# The DNS security mess

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

---

## Paul Vixie, 1995, on DNSSEC:

This sounds simple but it has deep reaching consequences in both the protocol and the implementation—which is why it's taken more than a year to choose a security model and design a solution. We expect it to be another year before DNSSEC is in wide use on the leading edge, and at least a year after that before its use is commonplace on the Internet.

---

Before I start my talk,
some comments on HTTPSEC.
Warning: HTTPSEC $\neq$ HTTPS.

# The DNS security mess

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

---

## Paul Vixie, 1995, on DNSSEC:

This sounds simple but it has deep reaching consequences in both the protocol and the implementation—which is why it's taken more than a year to choose a security model and design a solution. We expect it to be another year before DNSSEC is in wide use on the leading edge, and at least a year after that before its use is commonplace on the Internet.

---

Before I start my talk,
some comments on HTTPSEC.
Warning: HTTPSEC $\neq$ HTTPS.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

# The DNS security mess

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

---

Paul Vixie, 1995, on DNSSEC:

> This sounds simple but it has deep reaching consequences in both the protocol and the implementation—which is why it's taken more than a year to choose a security model and design a solution. We expect it to be another year before DNSSEC is in wide use on the leading edge, and at least a year after that before its use is commonplace on the Internet.

---

Before I start my talk,
some comments on HTTPSEC.
Warning: HTTPSEC $\neq$ HTTPS.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using
a wireless network that
also has many other computers.
Some of those computers are
*controlled by attackers*.

# The DNS security mess

D. J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

---

Paul Vixie, 1995, on DNSSEC:

> This sounds simple but it has deep reaching consequences in both the protocol and the implementation—which is why it's taken more than a year to choose a security model and design a solution. We expect it to be another year before DNSSEC is in wide use on the leading edge, and at least a year after that before its use is commonplace on the Internet.

---

Before I start my talk,
some comments on HTTPSEC.
Warning: HTTPSEC $\neq$ HTTPS.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using
a wireless network that
also has many other computers.
Some of those computers are
*controlled by attackers*.

Or maybe you're in Iran, and
the network *is* the attacker.

S security mess

ernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

---

kie, 1995, on DNSSEC:

s simple but it has deep reaching
es in both the protocol and the
tion—which is why it's taken more
r to choose a security model and
lution. We expect it to be another
DNSSEC is in wide use on the
e, and at least a year after that
se is commonplace on the Internet.

---

start my talk,
mments on HTTPSEC.
: HTTPSEC $\neq$ HTTPS.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using
a wireless network that
also has many other computers.
Some of those computers are
*controlled by attackers*.

Or maybe you're in Iran, and
the network *is* the attacker.

Standard

**Confide**
despite e
**Integrit**
despite c
**Availabi**

mess

is at Chicago &
siteit Eindhoven

---

on DNSSEC:

t has deep reaching
e protocol and the
s why it's taken more
security model and
xpect it to be another
n wide use on the
st a year after that
nplace on the Internet.

---

talk,
n HTTPSEC.
EC ≠ HTTPS.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using a wireless network that also has many other computers.
Some of those computers are *controlled by attackers*.

Or maybe you're in Iran, and the network *is* the attacker.

Standard security

**Confidentiality** (p
despite espionage.
**Integrity** (authent
despite corruption
**Availability** despit

ago &
hoven

EC:

aching
l the
en more
el and
another
n the
that
Internet.

EC.
TPS.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using a wireless network that also has many other computers.
Some of those computers are *controlled by attackers*.

Or maybe you're in Iran, and the network *is* the attacker.

Standard security goals:

**Confidentiality** (privacy etc
despite espionage.
**Integrity** (authenticity etc.)
despite corruption.
**Availability** despite sabotag

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using a wireless network that also has many other computers.
Some of those computers are *controlled by attackers*.

Or maybe you're in Iran, and the network *is* the attacker.

Standard security goals:

**Confidentiality** (privacy etc.) despite espionage.
**Integrity** (authenticity etc.) despite corruption.
**Availability** despite sabotage.

## HTTPSEC motivation

You use HTTP all the time:
e.g., `http://nu.nl`.

Your computer requests a web page from the `nu.nl` server.
The server sends a web page.

Your computer is using a wireless network that also has many other computers.
Some of those computers are *controlled by attackers*.

Or maybe you're in Iran, and the network *is* the attacker.

