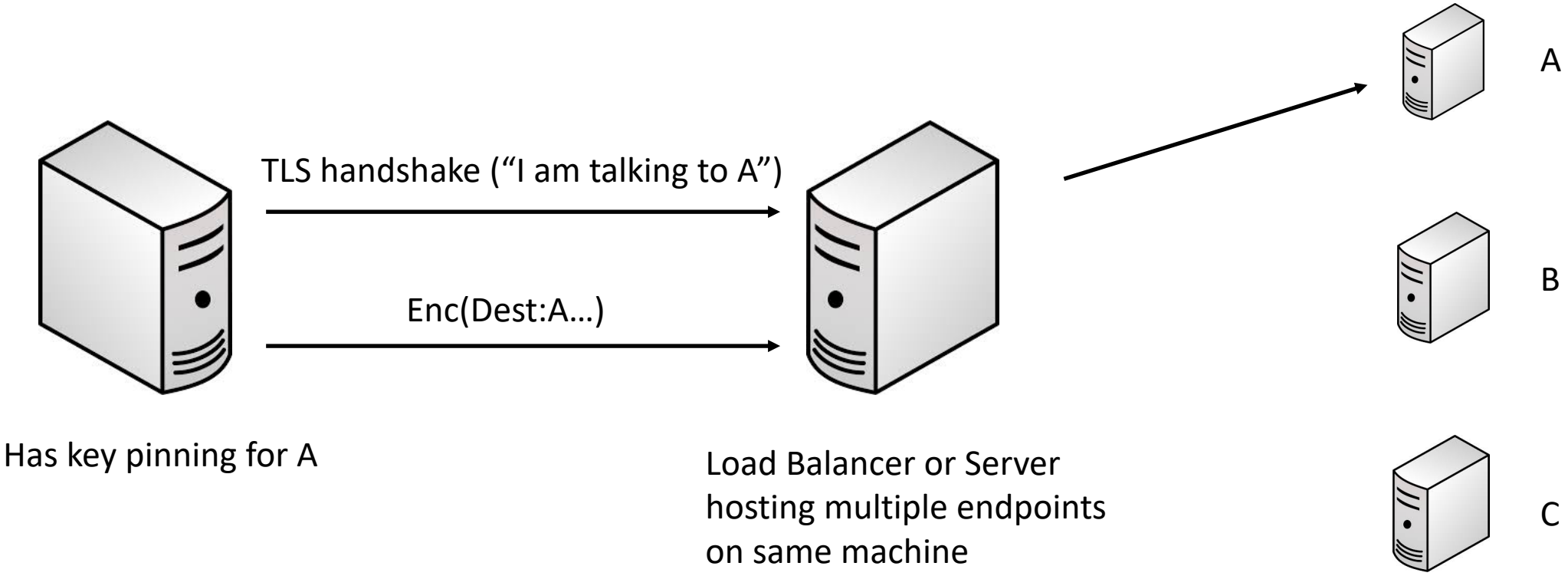
Hostname Selection

For TLS, SNI solves this by including the server id as part of the TLS negotiation



Hostname Selection

For TLS, SNI solves this by including the server id as part of the TLS negotiation



Hostname Selection

- Noise has a Prologue field that requires both parties to initialize the handshake with the same value

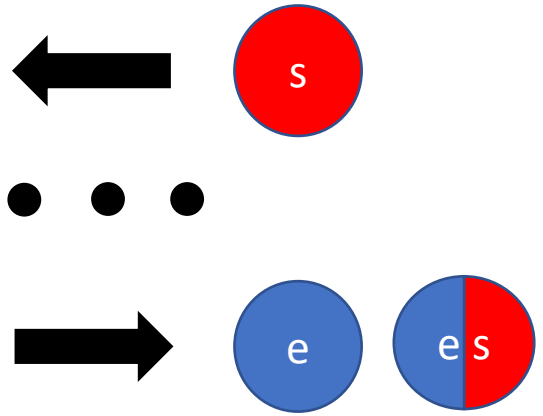
Hostname Selection

- Noise has a Prologue field that requires both parties to initialize the handshake with the same value

