The post-quantum Internet

Daniel J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Includes joint work with:

Tanja Lange

Technische Universiteit Eindhoven

IP: Internet Protocol

IP communicates "packets": limited-length byte strings.

Each computer on the Internet has a 4-byte "IP address". e.g. `www.pqcrypto.org` has address `131.155.70.11`.

Your browser creates a packet addressed to `131.155.70.11`; gives packet to the Internet. Hopefully the Internet delivers that packet to `131.155.70.11`.

t-quantum Internet

. Bernstein

ty of Illinois at Chicago &
che Universiteit Eindhoven

joint work with:

ange

che Universiteit Eindhoven

IP: Internet Protocol

IP communicates “packets”: limited-length byte strings.

Each computer on the Internet has a 4-byte “IP address”. e.g. `www.pqcrypto.org` has address `131.155.70.11`.

Your browser creates a packet addressed to `131.155.70.11`; gives packet to the Internet. Hopefully the Internet delivers that packet to `131.155.70.11`.

DNS: D

You actu
connect

Browser
by askin
the `pqc`

Browser
“`Where`

n Internet

n

is at Chicago &
siteit Eindhoven

k with:

siteit Eindhoven

IP: Internet Protocol

IP communicates “packets”:
limited-length byte strings.

Each computer on the Internet
has a 4-byte “IP address”.
e.g. `www.pqcrypto.org` has
address `131.155.70.11`.

Your browser creates a packet
addressed to `131.155.70.11`;
gives packet to the Internet.
Hopefully the Internet delivers
that packet to `131.155.70.11`.

DNS: Domain Na

You actually told y
connect to `www.pq`

Browser learns “13
by asking a name
the `pqcrypto.org`

Browser → `131.1`
“`Where is www.pq`

ago &
hoven

hoven

IP: Internet Protocol

IP communicates "packets": limited-length byte strings.

Each computer on the Internet has a 4-byte "IP address". e.g. `www.pqcrypto.org` has address `131.155.70.11`.

Your browser creates a packet addressed to `131.155.70.11`; gives packet to the Internet. Hopefully the Internet delivers that packet to `131.155.70.11`.

DNS: Domain Name System

You actually told your brows
connect to `www.pqcrypto.`

Browser learns "`131.155.7`
by asking a name server,
the `pqcrypto.org` name se

Browser → `131.155.71.14`
"`Where is www.pqcrypto.`

IP: Internet Protocol

IP communicates "packets": limited-length byte strings.

Each computer on the Internet has a 4-byte "IP address". e.g. `www.pqcrypto.org` has address `131.155.70.11`.

Your browser creates a packet addressed to `131.155.70.11`; gives packet to the Internet. Hopefully the Internet delivers that packet to `131.155.70.11`.

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`: "`Where is www.pqcrypto.org?`"

IP: Internet Protocol

IP communicates "packets": limited-length byte strings.

Each computer on the Internet has a 4-byte "IP address". e.g. `www.pqcrypto.org` has address `131.155.70.11`.

Your browser creates a packet addressed to `131.155.70.11`; gives packet to the Internet. Hopefully the Internet delivers that packet to `131.155.70.11`.

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`: "`Where is www.pqcrypto.org?`"

IP packet from browser also includes a return address: the address of your computer.

`131.155.71.143` → browser: "`131.155.70.11`" rnet Protocol

nunicates “packets”:
ength byte strings.

mputer on the Internet
byte “IP address”.
.pqcrypto.org has
131.155.70.11.

owser creates a packet
ed to 131.155.70.11;
cket to the Internet.
y the Internet delivers
ket to 131.155.70.11.

DNS: Domain Name System

You actually told your browser to connect to www.pqcrypto.org.

Browser learns “131.155.70.11” by asking a name server, the pqcrypto.org name server.

Browser → 131.155.71.143:
“Where is www.pqcrypto.org?”

IP packet from browser also includes a return address: the address of your computer.

131.155.71.143 → browser:
“131.155.70.11”

Browser
address,
by askin

Browser
“Where

199.19.
“Ask th
name se

col

"packets":
e strings.

the Internet
ddress".
o.org has
70.11.

tes a packet
155.70.11;
e Internet.
rnet delivers
1.155.70.11.

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`: "`Where is www.pqcrypto.org?`"

IP packet from browser also includes a return address: the address of your computer.

`131.155.71.143` → browser: "`131.155.70.11`"

Browser learns the
address, "131.155
by asking the .org

Browser → 199.1
"Where is www.p

199.19.54.1 →
"Ask the pqcrypt
name server, 13

t

net

s

et

.1;

ers

.11.

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`: "`Where is www.pqcrypto.org?`"

IP packet from browser also includes a return address: the address of your computer.

`131.155.71.143` → browser: "`131.155.70.11`"

Browser learns the name-ser
address, "`131.155.71.143`'
by asking the `.org` name se

Browser → `199.19.54.1`:
"`Where is www.pqcrypto.`

`199.19.54.1` → browser:
"`Ask the pqcrypto.org`
`name server, 131.155.71`

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`: "`Where is www.pqcrypto.org?`"

IP packet from browser also includes a return address: the address of your computer.

`131.155.71.143` → browser: "`131.155.70.11`"

Browser learns the name-server address, "`131.155.71.143`", by asking the `.org` name server.

Browser → `199.19.54.1`: "`Where is www.pqcrypto.org?`"

`199.19.54.1` → browser: "`Ask the pqcrypto.org name server, 131.155.71.143`"

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`:
"`Where is www.pqcrypto.org?`"

IP packet from browser also includes a return address: the address of your computer.

`131.155.71.143` → browser:
"`131.155.70.11`"

Browser learns the name-server address, "`131.155.71.143`", by asking the `.org` name server.

Browser → `199.19.54.1`:
"`Where is www.pqcrypto.org?`"

`199.19.54.1` → browser:
"`Ask the pqcrypto.org name server, 131.155.71.143`"

Browser learns "`199.19.54.1`", the `.org` server address, by asking the root name server.

DNS: Domain Name System

You actually told your browser to connect to `www.pqcrypto.org`.

Browser learns "`131.155.70.11`" by asking a name server, the `pqcrypto.org` name server.

Browser → `131.155.71.143`: "`Where is www.pqcrypto.org?`"

IP packet from browser also includes a return address: the address of your computer.

`131.155.71.143` → browser: "`131.155.70.11`"

Browser learns the name-server address, "`131.155.71.143`", by asking the `.org` name server.

Browser → `199.19.54.1`: "`Where is www.pqcrypto.org?`"

`199.19.54.1` → browser: "`Ask the pqcrypto.org name server, 131.155.71.143`"

Browser learns "`199.19.54.1`", the `.org` server address, by asking the root name server.

Browser learned root address by consulting the Bible.

omain Name System

ually told your browser to
to www.pqcrypto.org.

learns "131.155.70.11"
g a name server,
rypto.org name server.

→ 131.155.71.143:
is www.pqcrypto.org?"

et from browser also
a return address:
ess of your computer.

5.71.143 → browser:
55.70.11"

Browser learns the name-server address, "131.155.71.143", by asking the .org name server.