Standard security goals:

**Confidentiality** (privacy etc.) despite espionage.
**Integrity** (authenticity etc.) despite corruption.
**Availability** despite sabotage.

HTTP provides none of this.

By watching the network, attacker easily **acquires** data: the HTTP request, the web page.
Attacker easily **changes** data.
Attacker easily **destroys** data.

## EC motivation

HTTP all the time:
`tp://nu.nl`.

mputer requests a web
m the `nu.nl` server.
ver sends a web page.

mputer is using
ss network that
many other computers.
those computers are
*ed by attackers*.

be you're in Iran, and
*work is* the attacker.

Standard security goals:

**Confidentiality** (privacy etc.)
despite espionage.
**Integrity** (authenticity etc.)
despite corruption.
**Availability** despite sabotage.

HTTP provides none of this.

By watching the network,
attacker easily **acquires** data:
the HTTP request, the web page.
Attacker easily **changes** data.
Attacker easily **destroys** data.

## HTTPSE

HTTPSE
to "bols

HTTPSE
for the `n`
to attac
to the `n`

These si
"verifica
authenti
data" ob

## tion

the time:
nl.

quests a web
.nl server.
a web page.

using
that
er computers.
mputers are
*ckers*.

n Iran, and
attacker.

Standard security goals:

**Confidentiality** (privacy etc.)
despite espionage.
**Integrity** (authenticity etc.)
despite corruption.
**Availability** despite sabotage.

HTTP provides none of this.

By watching the network,
attacker easily **acquires** data:
the HTTP request, the web page.
Attacker easily **changes** data.
Attacker easily **destroys** data.

## HTTPSEC: "HTT

HTTPSEC modifi
to "bolster online

HTTPSEC provide
for the nu.nl serv
to attach **PGP sig**
to the nu.nl HTT

These signatures a
"verification of the
authenticity, and i
data" obtained th

eb
.
e.

ters.
e

d

Standard security goals:

**Confidentiality** (privacy etc.)
despite espionage.
**Integrity** (authenticity etc.)
despite corruption.
**Availability** despite sabotage.

HTTP provides none of this.

By watching the network,
attacker easily **acquires** data:
the HTTP request, the web page.
Attacker easily **changes** data.
Attacker easily **destroys** data.

HTTPSEC: "HTTP Security

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP respons

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HT

Standard security goals:

**Confidentiality** (privacy etc.)
despite espionage.
**Integrity** (authenticity etc.)
despite corruption.
**Availability** despite sabotage.

HTTP provides none of this.

By watching the network,
attacker easily **acquires** data:
the HTTP request, the web page.
Attacker easily **changes** data.
Attacker easily **destroys** data.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

d security goals:

**ntiality** (privacy etc.)
espionage.
**y** (authenticity etc.)
corruption.
**ility** despite sabotage.

provides none of this.

hing the network,
easily **acquires** data:
TP request, the web page.
r easily **changes** data.
r easily **destroys** data.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

To verify
your con
retrieve
from the

goals:

privacy etc.)

ticity etc.)

.

te sabotage.

one of this.

network,

**quires** data:

t, the web page.

**anges** data.

**stroys** data.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

To verify these sig
your computer nee
retrieve the PGP p
from the `nu.nl` ac

.)

e.

.

a:

page.

a.

ta.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

To verify these signatures,
your computer needs to
retrieve the PGP public key
from the `nu.nl` admin.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

To verify these signatures,
your computer needs to
retrieve the PGP public key
from the `nu.nl` admin.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

To verify these signatures,
your computer needs to
retrieve the PGP public key
from the `nu.nl` admin.

What if the *key* is forged?

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP
to "bolster online security".

HTTPSEC provides a way
for the `nu.nl` server admin
to attach **PGP signatures**
to the `nu.nl` HTTP responses.

These signatures allow
"verification of the origin,
authenticity, and integrity of
data" obtained through HTTP.

To verify these signatures,
your computer needs to
retrieve the PGP public key
from the `nu.nl` admin.

What if the *key* is forged?

Answer: HTTPSEC provides a
way for a trusted Netherlands
government representative to
PGP-sign the `nu.nl` public key.

## HTTPSEC: "HTTP Security"

HTTPSEC modifies HTTP to "bolster online security".

HTTPSEC provides a way for the `nu.nl` server admin to attach **PGP signatures** to the `nu.nl` HTTP responses.

These signatures allow "verification of the origin, authenticity, and integrity of data" obtained through HTTP.