- We can use this field to support SNI

Hostname Selection

- Noise has a Prologue field that requires both parties to initialize the handshake with the same value
- We can use this field to support SNI
- If we need to hide this data from the forwarding server we can encrypt it using the N pattern:



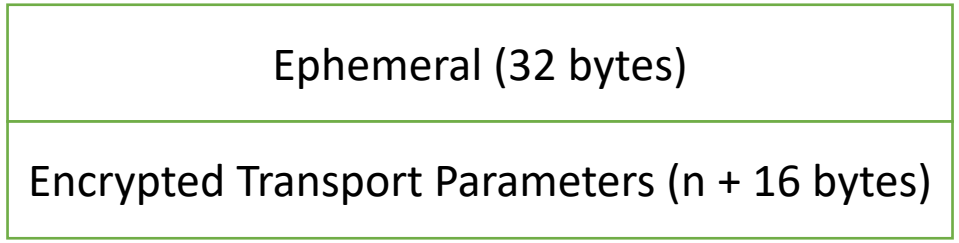
Handshake Request

- Indicates the start of the cryptographic handshake
- Transmits encrypted client transport parameters
- Optionally presents client’s Identity

Ephemeral (32 bytes)
Client Static (32 + 16 bytes)
Encrypted Transport Parameters (n + 16 bytes)

Handshake Response

- Completes Negotiation of Transport Keys
- Transmits encrypted server transport parameters
- Proves server’s identity



After this step, final keys are derived and passed to QUIC packet protector

Implicit Acknowledgement

- In order to avoid replay or Key Compromise attacks, client must immediately send a packet encrypted under transport keys upon completing the handshake

Implicit Acknowledgement

- In order to avoid replay or Key Compromise attacks, client must immediately send a packet encrypted under transport keys upon completing the handshake