Browser → 199.19.54.1:
"Where is www.pqcrypto.org?"

199.19.54.1 → browser:
"Ask the pqcrypto.org name server, 131.155.71.143"

Browser learns "199.19.54.1", the .org server address, by asking the root name server.

Browser learned root address by consulting the Bible.

TCP: Tr

Packets

(Actually
Oldest I
≥576. U
often 15

me System

your browser to
qcrypto.org.

31.155.70.11"
server,
g name server.

55.71.143:
qcrypto.org?"

owser also
address:
r computer.

→ browser:
,

Browser learns the name-server
address, "131.155.71.143",
by asking the .org name server.

Browser → 199.19.54.1:
"Where is www.pqcrypto.org?"

199.19.54.1 → browser:
"Ask the pqcrypto.org
name server, 131.155.71.143"

Browser learns "199.19.54.1",
the .org server address,
by asking the root name server.

Browser learned root address
by consulting the Bible.

TCP: Transmissio

Packets are limite

(Actually depends
Oldest IP standar
≥576. Usually 14
often 1500, somet

n

ser to
org.

0.11"

rver.

3:
org?"

er.

er:

Browser learns the name-server address, "131.155.71.143", by asking the .org name server.

Browser → 199.19.54.1:
"Where is www.pqcrypto.org?"

199.19.54.1 → browser:
"Ask the pqcrypto.org name server, 131.155.71.143"

Browser learns "199.19.54.1", the .org server address, by asking the root name server.

Browser learned root address by consulting the Bible.

TCP: Transmission Control

Packets are limited to 1280

(Actually depends on netwo
Oldest IP standards require
≥576. Usually 1492 is safe,
often 1500, sometimes more

Browser learns the name-server address, "`131.155.71.143`", by asking the `.org` name server.

Browser → `199.19.54.1`: "`Where is www.pqcrypto.org?`"

`199.19.54.1` → browser: "`Ask the pqcrypto.org name server, 131.155.71.143`"

Browser learns "`199.19.54.1`", the `.org` server address, by asking the root name server.

Browser learned root address by consulting the Bible.

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

Browser learns the name-server address, "`131.155.71.143`", by asking the `.org` name server.

Browser → `199.19.54.1`: "`Where is www.pqcrypto.org?`"

`199.19.54.1` → browser: "`Ask the pqcrypto.org name server, 131.155.71.143`"

Browser learns "`199.19.54.1`", the `.org` server address, by asking the root name server.

Browser learned root address by consulting the Bible.

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

The page you're downloading from `pqcrypto.org` doesn't fit.

Browser learns the name-server address, "131.155.71.143", by asking the .org name server.

Browser → 199.19.54.1: "Where is www.pqcrypto.org?"

199.19.54.1 → browser: "Ask the pqcrypto.org name server, 131.155.71.143"

Browser learns "199.19.54.1", the .org server address, by asking the root name server.

Browser learned root address by consulting the Bible.

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

The page you're downloading from pqcrypto.org doesn't fit.

Browser actually makes "TCP connection" to pqcrypto.org. Inside that connection: sends HTTP request, receives response.

learns the name-server
"131.155.71.143",
g the .org name server.

→ 199.19.54.1:
is www.pqcrypto.org?"

.54.1 → browser:
e pqcrypto.org
rver, 131.155.71.143"

learns "199.19.54.1",
g server address,
g the root name server.

learned root address
ulting the Bible.

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

The page you're downloading from pqcrypto.org doesn't fit.

Browser actually makes "TCP connection" to pqcrypto.org. Inside that connection: sends HTTP request, receives response.

Browser
"SYN 16

Server →
"ACK 16

Browser
"ACK 74

Server n
for this

Browser
counting

Server sp
counting

e name-server
5.71.143",
g name server.

9.54.1:
qcrypto.org?"

browser:
to.org
1.155.71.143"

99.19.54.1",
ddress,
name server.

oot address
Bible.

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

The page you're downloading from pqcrypto.org doesn't fit.

Browser actually makes "TCP connection" to pqcrypto.org. Inside that connection: sends HTTP request, receives response.

Browser → server:
"SYN 168bb5d9"

Server → browser:
"ACK 168bb5da,

Browser → server:
"ACK 747bfa42"

Server now allocat
for this TCP conn

Browser splits dat
counting bytes fro

Server splits data
counting bytes fro

ver
”,
rver.

org?”

.143”

.1”,

ver.

s

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

The page you're downloading from `pqcrypto.org` doesn't fit.

Browser actually makes "TCP connection" to `pqcrypto.org`. Inside that connection: sends HTTP request, receives response.

Browser → server:
"`SYN 168bb5d9`"

Server → browser:
"`ACK 168bb5da, SYN 747bf`

Browser → server:
"`ACK 747bfa42`"

Server now allocates buffers for this TCP connection.

Browser splits data into pac
counting bytes from `168bb5`

Server splits data into packe
counting bytes from `747bfa`

TCP: Transmission Control Protocol

Packets are limited to 1280 bytes.

(Actually depends on network. Oldest IP standards required ≥576. Usually 1492 is safe, often 1500, sometimes more.)

The page you're downloading from `pqcrypto.org` doesn't fit.

Browser actually makes "TCP connection" to `pqcrypto.org`. Inside that connection: sends HTTP request, receives response.

Browser → server:
"`SYN 168bb5d9`"

Server → browser:
"`ACK 168bb5da, SYN 747bfa41`"

Browser → server:
"`ACK 747bfa42`"

Server now allocates buffers for this TCP connection.

Browser splits data into packets, counting bytes from `168bb5da`.

Server splits data into packets, counting bytes from `747bfa42`.

ransmission Control Protocol

are limited to 1280 bytes.

y depends on network.
P standards required
Jsually 1492 is safe,
00, sometimes more.)

e you're downloading
crypto.org doesn't fit.

actually makes "TCP
on" to pqcrypto.org.
hat connection: sends
equest, receives response.

Browser → server:
"SYN 168bb5d9"

Server → browser:
"ACK 168bb5da, SYN 747bfa41"

Browser → server:
"ACK 747bfa42"

Server now allocates buffers
for this TCP connection.

Browser splits data into packets,
counting bytes from 168bb5da.

Server splits data into packets,
counting bytes from 747bfa42.

Main fea
"reliable

Internet
or delive
Doesn't
compute
inside ea

Comput
if data is
Complica
retransm
avoiding

n Control Protocol

d to 1280 bytes.

on network.
ds required
92 is safe,
imes more.)

ownloading
rg doesn't fit.

makes “TCP
crypto.org.
tion: sends
ceives response.

Browser → server:
“SYN 168bb5d9”

Server → browser:
“ACK 168bb5da, SYN 747bfa41”

Browser → server:
“ACK 747bfa42”

Server now allocates buffers
for this TCP connection.

Browser splits data into packets,
counting bytes from 168bb5da.

Server splits data into packets,
counting bytes from 747bfa42.

Main feature adve
“reliable data stre

Internet sometime
or delivers packets
Doesn't confuse T
computer checks t
inside each TCP p

Computer retransm
if data is not ackn
Complicated rules
retransmission sch
avoiding network

Protocol

bytes.

rk.

d

e.)

g

t fit.

