The post-quantum Internet

Daniel J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Includes joint work with:

Tanja Lange

Technische Universiteit Eindhoven

Risk management

"Combining congruences": state-of-the-art pre-quantum attack against original DH, RSA, and some lattice systems.

Long history, including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

t-quantum Internet

. Bernstein

ty of Illinois at Chicago &
che Universiteit Eindhoven

joint work with:

ange

che Universiteit Eindhoven

Risk management

"Combining congruences":
state-of-the-art pre-quantum
attack against original DH,
RSA, and some lattice systems.

Long history, including
many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

Also ma
$>$100 sc

Costs of
breaking
$\approx 2^{120}$,
$\approx 2^{110}$,
$\approx 2^{100}$,
$\approx 2^{80}$,
(FFS is

n Internet

n

is at Chicago &
siteit Eindhoven

k with:

siteit Eindhoven

Risk management

"Combining congruences": state-of-the-art pre-quantum attack against original DH, RSA, and some lattice systems.

Long history, including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

Also many smaller
>100 scientific pa

Costs of these alg
breaking RSA-102
$\approx 2^{120}$, $\approx 2^{170}$, CF
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS
$\approx 2^{80}$, $\approx 2^{112}$, NF
(FFS is not releva

ago &
hoven

hoven

Risk management

"Combining congruences": state-of-the-art pre-quantum attack against original DH, RSA, and some lattice systems.

Long history, including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

Also many smaller improvem
>100 scientific papers.

Costs of these algorithms fo
breaking RSA-1024, RSA-20
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA

Risk management

"Combining congruences": state-of-the-art pre-quantum attack against original DH, RSA, and some lattice systems.

Long history, including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

Also many smaller improvements: >100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

Risk management

"Combining congruences": state-of-the-art pre-quantum attack against original DH, RSA, and some lattice systems.

Long history, including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

Also many smaller improvements: >100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

Risk management

"Combining congruences": state-of-the-art pre-quantum attack against original DH, RSA, and some lattice systems.

Long history, including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS.

Also many smaller improvements: $>$100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

How much risk is there of secret breakthroughs?

nagement

ning congruences":
-the-art pre-quantum
gainst original DH,
d some lattice systems.

story, including
ajor improvements:
FRAC;
near sieve (LS);
uadratic sieve (QS);
umber-field sieve (NFS);
nction-field sieve (FFS);
edium-prime FFS/NFS;
$^q - x$ FFS.

Also many smaller improvements:
>100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

How much risk is there of secret breakthroughs?

If we pu
exploring
will we f
At least

uences":
e-quantum
ginal DH,
ttice systems.

ding
vements:

(LS);
eve (QS);
d sieve (NFS);
d sieve (FFS);
me FFS/NFS;
.

Also many smaller improvements:
>100 scientific papers.

Costs of these algorithms for
breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there
of future breakthroughs?

How much risk is there
of secret breakthroughs?

If we put enough e
exploring Attack M
will we find the hig
At least within ϵ?

n

ms.

FS);

FS);

FS;

Also many smaller improvements:
>100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

How much risk is there of secret breakthroughs?

If we put enough effort into
exploring Attack Mountain,
will we find the highest peak
At least within ϵ?

Also many smaller improvements:
$>$100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

How much risk is there of secret breakthroughs?

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Also many smaller improvements:
$>$100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

How much risk is there of secret breakthroughs?

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

Also many smaller improvements:
>100 scientific papers.

Costs of these algorithms for breaking RSA-1024, RSA-2048:
$\approx 2^{120}$, $\approx 2^{170}$, CFRAC;
$\approx 2^{110}$, $\approx 2^{160}$, LS;
$\approx 2^{100}$, $\approx 2^{150}$, QS;
$\approx 2^{80}$, $\approx 2^{112}$, NFS.
(FFS is not relevant to RSA.)

How much risk is there of future breakthroughs?

How much risk is there of secret breakthroughs?

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

ny smaller improvements:
ientific papers.

these algorithms for
; RSA-1024, RSA-2048:
$\approx 2^{170}$, CFRAC;
$\approx 2^{160}$, LS;
$\approx 2^{150}$, QS;
$\approx 2^{112}$, NFS.
not relevant to RSA.)

ch risk is there
e breakthroughs?

ch risk is there
t breakthroughs?

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

Conserva
prefer m
less hug
more th

improvements:
pers.

orithms for
4, RSA-2048:
RAC;

;
S.
nt to RSA.)

there
oughs?

there
oughs?

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

Conservative crypt
prefer mountains t
less huge, less fog
more thoroughly e

nents:

r

048:

.)

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

1986 Miller "Use of elliptic curves in cryptography": "It is extremely unlikely that an 'index calculus' attack [combining-congruences attack] on the elliptic curve method will ever be able to work."

If we put enough effort into exploring Attack Mountain, will we find the highest peak? At least within ϵ?

Combining-Congruences Mountain is a huge, foggy, high-dimensional mountain with many paths up. Scary: easy to imagine that we're not at the top yet.