- Only after this does the server consider the handshake finished

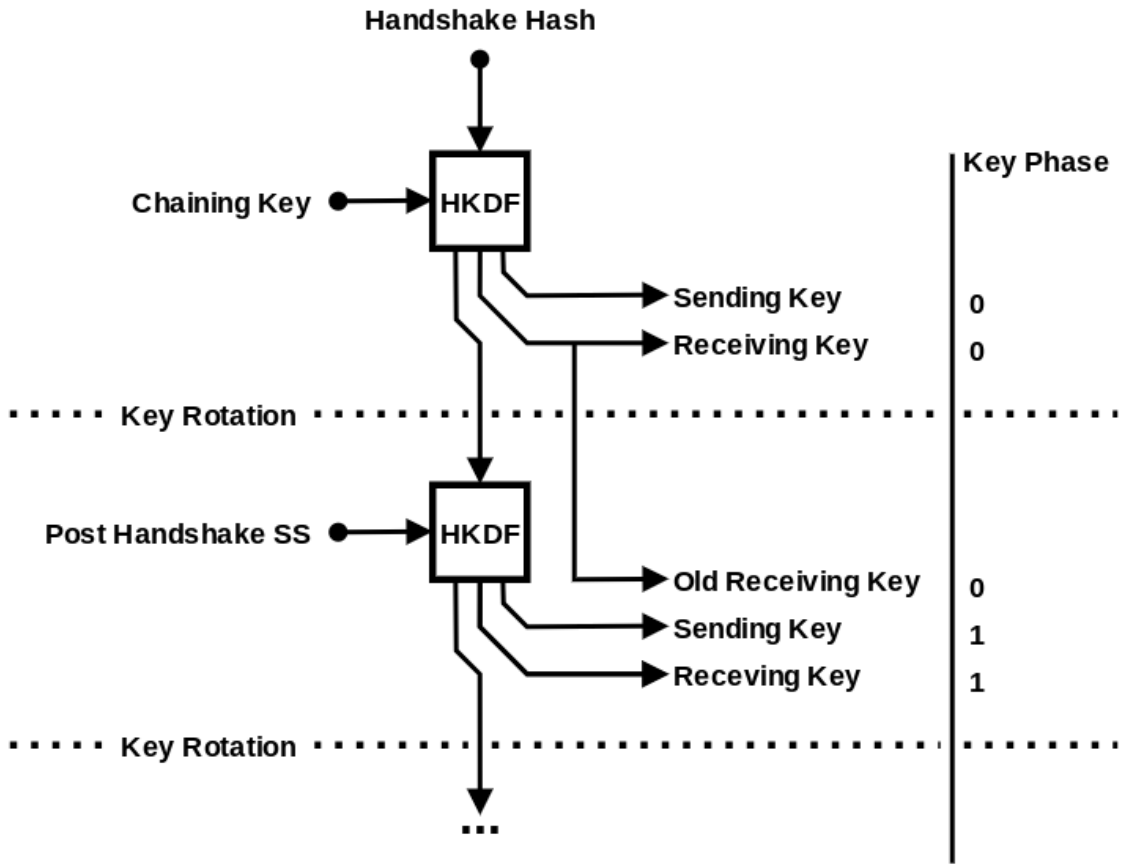
Implicit Acknowledgement

- In order to avoid replay or Key Compromise attacks, client must immediately send a packet encrypted under transport keys upon completing the handshake
- Only after this does the server consider the handshake finished
- If application data is not available a PADDING or PING frame is sent