CP

rg.

ls

ponse.

Browser → server:
"SYN 168bb5d9"

Server → browser:
"ACK 168bb5da, SYN 747bfa41"

Browser → server:
"ACK 747bfa42"

Server now allocates buffers for this TCP connection.

Browser splits data into packets, counting bytes from 168bb5da.

Server splits data into packets, counting bytes from 747bfa42.

Main feature advertised by T
"reliable data streams".

Internet sometimes loses pac
or delivers packets out of or
Doesn't confuse TCP conne
computer checks the counte
inside each TCP packet.

Computer retransmits data
if data is not acknowledged.
Complicated rules to decide
retransmission schedule,
avoiding network congestion

Browser → server:
"`SYN 168bb5d9`"

Server → browser:
"`ACK 168bb5da, SYN 747bfa41`"

Browser → server:
"`ACK 747bfa42`"

Server now allocates buffers for this TCP connection.

Browser splits data into packets, counting bytes from `168bb5da`.

Server splits data into packets, counting bytes from `747bfa42`.

Main feature advertised by TCP: "reliable data streams".

Internet sometimes loses packets or delivers packets out of order. Doesn't confuse TCP connections: computer checks the counter inside each TCP packet.

Computer retransmits data if data is not acknowledged. Complicated rules to decide retransmission schedule, avoiding network congestion.

→ server:
8bb5d9"

→ browser:
8bb5da, SYN 747bfa41"

→ server:
7bfa42"

ow allocates buffers
TCP connection.

splits data into packets,
bytes from 168bb5da.

plits data into packets,
bytes from 747bfa42.

Main feature advertised by TCP:
"reliable data streams".

Internet sometimes loses packets
or delivers packets out of order.
Doesn't confuse TCP connections:
computer checks the counter
inside each TCP packet.

Computer retransmits data
if data is not acknowledged.
Complicated rules to decide
retransmission schedule,
avoiding network congestion.

Stream-

http://
uses HT

https:/
uses HT

Your bro
- finds a
- makes
- inside
  builds
  by exc
- inside
  sends

:

:

SYN 747bfa41”

:

tes buffers
ection.

a into packets,
m 168bb5da.

into packets,
m 747bfa42.

Main feature advertised by TCP: “reliable data streams”.

Internet sometimes loses packets or delivers packets out of order. Doesn’t confuse TCP connections: computer checks the counter inside each TCP packet.

Computer retransmits data if data is not acknowledged. Complicated rules to decide retransmission schedule, avoiding network congestion.

Stream-level crypt

http://www.pqcr
uses HTTP over T

https://www.pqc
uses HTTP over T

Your browser

- finds address 13
- makes TCP con
- inside the TCP c
  builds a TLS co
  by exchanging c
- inside the TLS c
  sends HTTP rec

fa41"

kets,
da.

ts,
42.

Main feature advertised by TCP: "reliable data streams".

Internet sometimes loses packets or delivers packets out of order. Doesn't confuse TCP connections: computer checks the counter inside each TCP packet.

Computer retransmits data if data is not acknowledged. Complicated rules to decide retransmission schedule, avoiding network congestion.

Stream-level crypto

http://www.pqcrypto.org
uses HTTP over TCP.

https://www.pqcrypto.or
uses HTTP over TLS over T

Your browser

- finds address 131.155.70
- makes TCP connection;
- inside the TCP connectio builds a TLS connection by exchanging crypto keys
- inside the TLS connection sends HTTP request etc.

Main feature advertised by TCP: "reliable data streams".

Internet sometimes loses packets or delivers packets out of order. Doesn't confuse TCP connections: computer checks the counter inside each TCP packet.

Computer retransmits data if data is not acknowledged. Complicated rules to decide retransmission schedule, avoiding network congestion.

Stream-level crypto

`http://www.pqcrypto.org` uses HTTP over TCP.

`https://www.pqcrypto.org` uses HTTP over TLS over TCP.

Your browser

- finds address `131.155.70.11`;
- makes TCP connection;
- inside the TCP connection, builds a TLS connection by exchanging crypto keys;
- inside the TLS connection, sends HTTP request etc.

ature advertised by TCP:
data streams".

sometimes loses packets
ers packets out of order.
confuse TCP connections:
er checks the counter
ach TCP packet.

er retransmits data
s not acknowledged.
ated rules to decide
nission schedule,
network congestion.

Stream-level crypto

http://www.pqcrypto.org
uses HTTP over TCP.

https://www.pqcrypto.org
uses HTTP over TLS over TCP.

Your browser

- finds address 131.155.70.11;
- makes TCP connection;
- inside the TCP connection, builds a TLS connection by exchanging crypto keys;
- inside the TLS connection, sends HTTP request etc.

What ha
forges a
pointing
Or a TC
with bog

DNS sof
TCP sof
TLS soft
somethi
but has

Browser
make a
but this
Huge da

rtised by TCP:
ams".

s loses packets
; out of order.
TCP connections:
the counter
packet.

mits data
owledged.
to decide
edule,
congestion.

Stream-level crypto

http://www.pqcrypto.org
uses HTTP over TCP.

https://www.pqcrypto.org
uses HTTP over TLS over TCP.

Your browser

- finds address 131.155.70.11;
- makes TCP connection;
- inside the TCP connection, builds a TLS connection by exchanging crypto keys;
- inside the TLS connection, sends HTTP request etc.

What happens if a
forges a DNS pack
pointing to fake se
Or a TCP packet
with bogus data?

DNS software is fo
TCP software is fo
TLS software sees
something has gon
but has no way to

Browser using TLS
make a whole new
but this is slow an
Huge damage fron

TCP:

ckets
der.
ctions:
r

.

Stream-level crypto

http://www.pqcrypto.org
uses HTTP over TCP.

https://www.pqcrypto.org
uses HTTP over TLS over TCP.

Your browser

- finds address 131.155.70.11;
- makes TCP connection;
- inside the TCP connection, builds a TLS connection by exchanging crypto keys;
- inside the TLS connection, sends HTTP request etc.

What happens if attacker forges a DNS packet pointing to fake server? Or a TCP packet with bogus data?

DNS software is fooled. TCP software is fooled. TLS software sees that something has gone wrong, but has no way to recover.

Browser using TLS can make a whole new connectio but this is slow and fragile. Huge damage from forged p

Stream-level crypto

http://www.pqcrypto.org
uses HTTP over TCP.

https://www.pqcrypto.org
uses HTTP over TLS over TCP.

Your browser

- finds address 131.155.70.11;
- makes TCP connection;
- inside the TCP connection, builds a TLS connection by exchanging crypto keys;
- inside the TLS connection, sends HTTP request etc.

What happens if attacker forges a DNS packet pointing to fake server? Or a TCP packet with bogus data?

DNS software is fooled. TCP software is fooled. TLS software sees that something has gone wrong, but has no way to recover.

Browser using TLS can make a whole new connection, but this is slow and fragile. Huge damage from forged packet.

level crypto

/www.pqcrypto.org
TP over TCP.