18-year bet announced in 2014: Joux wins if RSA-2048 is broken first by pre-quantum algorithms; I win if RSA-2048 is broken first by quantum algorithms.

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

1986 Miller "Use of elliptic curves in cryptography": "It is extremely unlikely that an 'index calculus' attack [combining-congruences attack] on the elliptic curve method will ever be able to work."

This is the core argument for ECC. Exceptions: rare curves with special structure—e.g., pairings.

t enough effort into
g Attack Mountain,
find the highest peak?
within ϵ?

ng-Congruences Mountain
e, foggy, high-dimensional
n with many paths up.
asy to imagine that
ot at the top yet.

bet announced in 2014:
ns if RSA-2048 is broken
pre-quantum algorithms;
RSA-2048 is broken
quantum algorithms.

Conservative cryptographers
prefer mountains that seem
less huge, less foggy,
more thoroughly explored.

1986 Miller "Use of
elliptic curves in cryptography":
"It is extremely unlikely
that an 'index calculus' attack
[combining-congruences attack]
on the elliptic curve method
will ever be able to work."

This is the core argument for
ECC. Exceptions: rare curves with
special structure—e.g., pairings.

2015 La
bet your

The setting

It's 2050. Q

Evil Party A
practically a
vaccinations
the past 70
by law, but
2020. Your
doctor's pub
public-key a

Organs are
if they can i
presented w
Statement.

(This is mea
trust. Let's

effort into
Mountain,
ghest peak?

uences Mountain
high-dimensional
ny paths up.
agine that
op yet.

nced in 2014:
2048 is broken
um algorithms;
is broken
algorithms.

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

1986 Miller "Use of elliptic curves in cryptography": "It is extremely unlikely that an 'index calculus' attack [combining-congruences attack] on the elliptic curve method will ever be able to work."

This is the core argument for ECC. Exceptions: rare curves with special structure—e.g., pairings.

2015 Lange: "Wo
bet your kidneys o

The setting

It's 2050. Quantum computers w

Evil Party A now runs the countr
practically all 21st-century Intern
vaccinations are bad and jails any
the past 70 years. Doctor-patient
by law, but your health record fro
2020. Your health record is prote
doctor's public key, using our rec
public-key and authenticated sym

Organs are a scarce resource. Ho
if they can identify the donor (DN
presented with the donor's digital
Statement. They use our 2015 re

(This is meant to scare you, so th
trust. Let's make sure that this d

k?

untain
sional
up.

014:
oken
nms;

.

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

1986 Miller "Use of elliptic curves in cryptography": "It is extremely unlikely that an 'index calculus' attack [combining-congruences attack] on the elliptic curve method will ever be able to work."

This is the core argument for ECC. Exceptions: rare curves with special structure—e.g., pairings.

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to re
practically all 21st-century Internet traffic. Evil Party A
vaccinations are bad and jails anybody who was vaccina
the past 70 years. Doctor-patient confidentiality is still
by law, but your health record from birth has been onli
2020. Your health record is protected only by encryptio
doctor's public key, using our recommendation from 20
public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices
if they can identify the donor (DNA tests are cheap) ar
presented with the donor's digitally signed Donor Volun
Statement. They use our 2015 recommended signature

(This is meant to scare you, so that you recommend on
trust. Let's make sure that this dystopia will not happe

Conservative cryptographers prefer mountains that seem less huge, less foggy, more thoroughly explored.

1986 Miller "Use of elliptic curves in cryptography": "It is extremely unlikely that an 'index calculus' attack [combining-congruences attack] on the elliptic curve method will ever be able to work."

This is the core argument for ECC. Exceptions: rare curves with special structure—e.g., pairings.

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to records of practically all 21st-century Internet traffic. Evil Party A thinks vaccinations are bad and jails anybody who was vaccinated during the past 70 years. Doctor-patient confidentiality is still protected by law, but your health record from birth has been online since 2020. Your health record is protected only by encryption to your doctor's public key, using our recommendation from 2015 of public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices for organs if they can identify the donor (DNA tests are cheap) and are presented with the donor's digitally signed Donor Volunteer Statement. They use our 2015 recommended signature system.

(This is meant to scare you, so that you recommend only what you trust. Let's make sure that this dystopia will not happen.)

5

ative cryptographers
ountains that seem
e, less foggy,
oroughly explored.

ller "Use of
curves in cryptography":
tremely unlikely
'index calculus' attack
ing-congruences attack]
lliptic curve method
be able to work."

he core argument for
xceptions: rare curves with
structure—e.g., pairings.

6

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to records of practically all 21st-century Internet traffic. Evil Party A thinks vaccinations are bad and jails anybody who was vaccinated during the past 70 years. Doctor-patient confidentiality is still protected by law, but your health record from birth has been online since 2020. Your health record is protected only by encryption to your doctor's public key, using our recommendation from 2015 of public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices for organs if they can identify the donor (DNA tests are cheap) and are presented with the donor's digitally signed Donor Volunteer Statement. They use our 2015 recommended signature system.