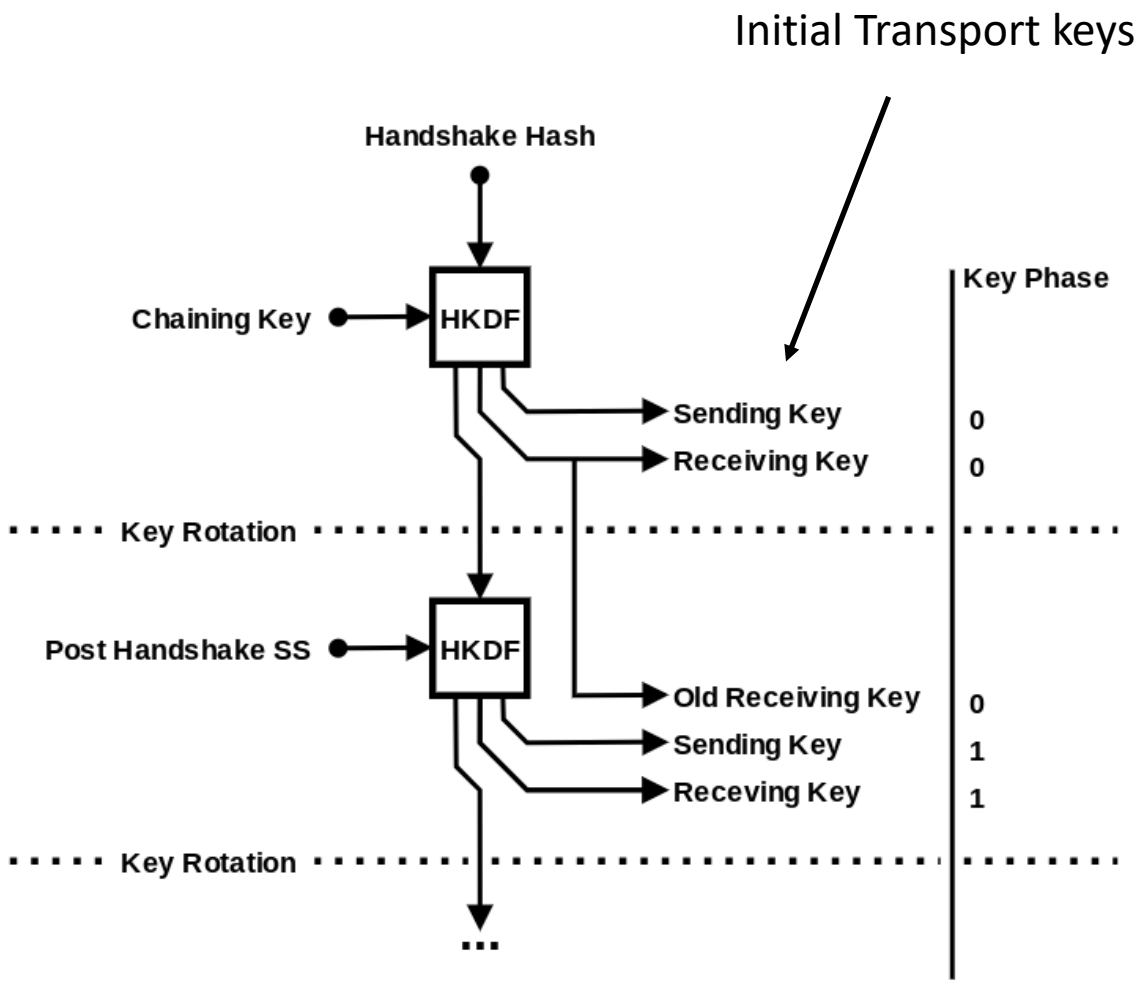
Implicit Acknowledgement

- In order to avoid replay or Key Compromise attacks, client must immediately send a packet encrypted under transport keys upon completing the handshake
- Only after this does the server consider the handshake finished
- If application data is not available a PADDING or PING frame is sent
- Clients ACK frames are not sufficient for this