//www.pqcrypto.org
TP over TLS over TCP.

owser
address 131.155.70.11;
TCP connection;
the TCP connection,
a TLS connection
hanging crypto keys;
the TLS connection,
HTTP request etc.

What happens if attacker forges a DNS packet pointing to fake server? Or a TCP packet with bogus data?

DNS software is fooled. TCP software is fooled. TLS software sees that something has gone wrong, but has no way to recover.

Browser using TLS can make a whole new connection, but this is slow and fragile. Huge damage from forged packet.

Modern
CurveCP
Google's
encrypt

Discard
immedia
Retransm
authenti

o

rypto.org
TCP.

crypto.org
TLS over TCP.

1.155.70.11;
nection;
connection,
nnection
rypto keys;
connection,
quest etc.

What happens if attacker
forges a DNS packet
pointing to fake server?
Or a TCP packet
with bogus data?

DNS software is fooled.
TCP software is fooled.
TLS software sees that
something has gone wrong,
but has no way to recover.

Browser using TLS can
make a whole new connection,
but this is slow and fragile.
Huge damage from forged packet.

Modern trend (e.g
CurveCP; see also
Google's QUIC): A
encrypt each pack

Discard forged pac
immediately: no d
Retransmit packet
authenticated ack

g

rg

TCP.

.11;

n,

;

,

What happens if attacker
forges a DNS packet
pointing to fake server?
Or a TCP packet
with bogus data?

DNS software is fooled.
TCP software is fooled.
TLS software sees that
something has gone wrong,
but has no way to recover.

Browser using TLS can
make a whole new connection,
but this is slow and fragile.
Huge damage from forged packet.

Modern trend (e.g., DNSCu
CurveCP; see also MinimalT
Google's QUIC): Authentica
encrypt each packet separat

Discard forged packet
immediately: no damage.
Retransmit packet if no
authenticated acknowledgm

What happens if attacker forges a DNS packet pointing to fake server? Or a TCP packet with bogus data?

DNS software is fooled. TCP software is fooled. TLS software sees that something has gone wrong, but has no way to recover.

Browser using TLS can make a whole new connection, but this is slow and fragile. Huge damage from forged packet.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

What happens if attacker forges a DNS packet pointing to fake server? Or a TCP packet with bogus data?

DNS software is fooled. TCP software is fooled. TLS software sees that something has gone wrong, but has no way to recover.

Browser using TLS can make a whole new connection, but this is slow and fragile. Huge damage from forged packet.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

What happens if attacker forges a DNS packet pointing to fake server? Or a TCP packet with bogus data?

DNS software is fooled. TCP software is fooled. TLS software sees that something has gone wrong, but has no way to recover.

Browser using TLS can make a whole new connection, but this is slow and fragile. Huge damage from forged packet.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

appens if attacker
DNS packet
to fake server?
CP packet
gus data?

ftware is fooled.
ftware is fooled.
tware sees that
ng has gone wrong,
no way to recover.

using TLS can
whole new connection,
is slow and fragile.
mage from forged packet.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

The KE

Original
Message
as m^e m

ttacker
ket
erver?

ooled.
ooled.
that
ne wrong,
recover.

S can
connection,
d fragile.
n forged packet.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

The KEM+AE ph

Original view of R
Message m is encr
as m^e mod pq.

on,

acket.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

The KEM+AE philosophy

Original view of RSA: Message m is encrypted as $m^e \bmod pq$.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

The KEM+AE philosophy

Original view of RSA: Message m is encrypted as $m^e \bmod pq$.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

The KEM+AE philosophy

Original view of RSA: Message m is encrypted as $m^e \bmod pq$.

"Hybrid" view of RSA, including random padding:
Choose random AES-GCM key k.
Randomly pad k as r.
Encrypt r as $r^e \bmod pq$.
Encrypt m under k.

Modern trend (e.g., DNSCurve, CurveCP; see also MinimaLT, Google's QUIC): Authenticate and encrypt each packet separately.

Discard forged packet immediately: no damage. Retransmit packet if no *authenticated* acknowledgment.

Engineering advantage: Packet-level crypto works for more protocols than stream-level crypto.

Disadvantage: Crypto must fit into packet.

The KEM+AE philosophy

Original view of RSA: Message m is encrypted as $m^e \bmod pq$.

"Hybrid" view of RSA, including random padding: Choose random AES-GCM key k. Randomly pad k as r. Encrypt r as $r^e \bmod pq$. Encrypt m under k.

Fragile, many problems: e.g., Coppersmith attack, Bleichenbacher attack, bogus OAEP security proof.

trend (e.g., DNSCurve,
P; see also MinimaLT,
s QUIC): Authenticate and
each packet separately.

forged packet
tely: no damage.
mit packet if no
ficated acknowledgment.

ring advantage:
evel crypto
r more protocols
eam-level crypto.

ntage:
must fit into packet.

The KEM+AE philosophy

Original view of RSA:
Message m is encrypted
as $m^e \bmod pq$.

"Hybrid" view of RSA,
including random padding:
Choose random AES-GCM key k.
Randomly pad k as r.
Encrypt r as $r^e \bmod pq$.
Encrypt m under k.

Fragile, many problems:
e.g., Coppersmith attack,
Bleichenbacher attack,
bogus OAEP security proof.

Shoup's

"Key en
Choose
Encrypt
Define k

"Data e
Encrypt
m under

Authent
any mod

Much ea
Also gen
Can mix

g., DNSCurve,
MinimaLT,
Authenticate and
et separately.

cket
amage.
if no
nowledgment.

tage:
o
otocols
crypto.

to packet.

The KEM+AE philosophy

Original view of RSA:
Message m is encrypted
as $m^e \bmod pq$.

"Hybrid" view of RSA,
including random padding:
Choose random AES-GCM key k.
Randomly pad k as r.
Encrypt r as $r^e \bmod pq$.
Encrypt m under k.

Fragile, many problems:
e.g., Coppersmith attack,
Bleichenbacher attack,
bogus OAEP security proof.

Shoup's "KEM+D

"Key encapsulatio
Choose random r
Encrypt r as r^e m
Define $k = H(r, r^e$

"Data encapsulati
Encrypt and authe
m under AES-GCM

Authenticator catc
any modification c

Much easier to get
Also generalizes ni
Can mix multiple l

rve,

T,

te and

ely.

ent.

The KEM+AE philosophy

Original view of RSA:
Message m is encrypted
as m^e mod pq.

"Hybrid" view of RSA,
including random padding:
Choose random AES-GCM key k.
Randomly pad k as r.
Encrypt r as r^e mod pq.
Encrypt m under k.

Fragile, many problems:
e.g., Coppersmith attack,
Bleichenbacher attack,
bogus OAEP security proof.

Shoup's "KEM+DEM" view

"Key encapsulation mechani
Choose random r mod pq.
Encrypt r as r^e mod pq.
Define $k = H(r, r^e \text{ mod } pq)$

"Data encapsulation mechan
Encrypt and authenticate
m under AES-GCM key k.

Authenticator catches
any modification of r^e mod

Much easier to get right.
Also generalizes nicely.
Can mix multiple hashes.