(This is meant to scare you, so that you recommend only what you trust. Let's make sure that this dystopia will not happen.)

Risk of f
big unive
noticeab
terrifying

tographers
that seem
gy,
xplored.

of
ryptography":
nlikely
culus' attack
uences attack]
ve method
o work."

gument for
rare curves with
-e.g., pairings.

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to records of practically all 21st-century Internet traffic. Evil Party A thinks vaccinations are bad and jails anybody who was vaccinated during the past 70 years. Doctor-patient confidentiality is still protected by law, but your health record from birth has been online since 2020. Your health record is protected only by encryption to your doctor's public key, using our recommendation from 2015 of public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices for organs if they can identify the donor (DNA tests are cheap) and are presented with the donor's digitally signed Donor Volunteer Statement. They use our 2015 recommended signature system.

(This is meant to scare you, so that you recommend only what you trust. Let's make sure that this dystopia will not happen.)

Risk of future atta
big universal quan
noticeable probabi
terrifying impact.

ny":

ck

ack]

or

es with

ngs.

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to records of practically all 21st-century Internet traffic. Evil Party A thinks vaccinations are bad and jails anybody who was vaccinated during the past 70 years. Doctor-patient confidentiality is still protected by law, but your health record from birth has been online since 2020. Your health record is protected only by encryption to your doctor's public key, using our recommendation from 2015 of public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices for organs if they can identify the donor (DNA tests are cheap) and are presented with the donor's digitally signed Donor Volunteer Statement. They use our 2015 recommended signature system.

(This is meant to scare you, so that you recommend only what you trust. Let's make sure that this dystopia will not happen.)

Risk of future attacker havi
big universal quantum comp
noticeable probability;
terrifying impact.

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to records of practically all 21st-century Internet traffic. Evil Party A thinks vaccinations are bad and jails anybody who was vaccinated during the past 70 years. Doctor-patient confidentiality is still protected by law, but your health record from birth has been online since 2020. Your health record is protected only by encryption to your doctor's public key, using our recommendation from 2015 of public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices for organs if they can identify the donor (DNA tests are cheap) and are presented with the donor's digitally signed Donor Volunteer Statement. They use our 2015 recommended signature system.

(This is meant to scare you, so that you recommend only what you trust. Let's make sure that this dystopia will not happen.)

Risk of future attacker having big universal quantum computer: noticeable probability; terrifying impact.

2015 Lange: "Would you bet your kidneys on that?"

The setting

It's 2050. Quantum computers were built years ago.

Evil Party A now runs the country and has access to records of practically all 21st-century Internet traffic. Evil Party A thinks vaccinations are bad and jails anybody who was vaccinated during the past 70 years. Doctor-patient confidentiality is still protected by law, but your health record from birth has been online since 2020. Your health record is protected only by encryption to your doctor's public key, using our recommendation from 2015 of public-key and authenticated symmetric encryption.

Organs are a scarce resource. Hospitals pay high prices for organs if they can identify the donor (DNA tests are cheap) and are presented with the donor's digitally signed Donor Volunteer Statement. They use our 2015 recommended signature system.

(This is meant to scare you, so that you recommend only what you trust. Let's make sure that this dystopia will not happen.)

Risk of future attacker having big universal quantum computer: noticeable probability; terrifying impact.

Fortunately, we already know some confidence-inspiring post-quantum systems, including

- hash-based signatures;
- McEliece public-key encryption;
- AES-256 etc.

https://pqcrypto.eu.org/docs/initial-recommendations.pdf

nge: “Would you
kidneys on that?”

Quantum computers were built years ago.

now runs the country and has access to records of
ll 21st-century Internet traffic. Evil Party A thinks
s are bad and jails anybody who was vaccinated during
years. Doctor-patient confidentiality is still protected
your health record from birth has been online since
health record is protected only by encryption to your
lic key, using our recommendation from 2015 of
nd authenticated symmetric encryption.

a scarce resource. Hospitals pay high prices for organs
dentify the donor (DNA tests are cheap) and are
ith the donor's digitally signed Donor Volunteer
They use our 2015 recommended signature system.

ant to scare you, so that you recommend only what you
make sure that this dystopia will not happen.)

Risk of future attacker having
big universal quantum computer:
noticeable probability;
terrifying impact.

Fortunately, we already know
some confidence-inspiring
post-quantum systems, including

- hash-based signatures;
- McEliece public-key encryption;
- AES-256 etc.

https://pqcrypto.eu.org/docs/initial-recommendations.pdf

Applicat

Your co

new vers

Your co

signatur

from the

Critical

Otherwi

insert m

e.g. Ope

signed u

ECC sig

uld you
n that?"

ere built years ago.

y and has access to records of
et traffic. Evil Party A thinks
body who was vaccinated during
confidentiality is still protected
m birth has been online since
cted only by encryption to your
ommendation from 2015 of
metric encryption.

spitals pay high prices for organs
NA tests are cheap) and are
ly signed Donor Volunteer
commended signature system.

at you recommend only what you
ystopia will not happen.)