To verify these signatures, your computer needs to retrieve the PGP public key from the `nu.nl` admin.

What if the *key* is forged?

Answer: HTTPSEC provides a way for a trusted Netherlands government representative to PGP-sign the `nu.nl` public key.

What if *that* key is forged?
Answer: Internet Central Headquarters signed the Netherlands public key.

EC: "HTTP Security"

EC modifies HTTP
ter online security".

EC provides a way
nu.nl server admin
h **PGP signatures**
u.nl HTTP responses.

gnatures allow
tion of the origin,
city, and integrity of
btained through HTTP.

To verify these signatures, your computer needs to retrieve the PGP public key from the `nu.nl` admin.

What if the *key* is forged?

Answer: HTTPSEC provides a way for a trusted Netherlands government representative to PGP-sign the `nu.nl` public key.

What if *that* key is forged?
Answer: Internet Central Headquarters signed the Netherlands public key.

Internet
was gen
Hardwar
owned b
a well-kn

Hardwar
signs da
by 3 out
held by

3 VeriSig
meet eve
they hav

TP Security"

es HTTP
security".

es a way
ver admin
**gnatures**
TP responses.

allow
e origin,
ntegrity of
rough HTTP.

To verify these signatures, your computer needs to retrieve the PGP public key from the `nu.nl` admin.

What if the *key* is forged?

Answer: HTTPSEC provides a way for a trusted Netherlands government representative to PGP-sign the `nu.nl` public key.

What if *that* key is forged?
Answer: Internet Central Headquarters signed the Netherlands public key.

Internet Central H
was generated by
Hardware Security
owned by VeriSign
a well-known Ame

Hardware Security
signs data if autho
by 3 out of 16 sm
held by VeriSign T

3 VeriSign Trust M
meet every week i
they have to sign

y"

ses.

f

TP.

To verify these signatures,
your computer needs to
retrieve the PGP public key
from the `nu.nl` admin.

What if the *key* is forged?

Answer: HTTPSEC provides a
way for a trusted Netherlands
government representative to
PGP-sign the `nu.nl` public key.

What if *that* key is forged?
Answer: Internet Central
Headquarters signed the
Netherlands public key.

Internet Central HQ key
was generated by an expensi
Hardware Security Module
owned by VeriSign,
a well-known American com

Hardware Security Module
signs data if authorized
by 3 out of 16 smart cards
held by VeriSign Trust Mana

3 VeriSign Trust Managers
meet every week in case
they have to sign new data.

To verify these signatures,
your computer needs to
retrieve the PGP public key
from the `nu.nl` admin.

What if the *key* is forged?

Answer: HTTPSEC provides a
way for a trusted Netherlands
government representative to
PGP-sign the `nu.nl` public key.

What if *that* key is forged?
Answer: Internet Central
Headquarters signed the
Netherlands public key.

Internet Central HQ key
was generated by an expensive
Hardware Security Module
owned by VeriSign,
a well-known American company.

Hardware Security Module
signs data if authorized
by 3 out of 16 smart cards
held by VeriSign Trust Managers.

3 VeriSign Trust Managers
meet every week in case
they have to sign new data.

y these signatures,
mputer needs to
the PGP public key
e `nu.nl` admin.

the *key* is forged?

HTTPSEC provides a
a trusted Netherlands
ent representative to
n the `nu.nl` public key.

*that* key is forged?
Internet Central
arters signed the
nds public key.

Internet Central HQ key
was generated by an expensive
Hardware Security Module
owned by VeriSign,
a well-known American company.

Hardware Security Module
signs data if authorized
by 3 out of 16 smart cards
held by VeriSign Trust Managers.

3 VeriSign Trust Managers
meet every week in case
they have to sign new data.

If your c
software
Internet

Your co
the Neth
and the
signatur
PGP-ver

Next ste
the `nu.n`
and the

Finally F
HTTPSE

gnatures,
eds to
public key
dmin.

forged?

C provides a
Netherlands
sentative to
nl public key.

s forged?
Central
ed the
c key.

Internet Central HQ key
was generated by an expensive
Hardware Security Module
owned by VeriSign,
a well-known American company.

Hardware Security Module
signs data if authorized
by 3 out of 16 smart cards
held by VeriSign Trust Managers.

3 VeriSign Trust Managers
meet every week in case
they have to sign new data.

If your computer h
software then it al
Internet Central H

Your computer ret
the Netherlands pu
and the Internet C
signature of that p
PGP-verifies this s

Next step: retrieve
the `nu.nl` admin's
and the Netherlan

Finally PGP-verify
HTTPSEC-signed

s a
ds
o
key.