The KEM+AE philosophy

Original view of RSA:
Message m is encrypted as $m^e \bmod pq$.

"Hybrid" view of RSA, including random padding:
Choose random AES-GCM key k.
Randomly pad k as r.
Encrypt r as $r^e \bmod pq$.
Encrypt m under k.

Fragile, many problems:
e.g., Coppersmith attack, Bleichenbacher attack, bogus OAEP security proof.

Shoup's "KEM+DEM" view:

"Key encapsulation mechanism":
Choose random $r \bmod pq$.
Encrypt r as $r^e \bmod pq$.
Define $k = H(r, r^e \bmod pq)$.

"Data encapsulation mechanism":
Encrypt and authenticate m under AES-GCM key k.

Authenticator catches any modification of $r^e \bmod pq$.

Much easier to get right.
Also generalizes nicely.
Can mix multiple hashes.

M+AE philosophy

view of RSA:

m is encrypted

od pq.

' view of RSA,

g random padding:

random AES-GCM key k.

ly pad k as r.

r as r^e mod pq.

m under k.

many problems:

ppersmith attack,

bacher attack,

AEP security proof.

Shoup's "KEM+DEM" view:

"Key encapsulation mechanism":
Choose random r mod pq.
Encrypt r as r^e mod pq.
Define $k = H(r, r^e \bmod pq)$.

"Data encapsulation mechanism":
Encrypt and authenticate
m under AES-GCM key k.

Authenticator catches
any modification of r^e mod pq.

Much easier to get right.
Also generalizes nicely.
Can mix multiple hashes.

DEM se
weak sin
of securi
authenti

Chou: Is
for mult

Answer:
KEM+A
(But nee
AES-GC
aim for f

More co
Use KEM
n-time s

ilosophy

SA:
ypted

RSA,
padding:
ES-GCM key k.
s r.
od pq.
k.

lems:
attack,
tack,
rity proof.

Shoup's "KEM+DEM" view:

"Key encapsulation mechanism":
Choose random r mod pq.
Encrypt r as r^e mod pq.
Define $k = H(r, r^e \bmod pq)$.

"Data encapsulation mechanism":
Encrypt and authenticate
m under AES-GCM key k.

Authenticator catches
any modification of r^e mod pq.

Much easier to get right.
Also generalizes nicely.
Can mix multiple hashes.

DEM security hyp
weak single-messa
of security for secr
authenticated encr

Chou: Is it safe to
for multiple messa

Answer: KEM+AE
KEM+AE $\Rightarrow$ KEM
(But need literatu
AES-GCM, Salsa2
aim for full AE sec

More complicated
Use KEM+DEM t
n-time secret key

key k.

Shoup's "KEM+DEM" view:

"Key encapsulation mechanism":
Choose random r mod pq.
Encrypt r as r^e mod pq.
Define $k = H(r, r^e \bmod pq)$.

"Data encapsulation mechanism":
Encrypt and authenticate
m under AES-GCM key k.

Authenticator catches
any modification of r^e mod pq.

Much easier to get right.
Also generalizes nicely.
Can mix multiple hashes.

DEM security hypothesis:
weak single-message version
of security for secret-key
authenticated encryption.

Chou: Is it safe to reuse k
for multiple messages?

Answer: KEM+AE is safe;
KEM+AE $\Rightarrow$ KEM+"nDEM
(But need literature on this!
AES-GCM, Salsa20-Poly130
aim for full AE security goal

More complicated alternativ
Use KEM+DEM to encrypt
n-time secret key m; reuse r

Shoup's "KEM+DEM" view:

"Key encapsulation mechanism":
Choose random r mod pq.
Encrypt r as r^e mod pq.
Define $k = H(r, r^e \bmod pq)$.

"Data encapsulation mechanism":
Encrypt and authenticate
m under AES-GCM key k.

Authenticator catches
any modification of r^e mod pq.

Much easier to get right.
Also generalizes nicely.
Can mix multiple hashes.

DEM security hypothesis:
weak single-message version
of security for secret-key
authenticated encryption.

Chou: Is it safe to reuse k
for multiple messages?

Answer: KEM+AE is safe;
KEM+AE $\Rightarrow$ KEM+"nDEM".
(But need literature on this!)
AES-GCM, Salsa20-Poly1305, etc.
aim for full AE security goal.

More complicated alternative:
Use KEM+DEM to encrypt an
n-time secret key m; reuse m.

"KEM+DEM" view:

capsulation mechanism":
random r mod pq.
r as r^e mod pq.
$= H(r, r^e \bmod pq)$.

ncapsulation mechanism":
and authenticate
AES-GCM key k.

icator catches
dification of r^e mod pq.

asier to get right.
neralizes nicely.
multiple hashes.

DEM security hypothesis:
weak single-message version
of security for secret-key
authenticated encryption.

Chou: Is it safe to reuse k
for multiple messages?

Answer: KEM+AE is safe;
KEM+AE $\Rightarrow$ KEM+"nDEM".
(But need literature on this!)
AES-GCM, Salsa20-Poly1305, etc.
aim for full AE security goal.

More complicated alternative:
Use KEM+DEM to encrypt an
n-time secret key m; reuse m.

DNSCur

Server k

Client kn
server's

Client —
packet c
where k
E is aut
q is DNS

Server —
packet c
where r

EM" view:

n mechanism":

mod pq.

od pq.

^{e} mod pq).

on mechanism":

enticate

M key k.

ches

of r^e mod pq.

t right.

icely.

hashes.

DEM security hypothesis:
weak single-message version
of security for secret-key
authenticated encryption.

Chou: Is it safe to reuse k
for multiple messages?

Answer: KEM+AE is safe;
KEM+AE $\Rightarrow$ KEM+"nDEM".
(But need literature on this!)
AES-GCM, Salsa20-Poly1305, etc.
aim for full AE security goal.

More complicated alternative:
Use KEM+DEM to encrypt an
n-time secret key m; reuse m.

DNSCurve: ECDH

Server knows ECD

Client knows ECD
server's public key

Client $\rightarrow$ server:
packet containing
where $k = H(cS)$;
E is authenticated
q is DNS query.

Server $\rightarrow$ client:
packet containing
where r is DNS re

v:

ism”:

.

nism”:

pq.

DEM security hypothesis: weak single-message version of security for secret-key authenticated encryption.

Chou: Is it safe to reuse k for multiple messages?

Answer: KEM+AE is safe; KEM+AE ⇒ KEM+“nDEM”. (But need literature on this!) AES-GCM, Salsa20-Poly1305, etc. aim for full AE security goal.

More complicated alternative: Use KEM+DEM to encrypt an n-time secret key m; reuse m.

DNSCurve: ECDH for DNS

Server knows ECDH secret k

Client knows ECDH secret k server’s public key $S = sG$.

Client → server: packet containing cG, $E_k(0,$ where $k = H(cS)$; E is authenticated cipher; q is DNS query.

Server → client: packet containing $E_k(1, r)$ where r is DNS response.

DEM security hypothesis: weak single-message version of security for secret-key authenticated encryption.

Chou: Is it safe to reuse k for multiple messages?