Risk of future attacker having big universal quantum computer: noticeable probability; terrifying impact.

Fortunately, we already know some confidence-inspiring post-quantum systems, including

- hash-based signatures;
- McEliece public-key encryption;
- AES-256 etc.

https://pqcrypto.eu.org/docs/initial-recommendations.pdf

Application: softw

Your computer do
new version of its

Your computer che
signature on the d
from the OS manu

Critical use of cryp
Otherwise crimina
insert malware into

e.g. OpenBSD upd
signed using state-
ECC signature sys

cords of
thinks
ated during
protected
ne since
n to your
15 of

for organs
nd are
teer
system.

ly what you
en.)

Risk of future attacker having big universal quantum computer: noticeable probability; terrifying impact.

Fortunately, we already know some confidence-inspiring post-quantum systems, including

- hash-based signatures;
- McEliece public-key encryption;
- AES-256 etc.

https://pqcrypto.eu.org/docs/initial-recommendations.pdf

Application: software updat

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto! Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed2

Risk of future attacker having big universal quantum computer: noticeable probability; terrifying impact.

Fortunately, we already know some confidence-inspiring post-quantum systems, including

- hash-based signatures;
- McEliece public-key encryption;
- AES-256 etc.

`https://pqcrypto.eu.org/docs/initial-recommendations.pdf`

Application: software updates

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto! Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed25519.

future attacker having
ersal quantum computer:
le probability;
g impact.

tely, we already know
nfidence-inspiring
antum systems, including
based signatures;
ece public-key encryption;
56 etc.

`//pqcrypto.eu.org/docs/`
`1-recommendations.pdf`

Application: software updates

Your computer downloads
new version of its OS.

Your computer checks
signature on the download
from the OS manufacturer.

Critical use of crypto!
Otherwise criminals could
insert malware into the OS.

e.g. OpenBSD updates are
signed using state-of-the-art
ECC signature system: Ed25519.

Pre-quan
needs to
post-qua

cker having
tum computer:
lity;

ready know
nspiring
tems, including
atures;
key encryption;

to.eu.org/docs/
endations.pdf

Application: software updates

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto! Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed25519.

Pre-quantum signa
needs to be replac
post-quantum sign

ng
uter:

w

uding

ption;

g/docs/
s.pdf

Application: software updates

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto!
Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed25519.

Pre-quantum signature syste
needs to be replaced with
post-quantum signature syst

Application: software updates

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto! Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed25519.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Application: software updates

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto! Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed25519.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier: Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key.
$P + Q$ signature concatenates P signature, Q signature.

Application: software updates

Your computer downloads new version of its OS.

Your computer checks signature on the download from the OS manufacturer.

Critical use of crypto! Otherwise criminals could insert malware into the OS.

e.g. OpenBSD updates are signed using state-of-the-art ECC signature system: Ed25519.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier: Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key. $P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key? Replace public key with hash. Include missing information ($\leq$ entire key) inside signature.

ion: software updates

mputer downloads
sion of its OS.

mputer checks
e on the download
e OS manufacturer.

use of crypto!
se criminals could
alware into the OS.

enBSD updates are
sing state-of-the-art
nature system: Ed25519.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier: Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key.
$P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key? Replace public key with hash. Include missing information ($\leq$ entire key) inside signature.

e.g. Ed2

SPHINC
≈50 mil
≈1 milli
Negligib
verify co

vare updates

wnloads
OS.

ecks
ownload
ufacturer.

pto!
ls could
o the OS.

dates are
-of-the-art
tem: Ed25519.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier: Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key. $P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key? Replace public key with hash. Include missing information ($\leq$ entire key) inside signature.

e.g. Ed25519+SPI

SPHINCS-256 sigr
≈50 million cycles
≈1 million cycles t
Negligible cost to
verify compared to

es

5519.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier:
Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key.
$P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key?
Replace public key with hash.
Include missing information ($\leq$ entire key) inside signature.

e.g. Ed25519+SPHINCS-25

SPHINCS-256 signature is 4
≈50 million cycles to genera
≈1 million cycles to verify.
Negligible cost to sign, trans
verify compared to OS upda

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier:
Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key.
$P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key?
Replace public key with hash.
Include missing information ($\leq$ entire key) inside signature.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB;
$\approx$50 million cycles to generate;
$\approx$1 million cycles to verify.
Negligible cost to sign, transmit, verify compared to OS update.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier:
Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key.
$P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key?
Replace public key with hash.
Include missing information ($\leq$ entire key) inside signature.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB;
$\approx$50 million cycles to generate;
$\approx$1 million cycles to verify.
Negligible cost to sign, transmit, verify compared to OS update.

+Ed25519: unnoticeable cost.
Some extra system complexity, but the system includes Ed25519 code anyway.

Pre-quantum signature system P needs to be replaced with post-quantum signature system Q.

Make auditors happier: Replace P with $P + Q$.

$P + Q$ public key concatenates P public key, Q public key. $P + Q$ signature concatenates P signature, Q signature.