Internet Central HQ key
was generated by an expensive
Hardware Security Module
owned by VeriSign,
a well-known American company.

Hardware Security Module
signs data if authorized
by 3 out of 16 smart cards
held by VeriSign Trust Managers.

3 VeriSign Trust Managers
meet every week in case
they have to sign new data.

If your computer has HTTP
software then it already kno
Internet Central HQ public k

Your computer retrieves
the Netherlands public key
and the Internet Central HQ
signature of that public key;
PGP-verifies this signature.

Next step: retrieve and verif
the `nu.nl` admin's public ke
and the Netherlands signatu

Finally PGP-verify `nu.nl`'s
HTTPSEC-signed responses

Internet Central HQ key
was generated by an expensive
Hardware Security Module
owned by VeriSign,
a well-known American company.

Hardware Security Module
signs data if authorized
by 3 out of 16 smart cards
held by VeriSign Trust Managers.

3 VeriSign Trust Managers
meet every week in case
they have to sign new data.

If your computer has HTTPSEC
software then it already knows the
Internet Central HQ public key.

Your computer retrieves
the Netherlands public key
and the Internet Central HQ
signature of that public key;
PGP-verifies this signature.

Next step: retrieve and verify
the `nu.nl` admin's public key
and the Netherlands signature.

Finally PGP-verify `nu.nl`'s
HTTPSEC-signed responses.

Central HQ key
erated by an expensive
e Security Module
y VeriSign,
nown American company.

e Security Module
ta if authorized
of 16 smart cards
VeriSign Trust Managers.

gn Trust Managers
ery week in case
ve to sign new data.

If your computer has HTTPSEC
software then it already knows the
Internet Central HQ public key.

Your computer retrieves
the Netherlands public key
and the Internet Central HQ
signature of that public key;
PGP-verifies this signature.

Next step: retrieve and verify
the `nu.nl` admin's public key
and the Netherlands signature.

Finally PGP-verify `nu.nl`'s
HTTPSEC-signed responses.

HTTPS

Many In
are extre
Can they

The crit
in HTTP
PGP sig
"Per-que

Signatur
saved; se
Hopefull
sign eac

Q key
an expensive
Module
,
erican company.

Module
orized
art cards
Trust Managers.

Managers
n case
new data.

If your computer has HTTPSEC
software then it already knows the
Internet Central HQ public key.

Your computer retrieves
the Netherlands public key
and the Internet Central HQ
signature of that public key;
PGP-verifies this signature.

Next step: retrieve and verify
the `nu.nl` admin's public key
and the Netherlands signature.

Finally PGP-verify `nu.nl`'s
HTTPSEC-signed responses.

HTTPSEC perform

Many Internet serv
are extremely busy
Can they afford cr

The critical design
in HTTPSEC: *pre*
PGP signatures of
"Per-query crypto

Signature is compu
saved; sent to mar
Hopefully the adm
sign each HTTP r

ive

pany.

agers.

If your computer has HTTPSEC software then it already knows the Internet Central HQ public key.

Your computer retrieves the Netherlands public key and the Internet Central HQ signature of that public key; PGP-verifies this signature.

Next step: retrieve and verify the `nu.nl` admin's public key and the Netherlands signature.

Finally PGP-verify `nu.nl`'s HTTPSEC-signed responses.

## HTTPSEC performance

Many Internet servers are extremely busy. Can they afford crypto?

The critical design decision in HTTPSEC: *precompute* PGP signatures of all data. "Per-query crypto is bad."

Signature is computed once; saved; sent to many clients. Hopefully the admin can aff

sign each HTTP response o

If your computer has HTTPSEC software then it already knows the Internet Central HQ public key.

Your computer retrieves the Netherlands public key and the Internet Central HQ signature of that public key; PGP-verifies this signature.

Next step: retrieve and verify the `nu.nl` admin's public key and the Netherlands signature.

Finally PGP-verify `nu.nl`'s HTTPSEC-signed responses.

## HTTPSEC performance

Many Internet servers are extremely busy. Can they afford crypto?

The critical design decision in HTTPSEC: *precompute* PGP signatures of all data. "Per-query crypto is bad."