Answer: KEM+AE is safe; KEM+AE ⇒ KEM+"nDEM". (But need literature on this!) AES-GCM, Salsa20-Poly1305, etc. aim for full AE security goal.

More complicated alternative: Use KEM+DEM to encrypt an n-time secret key m; reuse m.

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c, server's public key $S = sG$.

Client → server:
packet containing $cG, E_k(0, q)$
where $k = H(cS)$;
E is authenticated cipher;
q is DNS query.

Server → client:
packet containing $E_k(1, r)$
where r is DNS response.

curity hypothesis:
gle-message version
ty for secret-key
cated encryption.

s it safe to reuse k
iple messages?

KEM+AE is safe;
E $\Rightarrow$ KEM+“nDEM”.
ed literature on this!)
M, Salsa20-Poly1305, etc.
full AE security goal.

mplicated alternative:
M+DEM to encrypt an
ecret key m; reuse m.

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c, server’s public key $S = sG$.

Client $\rightarrow$ server:
packet containing $cG, E_k(0, q)$
where $k = H(cS)$;
E is authenticated cipher;
q is DNS query.

Server $\rightarrow$ client:
packet containing $E_k(1, r)$
where r is DNS response.

Client ca
across m
but this
Let’s ass

othesis:
ge version
ret-key
ryption.

reuse k
ges?

E is safe;
M+“nDEM”.
re on this!)
0-Poly1305, etc.
curity goal.

alternative:
to encrypt an
m; reuse m.

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c, server’s public key $S = sG$.

Client $\rightarrow$ server: packet containing $cG, E_k(0, q)$ where $k = H(cS)$; E is authenticated cipher; q is DNS query.

Server $\rightarrow$ client: packet containing $E_k(1, r)$ where r is DNS response.

Client can reuse c
across multiple qu
but this leaks met
Let’s assume one-t

M".
)
5, etc.
.
e:
an
n.

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c, server's public key $S = sG$.

Client $\rightarrow$ server:
packet containing $cG, E_k(0, q)$
where $k = H(cS)$;
E is authenticated cipher;
q is DNS query.

Server $\rightarrow$ client:
packet containing $E_k(1, r)$
where r is DNS response.

Client can reuse c across multiple queries, but this leaks metadata. Let's assume one-time c.

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c,
server's public key $S = sG$.

Client $\rightarrow$ server:
packet containing $cG, E_k(0, q)$
where $k = H(cS)$;
E is authenticated cipher;
q is DNS query.

Server $\rightarrow$ client:
packet containing $E_k(1, r)$
where r is DNS response.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c, server's public key $S = sG$.

Client $\to$ server:
packet containing $cG, E_k(0, q)$
where $k = H(cS)$;
E is authenticated cipher;
q is DNS query.

Server $\to$ client:
packet containing $E_k(1, r)$
where r is DNS response.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

DNSCurve: ECDH for DNS

Server knows ECDH secret key s.

Client knows ECDH secret key c,
server's public key $S = sG$.

Client $\rightarrow$ server:
packet containing $cG, E_k(0, q)$
where $k = H(cS)$;
E is authenticated cipher;
q is DNS query.

Server $\rightarrow$ client:
packet containing $E_k(1, r)$
where r is DNS response.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

ve: ECDH for DNS

nows ECDH secret key s.

nows ECDH secret key c,
public key $S = sG$.

→ server:
ontaining $cG, E_k(0, q)$
$= H(cS)$;
henticated cipher;
S query.

→ client:
ontaining $E_k(1, r)$
is DNS response.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

Post-qua

"McElie
Client se
encapsul

Random
random
public ke

H for DNS

DH secret key s.

H secret key c,

$S = sG$.

$cG, E_k(0, q)$

;

cipher;

$E_k(1, r)$

sponse.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

Post-quantum enc

"McEliece KEM":
Client sends $k = H$
encapsulated as S

Random $c \in \mathbf{F}_2^{541}$
random small $e \in$
public key $S \in \mathbf{F}_2^{69}$

key s.

key c,

$q)$

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

Post-quantum encrypted DN

"McEliece KEM":
Client sends $k = H(c, e, Sc$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960 \times 5413}$.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960 \times 5413}$.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960\times 5413}$.

S has secret Goppa structure
allowing server to decrypt.

Client can reuse c
across multiple queries,
but this leaks metadata.
Let's assume one-time c.

KEM+AE view:

Client is sending $k = H(cS)$
encapsulated as cG.
This is an "ECDH KEM".

Client then uses k
to authenticate+encrypt.

Server also uses k
to authenticate+encrypt.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960\times 5413}$.

S has secret Goppa structure
allowing server to decrypt.

"Niederreiter KEM", smaller:
Client sends $k = H(e, S'e)$
encapsulated as $S'e \in \mathbf{F}_2^{1547}$.

an reuse c
multiple queries,
leaks metadata.
sume one-time c.

E view:

sending $k = H(cS)$
lated as cG.
an "ECDH KEM".

en uses k
nticate+encrypt.

lso uses k
nticate+encrypt.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960 \times 5413}$.

S has secret Goppa structure
allowing server to decrypt.

"Niederreiter KEM", smaller:
Client sends $k = H(e, S'e)$
encapsulated as $S'e \in \mathbf{F}_2^{1547}$.

Client —
packet c
(Combin

Server —
packet c

eries,

adata.

time c.

$= H(cS)$

G.

KEM".

ncrypt.

ncrypt.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960\times 5413}$.

S has secret Goppa structure
allowing server to decrypt.

"Niederreiter KEM", smaller:
Client sends $k = H(e, S'e)$
encapsulated as $S'e \in \mathbf{F}_2^{1547}$.

Client $\rightarrow$ server:
packet containing
(Combine with EC

Server $\rightarrow$ client:
packet containing

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960 \times 5413}$.

S has secret Goppa structure allowing server to decrypt.

"Niederreiter KEM", smaller:
Client sends $k = H(e, S'e)$
encapsulated as $S'e \in \mathbf{F}_2^{1547}$.

Client → server:
packet containing $Sc+e, E_k$
(Combine with ECDH KEM

Server → client:
packet containing $E_k(1, r)$.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960\times 5413}$.

S has secret Goppa structure
allowing server to decrypt.

"Niederreiter KEM", smaller:
Client sends $k = H(e, S'e)$
encapsulated as $S'e \in \mathbf{F}_2^{1547}$.

Client $\rightarrow$ server:
packet containing $Sc+e, E_k(0, q)$.
(Combine with ECDH KEM.)

Server $\rightarrow$ client:
packet containing $E_k(1, r)$.

Post-quantum encrypted DNS

"McEliece KEM":
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random $c \in \mathbf{F}_2^{5413}$;
random small $e \in \mathbf{F}_2^{6960}$;
public key $S \in \mathbf{F}_2^{6960\times 5413}$.

S has secret Goppa structure
allowing server to decrypt.

"Niederreiter KEM", smaller:
Client sends $k = H(e, S'e)$
encapsulated as $S'e \in \mathbf{F}_2^{1547}$.

Client $\rightarrow$ server:
packet containing $Sc+e$, $E_k(0, q)$.
(Combine with ECDH KEM.)