Want a tiny public key? Replace public key with hash. Include missing information ($\leq$ entire key) inside signature.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB; $\approx$50 million cycles to generate; $\approx$1 million cycles to verify. Negligible cost to sign, transmit, verify compared to OS update.

+Ed25519: unnoticeable cost. Some extra system complexity, but the system includes Ed25519 code anyway.

Auditor sees very easily that Ed25519+SPHINCS-256 security $\geq$ Ed25519 security.

ntum signature system P
be replaced with
antum signature system Q.

uditors happier:
P with $P+Q$.

ublic key concatenates
key, Q public key.
ignature concatenates
ture, Q signature.

tiny public key?
public key with hash.
missing information
e key) inside signature.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB;
≈50 million cycles to generate;
≈1 million cycles to verify.
Negligible cost to sign, transmit,
verify compared to OS update.

+Ed25519: unnoticeable cost.
Some extra system complexity,
but the system includes
Ed25519 code anyway.

Auditor sees very easily
that Ed25519+SPHINCS-256
security $\geq$ Ed25519 security.

Does de
mean th
On the o

Pre-quar
Hash-ba
even mo
than EC
But und
takes ex

ature system P
ed with
nature system Q.

pier:
$+ Q$.

concatenates
ublic key.
oncatenates
nature.

key?
with hash.
formation
de signature.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB; ≈50 million cycles to generate; ≈1 million cycles to verify. Negligible cost to sign, transmit, verify compared to OS update.

+Ed25519: unnoticeable cost. Some extra system complexity, but the system includes Ed25519 code anyway.

Auditor sees very easily that Ed25519+SPHINCS-256 security $\geq$ Ed25519 security.

Does deployment
mean that we don
On the contrary!

Pre-quantum situa
Hash-based signat
even more confide
than ECC signatu
But understanding
takes extra work f

em P

tem Q.

tes

es

n.

ure.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB;
≈50 million cycles to generate;
≈1 million cycles to verify.
Negligible cost to sign, transmit, verify compared to OS update.

+Ed25519: unnoticeable cost.
Some extra system complexity, but the system includes Ed25519 code anyway.

Auditor sees very easily that Ed25519+SPHINCS-256 security $\geq$ Ed25519 security.

Does deployment of $P + Q$
mean that we don't trust Q
On the contrary!

Pre-quantum situation:
Hash-based signatures are even more confidence-inspiri
than ECC signatures.
But understanding this fact
takes extra work for auditor.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB; $\approx$50 million cycles to generate; $\approx$1 million cycles to verify. Negligible cost to sign, transmit, verify compared to OS update.

+Ed25519: unnoticeable cost. Some extra system complexity, but the system includes Ed25519 code anyway.

Auditor sees very easily that Ed25519+SPHINCS-256 security $\geq$ Ed25519 security.

Does deployment of $P + Q$ mean that we don't trust Q? On the contrary!

Pre-quantum situation: Hash-based signatures are even more confidence-inspiring than ECC signatures. But understanding this fact takes extra work for auditor.

e.g. Ed25519+SPHINCS-256.

SPHINCS-256 signature is 41KB;
≈50 million cycles to generate;
≈1 million cycles to verify.
Negligible cost to sign, transmit, verify compared to OS update.

+Ed25519: unnoticeable cost.
Some extra system complexity, but the system includes Ed25519 code anyway.

Auditor sees very easily that Ed25519+SPHINCS-256 security $\geq$ Ed25519 security.

Does deployment of $P + Q$ mean that we don't trust Q?
On the contrary!

Pre-quantum situation:
Hash-based signatures are even more confidence-inspiring than ECC signatures.
But understanding this fact takes extra work for auditor.

Long-term situation:
Users see quantum computers easily breaking P. Simplify system by switching from $P + Q$ to Q.

5519+SPHINCS-256.

S-256 signature is 41KB;
lion cycles to generate;
on cycles to verify.
le cost to sign, transmit,
mpared to OS update.

19: unnoticeable cost.
tra system complexity,
system includes
code anyway.

sees very easily
25519+SPHINCS-256
$\geq$ Ed25519 security.

Does deployment of $P + Q$
mean that we don't trust Q?
On the contrary!

Pre-quantum situation:
Hash-based signatures are
even more confidence-inspiring
than ECC signatures.
But understanding this fact
takes extra work for auditor.

Long-term situation:
Users see quantum computers
easily breaking P. Simplify system
by switching from $P + Q$ to Q.

IP: Inter

IP comm
limited-l

Each co
has a 4-
e.g. www
address

Your bro
addresse
gives pa
Hopefull
that pac

HINCS-256.

nature is 41KB;
; to generate;
to verify.
sign, transmit,
o OS update.

iceable cost.
n complexity,
cludes
way.

easily
HINCS-256
.9 security.

Does deployment of $P + Q$ mean that we don't trust Q? On the contrary!

Pre-quantum situation: Hash-based signatures are even more confidence-inspiring than ECC signatures. But understanding this fact takes extra work for auditor.