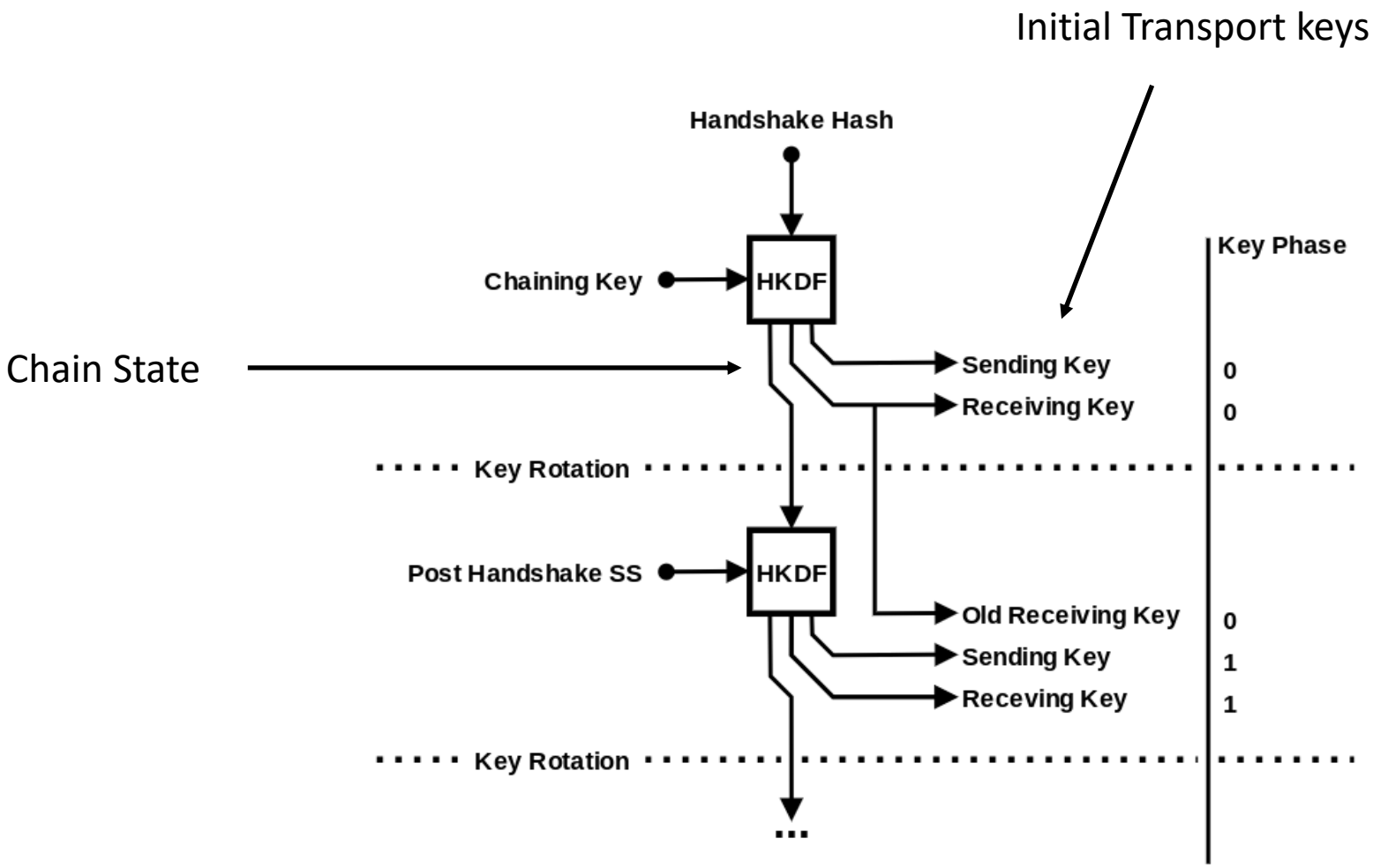
Ratcheting in nQUIC



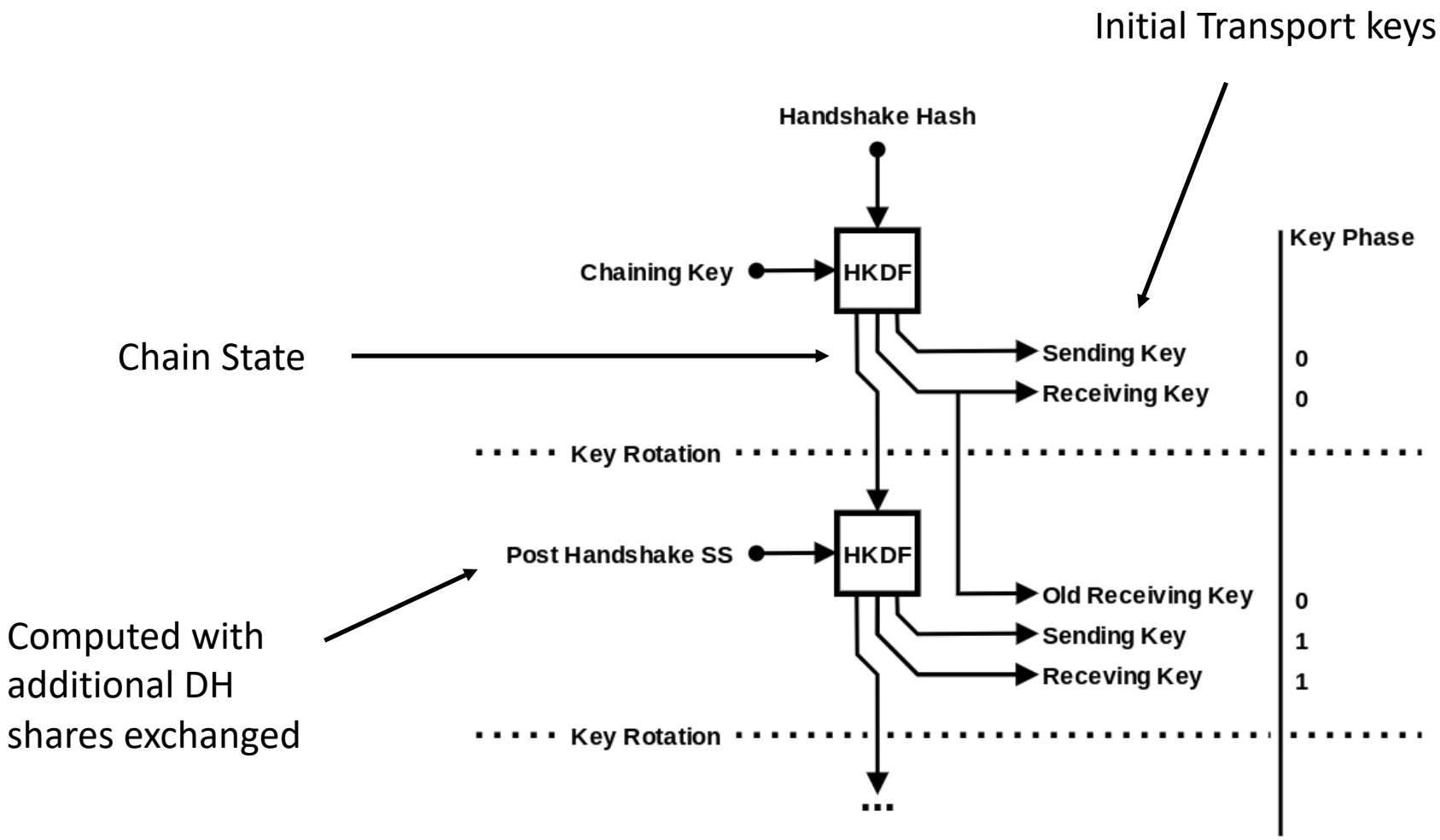
Ratcheting in nQUIC



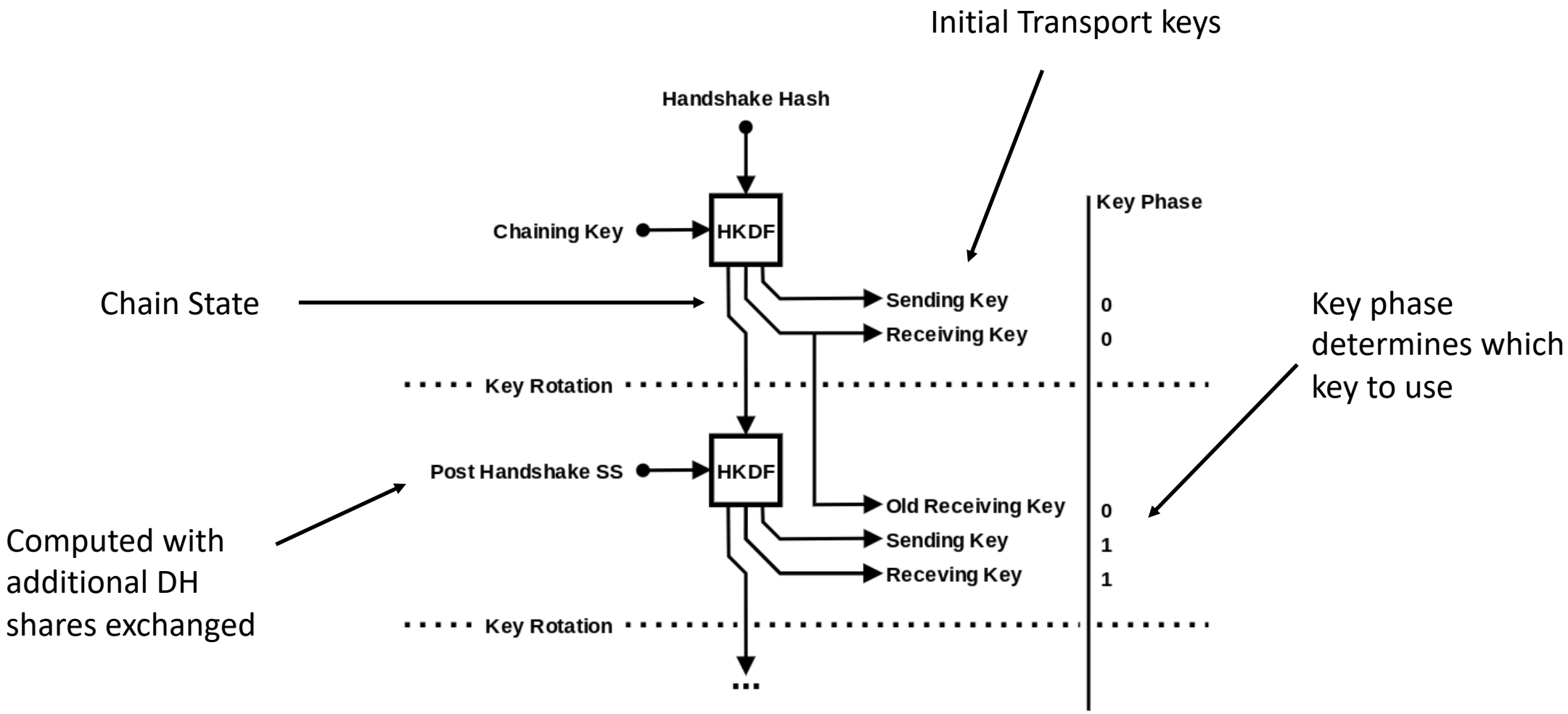
Ratcheting in nQUIC



Ratcheting in nQUIC



Ratcheting in nQUIC



Interoperability

- nQUIC is intended to be a minimal update to QUIC-TLS

Interoperability

- nQUIC is intended to be a minimal update to QUIC-TLS
- Potentially can be specified as a new QUIC version compatible with existing RFCs

Interoperability

- nQUIC is intended to be a minimal update to QUIC-TLS
- Potentially can be specified as a new QUIC version compatible with existing RFCs
- Since versions with the upper 16 bits clear are reserved for future IETF use we chose 0xff00000b for nQUIC

Cost Comparison

C_{key} be the cost of a key generation operation
C_{dh} be the cost of a key exchange operation
C_{sign} be the cost of a signing operation
C_{verif} be the cost of a signature verification operation