Server $\rightarrow$ client:
packet containing $E_k(1, r)$.

r states a server address
and the server's public key.
What if the key is too long
to fit into a single packet?

One simple answer:
Client separately requests
each block of public key.
Can do many requests in parallel.

antum encrypted DNS

ce KEM":

ends $k = H(c, e, Sc + e)$

lated as $Sc + e$.

$c \in \mathbf{F}_2^{5413}$;

small $e \in \mathbf{F}_2^{6960}$;

ey $S \in \mathbf{F}_2^{6960\times 5413}$.

ecret Goppa structure

server to decrypt.

reiter KEM", smaller:

ends $k = H(e, S'e)$

lated as $S'e \in \mathbf{F}_2^{1547}$.

Client $\rightarrow$ server:

packet containing $Sc+e$, $E_k(0, q)$.

(Combine with ECDH KEM.)

Server $\rightarrow$ client:

packet containing $E_k(1, r)$.

r states a server address

and the server's public key.

What if the key is too long

to fit into a single packet?

One simple answer:

Client separately requests

each block of public key.

Can do many requests in parallel.

Confider

Attacker

can't de

Integrity

Server n

but E_k i

Attacker

but can'

Attacker

Availabil

Client di

continue

eventual

rypted DNS

$H(c, e, Sc + e)$

$c + e$.

3;

$\mathbf{F}_2^{6960}$;

960×5413.

a structure

decrypt.

M", smaller:

$H(e, S'e)$

$'e \in \mathbf{F}_2^{1547}$.

Client $\to$ server:

packet containing $Sc+e$, $E_k(0, q)$.

(Combine with ECDH KEM.)

Server $\to$ client:

packet containing $E_k(1, r)$.

r states a server address
and the server's public key.
What if the key is too long
to fit into a single packet?

One simple answer:
Client separately requests
each block of public key.
Can do many requests in parallel.

Confidentiality:

Attacker can't gue

can't decrypt E_k(0

Integrity:

Server never signs

but E_k includes au

Attacker can send

but can't forge q o

Attacker *can* repla

Availability:

Client discards for

continues waiting

eventually retransm

NS

+ e)

e

r:

.

Client $\rightarrow$ server:
packet containing $Sc+e$, $E_k(0, q)$.
(Combine with ECDH KEM.)

Server $\rightarrow$ client:
packet containing $E_k(1, r)$.

r states a server address
and the server's public key.
What if the key is too long
to fit into a single packet?

One simple answer:
Client separately requests
each block of public key.
Can do many requests in parallel.

Confidentiality:
Attacker can't guess k,
can't decrypt $E_k(0, q)$, $E_k(1$

Integrity:
Server never signs anything,
but E_k includes authenticati
Attacker can send new queri
but can't forge q or r.
Attacker *can* replay request.

Availability:
Client discards forgery,
continues waiting for reply,
eventually retransmits reque

Client $\rightarrow$ server:
packet containing $Sc+e, E_k(0,q)$.
(Combine with ECDH KEM.)

Server $\rightarrow$ client:
packet containing $E_k(1,r)$.

r states a server address
and the server's public key.
What if the key is too long
to fit into a single packet?

One simple answer:
Client separately requests
each block of public key.
Can do many requests in parallel.

Confidentiality:
Attacker can't guess k,
can't decrypt $E_k(0,q), E_k(1,r)$.

Integrity:
Server never signs anything,
but E_k includes authentication.
Attacker can send new queries
but can't forge q or r.
Attacker *can* replay request.

Availability:
Client discards forgery,
continues waiting for reply,
eventually retransmits request.

→ server:
containing $Sc+e$, $E_k(0,q)$.
ne with ECDH KEM.)

→ client:
containing $E_k(1,r)$.

a server address
server's public key.
the key is too long
to a single packet?

ple answer:
eparately requests
ck of public key.
many requests in parallel.

Confidentiality:
Attacker can't guess k,
can't decrypt $E_k(0,q)$, $E_k(1,r)$.

Integrity:
Server never signs anything,
but E_k includes authentication.
Attacker can send new queries
but can't forge q or r.
Attacker *can* replay request.

Availability:
Client discards forgery,
continues waiting for reply,
eventually retransmits request.

Big keys

McEliece
for long-

Is this si
Do we n
lower-co
such as

Size of a
in Alexa

Web pag
public ke
but publ
can be r

$Sc+e$, $E_k(0,q)$.
CDH KEM.)

$E_k(1,r)$.

ddress
ublic key.
too long
packet?

r:
equests
lic key.
uests in parallel.

Confidentiality:
Attacker can't guess k,
can't decrypt $E_k(0,q)$, $E_k(1,r)$.

Integrity:
Server never signs anything,
but E_k includes authentication.
Attacker can send new queries
but can't forge q or r.
Attacker *can* replay request.

Availability:
Client discards forgery,
continues waiting for reply,
eventually retransmits request.

Big keys

McEliece public ke
for long-term conf

Is this size a probl
Do we need to sw
lower-confidence a
such as NTRU or

Size of average we
in Alexa Top 1000

Web page often ne
public keys for sev
but public key for
can be reused for

$_k(0, q)$.

.)

rallel.

Confidentiality:
Attacker can't guess k,
can't decrypt $E_k(0, q), E_k(1, r)$.

Integrity:
Server never signs anything,
but E_k includes authentication.
Attacker can send new queries
but can't forge q or r.
Attacker *can* replay request.

Availability:
Client discards forgery,
continues waiting for reply,
eventually retransmits request.

Big keys

McEliece public key is 1MB
for long-term confidence toc

Is this size a problem?
Do we need to switch to
lower-confidence approaches
such as NTRU or QC-MDP

Size of average web page
in Alexa Top 1000000: 1.8M

Web page often needs
public keys for several server
but public key for a server
can be reused for many page

Confidentiality:

Attacker can't guess k, can't decrypt $E_k(0, q)$, $E_k(1, r)$.

Integrity:

Server never signs anything, but E_k includes authentication. Attacker can send new queries but can't forge q or r. Attacker *can* replay request.

Availability:

Client discards forgery, continues waiting for reply, eventually retransmits request.

Big keys

McEliece public key is 1MB for long-term confidence today.

Is this size a problem? Do we need to switch to lower-confidence approaches such as NTRU or QC-MDPC?

Size of average web page in Alexa Top 1000000: 1.8MB.

Web page often needs public keys for several servers, but public key for a server can be reused for many pages.

ntiality:

r can't guess k,

crypt $E_k(0, q), E_k(1, r)$.

y:

ever signs anything,

ncludes authentication.

r can send new queries

t forge q or r.

r *can* replay request.

lity:

iscards forgery,

es waiting for reply,

ly retransmits request.

Big keys

McEliece public key is 1MB
for long-term confidence today.

Is this size a problem?
Do we need to switch to
lower-confidence approaches
such as NTRU or QC-MDPC?

Size of average web page
in Alexa Top 1000000: 1.8MB.

Web page often needs
public keys for several servers,
but public key for a server
can be reused for many pages.