Long-term situation: Users see quantum computers easily breaking P. Simplify system by switching from $P + Q$ to Q.

IP: Internet Proto

IP communicates
limited-length byt

Each computer on
has a 4-byte "IP a
e.g. `www.pqcrypt`
address `131.155.`

Your browser crea
addressed to `131.`
gives packet to th
Hopefully the Inte
that packet to `131`

6.

1KB;

te;

smit,

te.

st.

ity,

56

.

Does deployment of $P + Q$ mean that we don't trust Q? On the contrary!

Pre-quantum situation: Hash-based signatures are even more confidence-inspiring than ECC signatures. But understanding this fact takes extra work for auditor.

Long-term situation: Users see quantum computers easily breaking P. Simplify system by switching from $P + Q$ to Q.

IP: Internet Protocol

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

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

Your browser creates a pack
addressed to `131.155.70.1`
gives packet to the Internet.
Hopefully the Internet delive
that packet to `131.155.70`

Does deployment of $P+Q$ mean that we don't trust Q? On the contrary!

Pre-quantum situation: Hash-based signatures are even more confidence-inspiring than ECC signatures. But understanding this fact takes extra work for auditor.

Long-term situation: Users see quantum computers easily breaking P. Simplify system by switching from $P+Q$ to Q.

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`.

ployment of $P + Q$
at we don't trust Q?
contrary!

ntum situation:
sed signatures are
re confidence-inspiring
C signatures.
erstanding this fact
tra work for auditor.

rm situation:
e quantum computers
eaking P. Simplify system
hing from $P + Q$ to Q.

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

of $P + Q$

't trust Q?

ation:

ures are

nce-inspiring

res.

this fact

or auditor.

on:

n computers

Simplify system

$P + Q$ to Q.

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

connect to `www.p`

Browser learns "1

by asking a name

the `pqcrypto.or`

Browser → `131.1`

"`Where is www.p`

?

ng

ers
system
Q.

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.pq`

`199.19.54.1` →
“`Ask the pqcrypt`
`name server, 13`

:

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:
ress 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, S

Browser → server:
"ACK 747bfa42"

Server now allocat
for this TCP conn

Browser splits data
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
Usually 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 i
Complic
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 retransn
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 conn
- inside the TCP c
  builds a TLS con
  by exchanging c
- inside the TLS c
  sends HTTP req

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 connectior 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
ission 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
somethin
but has

Browser
make a
but this
Huge da

rtised by TCP:
ams”.

s loses packets
out of order.
CP connections:
he counter
acket.

nits 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 from

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
he 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 mod 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
$P + Q$:

.., DNSCurve,
MinimaLT,
uthenticate and
et separately.

ket
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 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+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 cat
any modification

Much easier to ge
Also generalizes ni
$P + Q$: hash conc

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 $k = H(r, r^e \bmod pq)$

"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.
$P + Q$: hash concatenation.

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.
$P + Q$: hash concatenation.

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.
$P + Q$: hash concatenation.

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:
rypted

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

blems:
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.
$P + Q$: hash concatenation.

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. $P + Q$: hash concatenation.

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.
$P + Q$: hash concatenation.

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.
$k = H(r, r^e \bmod pq)$.

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

icator catches
dification of r^e mod pq.

asier to get right.
neralizes nicely.
hash concatenation.

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.

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.

atenation.

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 $\to$ server:

packet containing

where $k = H(cS)$;

E is authenticated

q is DNS query.

Server $\to$ client:

packet containing

where r is DNS re

y:

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 $\Rightarrow$ 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 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.

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

s it safe to reuse k
iple messages?

KEM+AE is safe;
AE $\Rightarrow$ 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:
o 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 $\to$ 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 qu
but this leaks met
Let’s assume one-t

M”.
)
5, etc.
.
e:
an
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.

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.

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.

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 → 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.

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

→ client:
containing $E_k(1, r)$
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,

$S = sG$.

$cG, E_k(0, q)$

;

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

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.

AE view:

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

hen uses k
enticate+encrypt.

lso uses k
enticate+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 $S'e \in \mathbf{F}_2^{1547}$.

"NTRU
obviousl
Client se
encapsul

eries,

adata.

time c.

$k = H(cS)$

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

"NTRU KEM",
obviously totally u
Client sends $k = H$
encapsulated as S

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

"NTRU KEM",
obviously totally unrelated:
Client sends $k = H(c, e, Sc$
encapsulated as $Sc + e$.

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

"NTRU KEM",
obviously totally unrelated:
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

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

"NTRU KEM",
obviously totally unrelated:
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random small
$c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;
public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.
Server recovers $3sc + te$,
then te mod 3, then e, then c.

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

"NTRU KEM",
obviously totally unrelated:
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random small
$c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;
public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.
Server recovers $3sc + te$,
then te mod 3, then e, then c.

Can imitate Niederreiter in the
NTRU context: e.g. "Ring-LWR".

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

“NTRU KEM”,
obviously totally unrelated:
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random small
$c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;
public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.
Server recovers $3sc + te$,
then te mod 3, then e, then c.