Signature is computed once; saved; sent to many clients. Hopefully the admin can afford to sign each HTTP response once.

computer has HTTPSEC
then it already knows the
Central HQ public key.

mputer retrieves
herlands public key
Internet Central HQ
e of that public key;
rifies this signature.

ep: retrieve and verify
nl admin's public key
Netherlands signature.

PGP-verify nu.nl's
EC-signed responses.

## HTTPSEC performance

Many Internet servers
are extremely busy.
Can they afford crypto?

The critical design decision
in HTTPSEC: *precompute*
PGP signatures of all data.
"Per-query crypto is bad."

Signature is computed once;
saved; sent to many clients.
Hopefully the admin can afford to
sign each HTTP response once.

Clients c
of *verify*

HTTPSE
client-sid
precomp
choice o

Many H
640-bit
768-bit
1024-bit
signatur
DSA, "1
verificati

has HTTPSEC
ready knows the
Q public key.

trieves
ublic key
Central HQ
public key;
signature.

e and verify
s public key
ds signature.

`nu.nl`'s
responses.

## HTTPSEC performance

Many Internet servers
are extremely busy.
Can they afford crypto?

The critical design decision
in HTTPSEC: *precompute*
PGP signatures of all data.
"Per-query crypto is bad."

Signature is computed once;
saved; sent to many clients.
Hopefully the admin can afford to
sign each HTTP response once.

Clients don't share
of *verifying* a sign

HTTPSEC tries to
client-side costs (a
precomputation co
choice of crypto p

Many HTTPSEC
640-bit RSA, origi
768-bit RSA, man
1024-bit RSA, cur
signatures from Ve
DSA, "10 to 40 ti
verification" but fa

SEC
ws the
key.
Q
fy
ey
re.
.

## HTTPSEC performance

Many Internet servers
are extremely busy.
Can they afford crypto?

The critical design decision
in HTTPSEC: *precompute*
PGP signatures of all data.
"Per-query crypto is bad."

Signature is computed once;
saved; sent to many clients.
Hopefully the admin can afford to
sign each HTTP response once.

Clients don't share the work
of *verifying* a signature.

HTTPSEC tries to reduce
client-side costs (and
precomputation costs) throu
choice of crypto primitive.

Many HTTPSEC crypto opt
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current
signatures from VeriSign etc
DSA, "10 to 40 times as slo
verification" but faster for si

## HTTPSEC performance

Many Internet servers
are extremely busy.
Can they afford crypto?

The critical design decision
in HTTPSEC: *precompute*
PGP signatures of all data.
"Per-query crypto is bad."

Signature is computed once;
saved; sent to many clients.
Hopefully the admin can afford to
sign each HTTP response once.

Clients don't share the work
of *verifying* a signature.

HTTPSEC tries to reduce
client-side costs (and
precomputation costs) through
choice of crypto primitive.

Many HTTPSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current
signatures from VeriSign etc.;
DSA, "10 to 40 times as slow for
verification" but faster for signing.

EC performance

ternet servers
emely busy.
y afford crypto?

ical design decision
PSEC: *precompute*
natures of all data.
ery crypto is bad."

re is computed once;
ent to many clients.
y the admin can afford to
h HTTP response once.

Clients don't share the work of *verifying* a signature.

HTTPSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many HTTPSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current signatures from VeriSign etc.;
DSA, "10 to 40 times as slow for verification" but faster for signing.

HTTPS
choices
for no re
fear of o

HTTPS
to surviv
and even
for reaso
More co
including

Author o
HTTP s
impleme
is just st

mance

vers

.

ypto?

decision

*compute*

all data.

is bad."

uted once;

ny clients.

in can afford to

esponse once.

Clients don't share the work of *verifying* a signature.

HTTPSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many HTTPSEC crypto options:
640-bit RSA, original specs;
768-bit RSA, many docs;
1024-bit RSA, current signatures from VeriSign etc.;
DSA, "10 to 40 times as slow for verification" but faster for signing.

HTTPSEC made

choices such as 64

for no reason othe

fear of overload.

HTTPSEC needed

to survive the inev

and even more co

for reasons I'll exp

More complexity =

including security

Author of one very

HTTP server: "T

implementing ever

is just staggering."

Clients don't share the work of *verifying* a signature.

HTTPSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many HTTPSEC crypto options: 640-bit RSA, original specs; 768-bit RSA, many docs; 1024-bit RSA, current signatures from VeriSign etc.; DSA, "10 to 40 times as slow for verification" but faster for signing.