Figure 5: nQUIC Key Exchange Cost Comparison

Authentication	Client Cost	Server Cost
server	$1C_{key} + 4C_{dh}$	$1C_{key} + 4C_{dh}$
mutual	$1C_{key} + 4C_{dh}$	$1C_{key} + 4C_{dh}$

Figure 6: QUIC-TLS Key Exchange Cost Comparison

Authentication	Client Cost	Server Cost
server	$1C_{key} + 1C_{dh} + XC_{verif}$	$1C_{key} + 1C_{dh} + 1C_{sign}$
mutual	$1C_{key} + 1C_{dh} + XC_{verif} + 1C_{sign}$	$1C_{key} + 1C_{dh} + YC_{verif} + 1C_{sign}$
psk	None	None
psk_dhe	$1C_{key} + 2C_{dh}$	$1C_{key} + 2C_{dh}$

Performance Evaluation

The key difference in nQUIC and QUIC-TLS is the handshake

Performance Evaluation

The key difference in nQUIC and QUIC-TLS is the handshake

We created proof of concept implementations of nQUIC in Rust and Go

Performance Evaluation

The key difference in nQUIC and QUIC-TLS is the handshake

We created proof of concept implementations of nQUIC in Rust and Go

These libraries where significantly smaller than the QUIC libraries they were based on

Performance Evaluation

Implementation	Encryption	Hash	Handshake Time (s)	Handshake Bandwidth (B)
ninn (nQUIC)	AES-GCM-256	SHA-256	0.00135	1496
quinn (QUIC-TLS)	AES-GCM-256	SHA-384	0.00193	3426
quic-go (QUIC-TLS)	AES-GCM-256	SHA-384	0.02949	3230
quic-go (QUIC-TLS PSK)	AES-GCM-256	SHA-384	0.02689	2036
nquic-go (nQUIC)	AES-GCM-256	SHA-256	0.01023	1463

Conclusion

nQUIC offers the following advantages:

Conclusion

nQUIC offers the following advantages:

- Improved performance

Conclusion

nQUIC offers the following advantages:

- Improved performance
- Simple implementations

Conclusion

nQUIC offers the following advantages:

- Improved performance
- Simple implementations
- Clearer Security properties

Conclusion

nQUIC offers the following advantages:

- Improved performance
- Simple implementations
- Clearer Security properties

Supporting a variety of handshake protocols will help avoid ossification and allow for scenario specific optimizations

Questions?

- Thank you!