Can imitate Niederreiter in the
NTRU context: e.g. “Ring-LWR”.

Client –
packet c
(Combin

Server –
packet c

rypted DNS

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

$c + e$.

3;

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

960×5413.

a structure

decrypt.

M”, smaller:

$H(e, S'e)$

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

“NTRU KEM”,

obviously totally unrelated:

Client sends $k = H(c, e, Sc + e)$

encapsulated as $Sc + e$.

Random small

$c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;

public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.

Server recovers $3sc + te$,

then te mod 3, then e, then c.

Can imitate Niederreiter in the

NTRU context: e.g. “Ring-LWR”.

Client $\to$ server:

packet containing

(Combine with EC

Server $\to$ client:

packet containing

NS

$+ e)$

e

r:

.

"NTRU KEM",

obviously totally unrelated:

Client sends $k = H(c, e, Sc + e)$

encapsulated as $Sc + e$.

Random small

$c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;

public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.

Server recovers $3sc + te$,

then te mod 3, then e, then c.

Can imitate Niederreiter in the

NTRU context: e.g. "Ring-LWR".

Client $\rightarrow$ server:

packet containing $Sc+e, E_k$

(Combine with ECDH KEM

Server $\rightarrow$ client:

packet containing $E_k(1, r)$.

"NTRU KEM",
obviously totally unrelated:
Client sends $k = H(c, e, Sc + e)$
encapsulated as $Sc + e$.

Random small
$c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;
public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.
Server recovers $3sc + te$,
then $te \bmod 3$, then e, then c.

Can imitate Niederreiter in the
NTRU context: e.g. "Ring-LWR".

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

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

"NTRU KEM", obviously totally unrelated:
Client sends $k = H(c, e, Sc + e)$ encapsulated as $Sc + e$.

Random small $c, e \in (\mathbf{Z}/q)[x]/(x^n - 1)$;
public key $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

Secretly $S = 3s/t$; small s, t.
Server recovers $3sc + te$, then te mod 3, then e, then c.

Can imitate Niederreiter in the NTRU context: e.g. "Ring-LWR".

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.

KEM",
y totally unrelated:
ends $k = H(c, e, Sc + e)$
lated as $Sc + e$.

small
$\mathbf{Z}/q)[x]/(x^n - 1)$;
ey $S \in (\mathbf{Z}/q)[x]/(x^n - 1)$.

$S = 3s/t$; small s, t.
ecovers $3sc + te$,
mod 3, then e, then c.

tate Niederreiter in the
context: e.g. "Ring-LWR".

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.

Confiden
Attacker
can't de

Integrity
Server n
but E_k i
Attacker
but can'
Attacker

Availabi
Client di
continue
eventual

nrelated:

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

$c + e$.

$x^n - 1)$;

$/q)[x]/(x^n - 1)$.

; small s, t.

$c + te$,

en e, then c.

rreiter in the

g. “Ring-LWR”.

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 c

Attacker *can* repla

Availability:

Client discards forg

continues waiting

eventually retransr

$+ e)$

$- 1)$.

t.

c.

the

LWR".

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

Server → 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.

Cookies

What if
into sam

Client se
containi

Server se
cookie
k) encry
Server ca

Client se
Server re

Server se

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

<u>Cookies</u>

What if $E_k(0,q)$ d
into same packet a

Client sends short
containing a **cook**

Server sends $E_k(1$
cookie r': server s
k) encrypted from
Server can now for

Client sends packe
Server recovers sta

Server sends $E_k(3$

$_k(0, q)$.
.)

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.

Cookies

What if $E_k(0, q)$ doesn't fit
into same packet as $Sc + e$?

Client sends short $E_k(0, q')$
containing a **cookie request**

Server sends $E_k(1, r')$ conta
cookie r': server state (incl
k) encrypted from server to
Server can now forget state.

Client sends packet $r', E_k(2,$
Server recovers state, decryp

Server sends $E_k(3, r)$.

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.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet r', $E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

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.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client au

Same st

for prote

$C \rightarrow S$,

isn't spe

many pa

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

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

gery,
for reply,
mits request.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client authenticati

Same strategy wor
for protecting conn
$C \to S$, $S \to C$ da
isn't special; reuse
many packets each

, r).

ion.

ies

st.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Another TCP availability problem: server allocates buffers for each connection; runs out of memory.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Another TCP availability problem: server allocates buffers for each connection; runs out of memory.

Semi-solution: Allocate buffers only after client sends r'.

Cookies

What if $E_k(0, q)$ doesn't fit into same packet as $Sc + e$?

Client sends short $E_k(0, q')$ containing a **cookie request** q'.

Server sends $E_k(1, r')$ containing **cookie** r': server state (including k) encrypted from server to itself. Server can now forget state.

Client sends packet $r', E_k(2, q)$. Server recovers state, decrypts.

Server sends $E_k(3, r)$.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Another TCP availability problem: server allocates buffers for each connection; runs out of memory.

Semi-solution: Allocate buffers only after client sends r'.

Solution 1: Hashcash from client.