HTTPSEC made breakable choices such as 640-bit RSA for no reason other than fear of overload.

HTTPSEC needed more opt to survive the inevitable brea and even more complexity for reasons I'll explain. More complexity ⇒ more bu including security holes.

Author of one very popular HTTP server: "The effort o implementing everything cor is just staggering."

Clients don't share the work of *verifying* a signature.

HTTPSEC tries to reduce client-side costs (and precomputation costs) through choice of crypto primitive.

Many HTTPSEC crypto options: 640-bit RSA, original specs; 768-bit RSA, many docs; 1024-bit RSA, current signatures from VeriSign etc.; DSA, "10 to 40 times as slow for verification" but faster for signing.

HTTPSEC made breakable choices such as 640-bit RSA for no reason other than fear of overload.

HTTPSEC needed more options to survive the inevitable breaks; and even more complexity for reasons I'll explain. More complexity ⇒ more bugs, including security holes.

Author of one very popular HTTP server: "The effort of implementing everything correctly is just staggering."

don't share the work
*ing* a signature.

EC tries to reduce
de costs (and
putation costs) through
f crypto primitive.

TTPSEC crypto options:
RSA, original specs;
RSA, many docs;
RSA, current
es from VeriSign etc.;
10 to 40 times as slow for
ion" but faster for signing.

HTTPSEC made breakable
choices such as 640-bit RSA
for no reason other than
fear of overload.

HTTPSEC needed more options
to survive the inevitable breaks;
and even more complexity
for reasons I'll explain.
More complexity ⇒ more bugs,
including security holes.

Author of one very popular
HTTP server: "The effort of
implementing everything correctly
is just staggering."

HTTPSI

How do
requests
without

e the work
ature.

o reduce
and
osts) through
rimitive.

crypto options:
nal specs;
y docs;
rent
eriSign etc.;
mes as slow for
aster for signing.

HTTPSEC made breakable
choices such as 640-bit RSA
for no reason other than
fear of overload.

HTTPSEC needed more options
to survive the inevitable breaks;
and even more complexity
for reasons I'll explain.
More complexity ⇒ more bugs,
including security holes.

Author of one very popular
HTTP server: "The effort of
implementing everything correctly
is just staggering."

HTTPSEC confide

How do you *encry*
requests and respo
without per-client

ugh

tions:

c.;

ow for

igning.

HTTPSEC made breakable
choices such as 640-bit RSA
for no reason other than
fear of overload.

HTTPSEC needed more options
to survive the inevitable breaks;
and even more complexity
for reasons I'll explain.
More complexity ⇒ more bugs,
including security holes.

Author of one very popular
HTTP server: "The effort of
implementing everything correctly
is just staggering."

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

HTTPSEC made breakable
choices such as 640-bit RSA
for no reason other than
fear of overload.

HTTPSEC needed more options
to survive the inevitable breaks;
and even more complexity
for reasons I'll explain.
More complexity ⇒ more bugs,
including security holes.

Author of one very popular
HTTP server: "The effort of
implementing everything correctly
is just staggering."

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

HTTPSEC made breakable
choices such as 640-bit RSA
for no reason other than
fear of overload.

HTTPSEC needed more options
to survive the inevitable breaks;
and even more complexity
for reasons I'll explain.
More complexity ⇒ more bugs,
including security holes.

Author of one very popular
HTTP server: "The effort of
implementing everything correctly
is just staggering."

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

Answer: You can't,
and HTTPSEC doesn't even try.

The HTTPSEC RFC says
"Due to a deliberate design
choice, HTTPSEC does not
provide confidentiality."

HTTPSEC made breakable
choices such as 640-bit RSA
for no reason other than
fear of overload.

HTTPSEC needed more options
to survive the inevitable breaks;
and even more complexity
for reasons I'll explain.
More complexity ⇒ more bugs,
including security holes.

Author of one very popular
HTTP server: "The effort of
implementing everything correctly
is just staggering."

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

Answer: You can't,
and HTTPSEC doesn't even try.

The HTTPSEC RFC says
"Due to a deliberate design
choice, HTTPSEC does not
provide confidentiality."

This is very strange, but
not the worst part of HTTPSEC.

EC made breakable
such as 640-bit RSA
eason other than
verload.

EC needed more options
ve the inevitable breaks;
n more complexity
ons I'll explain.
mplexity ⇒ more bugs,
g security holes.

of one very popular
erver: "The effort of
nting everything correctly
taggering."