Most im

on reuse

switchin

and **pro**

Rational

subseque

doesn't

e.g. Mic

switches

Safer: n

Easier to

new key

ess k,
$0, q), E_k(1, r)$.

anything,
uthentication.
new queries
or r.
y request.

gery,
for reply,
mits request.

Big keys

McEliece public key is 1MB
for long-term confidence today.

Is this size a problem?
Do we need to switch to
lower-confidence approaches
such as NTRU or QC-MDPC?

Size of average web page
in Alexa Top 1000000: 1.8MB.

Web page often needs
public keys for several servers,
but public key for a server
can be reused for many pages.

Most important li
on reuse of public
switching to new k
and **promptly era**

Rationale: "forwa
subsequent theft o
doesn't allow decr

e.g. Microsoft SCh
switches keys ever

Safer: new key ev

Easier to impleme
new key every con

, r).

ion.

ies

st.

Big keys

McEliece public key is 1MB
for long-term confidence today.

Is this size a problem?
Do we need to switch to
lower-confidence approaches
such as NTRU or QC-MDPC?

Size of average web page
in Alexa Top 1000000: 1.8MB.

Web page often needs
public keys for several servers,
but public key for a server
can be reused for many pages.

Most important limitation
on reuse of public keys:
switching to new keys
and **promptly erasing old k**

Rationale: “forward secrecy”
subsequent theft of compute
doesn’t allow decryption.

e.g. Microsoft SChannel
switches keys every two hou

Safer: new key every minute

Easier to implement:
new key every connection.

Big keys

McEliece public key is 1MB for long-term confidence today.

Is this size a problem?
Do we need to switch to lower-confidence approaches such as NTRU or QC-MDPC?

Size of average web page in Alexa Top 1000000: 1.8MB.

Web page often needs public keys for several servers, but public key for a server can be reused for many pages.

Most important limitation on reuse of public keys: switching to new keys and **promptly erasing old keys**.

Rationale: "forward secrecy"— subsequent theft of computer doesn't allow decryption.

e.g. Microsoft SChannel switches keys every two hours.

Safer: new key every minute.

Easier to implement: new key every connection.

s

e public key is 1MB
-term confidence today.

ze a problem?
eed to switch to
nfidence approaches
NTRU or QC-MDPC?

average web page
Top 1000000: 1.8MB.

ge often needs
eys for several servers,
lic key for a server
eused for many pages.

Most important limitation
on reuse of public keys:
switching to new keys
and **promptly erasing old keys**.

Rationale: "forward secrecy"—
subsequent theft of computer
doesn't allow decryption.

e.g. Microsoft SChannel
switches keys every two hours.

Safer: new key every minute.

Easier to implement:
new key every connection.

What is
a new ke

If server
key gen,
client en
server de

ey is 1MB
idence today.

em?
itch to
pproaches
QC-MDPC?

eb page
0000: 1.8MB.

eeds
eral servers,
a server
many pages.

Most important limitation
on reuse of public keys:
switching to new keys
and **promptly erasing old keys**.

Rationale: "forward secrecy"—
subsequent theft of computer
doesn't allow decryption.

e.g. Microsoft SChannel
switches keys every two hours.

Safer: new key every minute.

Easier to implement:
new key every connection.

What is the perfor
a new key every m

If server makes ne
key gen, $\leq$1 per m
client encrypts to
server decrypts.

day.

C?

MB.

rs,

es.

Most important limitation
on reuse of public keys:
switching to new keys
and **promptly erasing old keys**.

Rationale: “forward secrecy”—
subsequent theft of computer
doesn’t allow decryption.

e.g. Microsoft SChannel
switches keys every two hours.

Safer: new key every minute.

Easier to implement:
new key every connection.

What is the performance of
a new key every minute?

If server makes new key:
key gen, ≤ 1 per minute;
client encrypts to new key;
server decrypts.

Most important limitation
on reuse of public keys:
switching to new keys
and **promptly erasing old keys**.

Rationale: “forward secrecy”—
subsequent theft of computer
doesn’t allow decryption.

e.g. Microsoft SChannel
switches keys every two hours.

Safer: new key every minute.

Easier to implement:
new key every connection.

What is the performance of
a new key every minute?

If server makes new key:
key gen, $\leq$1 per minute;
client encrypts to new key;
server decrypts.

Most important limitation
on reuse of public keys:
switching to new keys
and **promptly erasing old keys**.

Rationale: "forward secrecy"—
subsequent theft of computer
doesn't allow decryption.

e.g. Microsoft SChannel
switches keys every two hours.

Safer: new key every minute.

Easier to implement:
new key every connection.

What is the performance of
a new key every minute?

If server makes new key:
key gen, $\leq$1 per minute;
client encrypts to new key;
server decrypts.

If client makes new key:
client has key-gen cost;
server has encryption cost;
client has decryption cost.

Either way:
one key transmission for each
active client-server pair.

portant limitation
of public keys:
g to new keys
mptly erasing old keys.

e: "forward secrecy"—
ent theft of computer
allow decryption.

rosoft SChannel
keys every two hours.

ew key every minute.

implement:
every connection.

What is the performance of
a new key every minute?

If server makes new key:
key gen, $\le$1 per minute;
client encrypts to new key;
server decrypts.

If client makes new key:
client has key-gen cost;
server has encryption cost;
client has decryption cost.

Either way:
one key transmission for each
active client-server pair.

How doe
encrypt
without

mitation
keys:
keys
sing old keys.

rd secrecy"—
of computer
yption.

annel
y two hours.

ery minute.

nt:
nection.

What is the performance of
a new key every minute?

If server makes new key:
key gen, ≤ 1 per minute;
client encrypts to new key;
server decrypts.

If client makes new key:
client has key-gen cost;
server has encryption cost;
client has decryption cost.

Either way:
one key transmission for each
active client-server pair.

How does a *statel*
encrypt to a new
without storing th

keys.

'—

er

rs.

e.

What is the performance of a new key every minute?

If server makes new key:
key gen, $\leq$1 per minute;
client encrypts to new key;
server decrypts.

If client makes new key:
client has key-gen cost;
server has encryption cost;
client has decryption cost.

Either way:
one key transmission for each active client-server pair.

How does a *stateless* server encrypt to a new client key without storing the key?

What is the performance of
a new key every minute?

If server makes new key:
key gen, $\leq$1 per minute;
client encrypts to new key;
server decrypts.

If client makes new key:
client has key-gen cost;
server has encryption cost;
client has decryption cost.

Either way:
one key transmission for each
active client-server pair.

How does a *stateless* server
encrypt to a new client key
without storing the key?

What is the performance of a new key every minute?

If server makes new key: key gen, $\leq$1 per minute; client encrypts to new key; server decrypts.

If client makes new key: client has key-gen cost; server has encryption cost; client has decryption cost.

Either way: one key transmission for each active client-server pair.

How does a *stateless* server encrypt to a new client key without storing the key?

Slice McEliece public key so that each slice of encryption produces separate small output.

Client sends slices (in parallel), receives outputs as cookies, sends cookies (in parallel). Server combines cookies. Continue up through tree.

Server generates randomness as secret function of key hash. Statelessly verifies key hash.