$E_k(0, q)$ doesn't fit
ne packet as $Sc + e$?

ends short $E_k(0, q')$
ng a **cookie request** q'.

ends $E_k(1, r')$ containing
r': server state (including
ypted from server to itself.
an now forget state.

ends packet $r', E_k(2, q)$.
ecovers state, decrypts.

ends $E_k(3, r)$.

Client authentication

Same strategy works
for protecting connections.
$C \to S$, $S \to C$ data flow
isn't special; reuse k for
many packets each direction.

Another TCP availability problem:
server allocates buffers for each
connection; runs out of memory.

Semi-solution: Allocate buffers
only after client sends r'.

Solution 1: Hashcash from client.

Solution
to avoid

Imitate

doesn't fit
as $Sc + e$?

$E_k(0, q')$
ie request q'.

$, r')$ containing
state (including
server to itself.
rget state.

et $r', E_k(2, q)$.
ate, decrypts.

$, r)$.

Client authentication

Same strategy works
for protecting connections.
$C \to S$, $S \to C$ data flow
isn't special; reuse k for
many packets each direction.

Another TCP availability problem:
server allocates buffers for each
connection; runs out of memory.

Semi-solution: Allocate buffers
only after client sends r'.

Solution 1: Hashcash from client.

Solution 2: Redo
to avoid state on

Imitate NFS, not

?

t q'.

ining
uding
itself.

, q).
ots.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Another TCP availability problem: server allocates buffers for each connection; runs out of memory.

Semi-solution: Allocate buffers only after client sends r'.

Solution 1: Hashcash from client.

Solution 2: Redo protocols to avoid state on server.

Imitate NFS, not HTTP.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Another TCP availability problem: server allocates buffers for each connection; runs out of memory.

Semi-solution: Allocate buffers only after client sends r'.

Solution 1: Hashcash from client.

Solution 2: Redo protocols to avoid state on server.

Imitate NFS, not HTTP.

Client authentication

Same strategy works for protecting connections. $C \to S$, $S \to C$ data flow isn't special; reuse k for many packets each direction.

Another TCP availability problem: server allocates buffers for each connection; runs out of memory.

Semi-solution: Allocate buffers only after client sends r'.

Solution 1: Hashcash from client.

Solution 2: Redo protocols to avoid state on server.

Imitate NFS, not HTTP.

Solution 3 for, e.g., SSH: Authenticate client.

Server can authenticate client without signatures, same way client authenticates server:

- Send to client's public key encapsulation of new key k'.
- Hash k' into shared secret.

uthentication

rategy works
ecting connections.
$S \to C$ data flow
ecial; reuse k for
ackets each direction.

TCP availability problem:
locates buffers for each
on; runs out of memory.

ution: Allocate buffers
er client sends r'.

1: Hashcash from client.

Solution 2: Redo protocols to avoid state on server.

Imitate NFS, not HTTP.

Solution 3 for, e.g., SSH: Authenticate client.

Server can authenticate client without signatures, same way client authenticates server:

- Send to client's public key encapsulation of new key k'.
- Hash k' into shared secret.

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

ion

rks
nections.
ata flow
k for
n direction.

lability problem:
uffers for each
out of memory.

ocate buffers
ends r'.

ash from client.

Solution 2: Redo protocols
to avoid state on server.

Imitate NFS, not HTTP.

Solution 3 for, e.g., SSH:
Authenticate client.

Server can authenticate client
without signatures, same way
client authenticates server:

- Send to client's public key encapsulation of new key k'.
- Hash k' into shared secret.

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

n.

oblem:
ach
nory.

fers

client.

Solution 2: Redo protocols
to avoid state on server.

Imitate NFS, not HTTP.

Solution 3 for, e.g., SSH:
Authenticate client.

Server can authenticate client
without signatures, same way
client authenticates server:

- Send to client's public key encapsulation of new key k'.
- Hash k' into shared secret.

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-MDPC

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

Solution 2: Redo protocols to avoid state on server.

Imitate NFS, not HTTP.

Solution 3 for, e.g., SSH: Authenticate client.

Server can authenticate client without signatures, same way client authenticates server:

- Send to client's public key encapsulation of new key k'.
- Hash k' into shared secret.

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.

2: Redo protocols
state on server.

NFS, not HTTP.

3 for, e.g., SSH:
icate client.

an authenticate client
signatures, same way
uthenticates server:
to client's public key
sulation of new key k'.
k' into shared secret.

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

protocols
server.

HTTP.

., SSH:
t.

ticate client
, same way
es server:
public key
f new key k'.
ared secret.

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: “forwar
subsequent theft o
doesn’t allow decr

e.g. Microsoft SCh
switches keys ever

Safer: new key eve

Easier to implemen
new key every con

nt
ny
y
k'.
.

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
000: 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, ≤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, $\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.

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, $\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 doe
encrypt
without

mitation
keys:
keys
sing old keys.

rd secrecy"—
of computer
yption.

hannel
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 c
without storing th

keys.

'—

er

rs.

e.

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 *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.