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

Answer: You can't,
and HTTPSEC doesn't even try.

The HTTPSEC RFC says
"Due to a deliberate design
choice, HTTPSEC does not
provide confidentiality."

This is very strong, but
not the worst part of HTTPSEC.

## The HT

When `nu`
receives
`http://`
it looks
`/var/ww`
on its lo

An HTT
`http://`
`index.h`
Server a
`index.h`
with a si

breakable
-0-bit RSA
r than

l more options
vitable breaks;
mplexity
lain.
⇒ more bugs,
holes.

y popular
he effort of
rything correctly
"

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

Answer: You can't,
and HTTPSEC doesn't even try.

The HTTPSEC RFC says
"Due to a deliberate design
choice, HTTPSEC does not
provide confidentiality."

This is very strange, but
not the worst part of HTTPSEC.

## The HTTPSEC da

When `nu.nl` HTT
receives a request
`http://nu.nl/ec`
it looks for a file
`/var/www/econom`
on its local disk.

An HTTPSEC clie
`http://nu.nl/ec`
`index.html.http`
Server admin has
`index.html.http`
with a signature o

tions
aks;

ugs,

f
rrectly

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

Answer: You can't,
and HTTPSEC doesn't even try.

The HTTPSEC RFC says
"Due to a deliberate design
choice, HTTPSEC does not
provide confidentiality."

This is very strange, but
not the worst part of HTTPSEC.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index`
on its local disk.

An HTTPSEC client also as
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.h`

## HTTPSEC confidentiality

How do you *encrypt*
requests and responses
without per-client crypto?

Answer: You can't,
and HTTPSEC doesn't even try.

The HTTPSEC RFC says
"Due to a deliberate design
choice, HTTPSEC does not
provide confidentiality."

This is very strange, but
not the worst part of HTTPSEC.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

## EC confidentiality

you *encrypt*
and responses
per-client crypto?

You can't,
TPSEC doesn't even try.

TPSEC RFC says
a deliberate design
HTTPSEC does not
confidentiality."

very strange, but
worst part of HTTPSEC.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There ar
of softw
admins
e.g., wik

entiality

*pt*

onses

crypto?

t,

oesn't even try.

FC says

ate design

does not

ality."

ge, but

of HTTPSEC.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There are hundred
of software tools t
admins manage we
e.g., wiki-creation

n try.

SEC.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There are hundreds (thousa
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
`index.html`,
do they also create
`index.html.httpsec-pgp`?

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
`index.html`,
do they also create
`index.html.httpsec-pgp`?

What about *dynamic* data?

## The HTTPSEC data model

When `nu.nl` HTTP server
receives a request for
`http://nu.nl/economie/`,
it looks for a file
`/var/www/economie/index.html`
on its local disk.

An HTTPSEC client also asks for
`http://nu.nl/economie/`
`index.html.httpsec-pgp`.
Server admin has created
`index.html.httpsec-pgp`
with a signature of `index.html`.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
`index.html`,
do they also create
`index.html.httpsec-pgp`?

What about *dynamic* data?

HTTPSEC purists say "Answers should always be static".

## TPSEC data model

u.nl HTTP server
a request for
/nu.nl/economie/,
for a file
ww/economie/index.html
cal disk.

TPSEC client also asks for
/nu.nl/economie/
html.httpsec-pgp.
dmin has created
html.httpsec-pgp
ignature of index.html.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
index.html,
do they also create
index.html.httpsec-pgp?

What about *dynamic* data?

HTTPSEC purists say "Answers should always be static".

What ab
Are the

Can an
obsolete

If clocks
then sig
include
But freq
is an ad

HTTPS
admin s
expire; e
refuses t

ata model

TP server
for
conomie/,

mie/index.html

ent also asks for
conomie/
psec-pgp.
created
psec-pgp
f index.html.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
index.html,
do they also create
index.html.httpsec-pgp?

What about *dynamic* data?

HTTPSEC purists say "Answers should always be static".

What about *old* d
Are the signatures

Can an attacker re
obsolete signed da

If clocks are synch
then signatures ca
include expiration
But frequent re-sig
is an administrativ

HTTPSEC suicide
admin screws up;
expire; every HTT
refuses to load the

x.html

ks for

tml.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
`index.html`,
do they also create
`index.html.httpsec-pgp`?

What about *dynamic* data?

HTTPSEC purists say "Answers should always be static".

What about *old* data?
Are the signatures still valid

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC clie
refuses to load the page.

There are hundreds (thousands?)
of software tools to help
admins manage web sites:
e.g., wiki-creation tools.

When these tools create
`index.html`,
do they also create
`index.html.httpsec-pgp`?

What about *dynamic* data?

HTTPSEC purists say "Answers should always be static".

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

re hundreds (thousands?)
are tools to help
manage web sites:
i-creation tools.

hese tools create
`html`,
also create
`html.httpsec-pgp`?

bout *dynamic* data?

EC purists say "Answers
lways be static".

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSE

2010.09.

2010.10.

ls (thousands?)
o help
eb sites:
tools.

create

e
psec-pgp?

*mic* data?

say "Answers
static".

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSEC suicide

2010.09.02: US go

2010.10.07: Belgia

nds?)

?

wers

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSEC suicide examples

2010.09.02: US government

2010.10.07: Belgian governm

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

What about *old* data?
Are the signatures still valid?

Can an attacker replay
obsolete signed data?

If clocks are synchronized
then signatures can
include expiration times.
But frequent re-signing
is an administrative disaster.

HTTPSEC suicide:
admin screws up; signatures
expire; every HTTPSEC client
refuses to load the page.

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF
tech-support rep: "`httpsec-accept-expired yes`"

bout *old* data?
signatures still valid?

attacker replay
signed data?

are synchronized
natures can
expiration times.
uent re-signing
ministrative disaster.

EC suicide:
crews up; signatures
very HTTPSEC client
o load the page.

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What ab

ata?
still valid?

eplay
ta?

ronized
n
times.
gning
ve disaster.
:
signatures
PSEC client
e page.

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What about *nonex*

?

nt

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What about *nonexistent* file

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What about *nonexistent* files?

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What about *nonexistent* files?

Does the server admin precompute PGP signatures on "`aaaaa` does not exist", "`aaaab` does not exist", etc.?

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What about *nonexistent* files?

Does the server admin precompute PGP signatures on "`aaaaa` does not exist", "`aaaab` does not exist", etc.?

Crazy! Obvious approach: "We sign each page that exists, and don't sign anything else."

HTTPSEC suicide examples:

2010.09.02: US government.

2010.10.07: Belgian government.

2012.02.23: `httpsec-ref.org`.

2012.02.28: "`Last night I was unable to check the weather forecast, because the fine folks at NOAA.gov / weather.gov broke their HTTPSEC.`"

2012.02.28, HTTPSEC-REF tech-support rep: "`httpsec-accept-expired yes`"

What about *nonexistent* files?

Does the server admin precompute PGP signatures on "`aaaaa` does not exist", "`aaaab` does not exist", etc.?

Crazy! Obvious approach: "We sign each page that exists, and don't sign anything else."

User asks for nonexistent page. Receives *unsigned* answer saying the page doesn't exist. Has no choice but to trust it.

EC suicide examples:

02: US government.

07: Belgian government.

23: `httpsec-ref.org`.

28: "`Last night I
ble to check the
r forecast, because
e folks at NOAA.gov
er.gov broke their
C.`"

28, HTTPSEC-REF
pport rep: "`httpsec-
-expired yes`"

What about *nonexistent* files?

Does the server admin
precompute PGP signatures on
"`aaaaa` does not exist",
"`aaaab` does not exist", etc.?

Crazy! Obvious approach:
"We sign each page that exists,
and don't sign anything else."

User asks for nonexistent page.
Receives *unsigned* answer
saying the page doesn't exist.
Has no choice but to trust it.

User ask
Receives
a respon
saying th
Has no c

Clearly a
Sometim
This is n

examples:

overnment.

an government.

sec-ref.org.

t night I
heck the
st, because
at NOAA.gov
roke their

PSEC-REF
"httpsec-
yes"

What about *nonexistent* files?

Does the server admin
precompute PGP signatures on
"aaaaa does not exist",
"aaaab does not exist", etc.?

Crazy! Obvious approach:
"We sign each page that exists,
and don't sign anything else."

User asks for nonexistent page.
Receives *unsigned* answer
saying the page doesn't exist.
Has no choice but to trust it.

User asks for nu
Receives unsigned
a response forged
saying the page do
Has no choice but

Clearly a violation
Sometimes a viola
This is not a good

:

.

ment.

org.

se

gov

ir

-

:-

What about *nonexistent* files?

Does the server admin
precompute PGP signatures on
"aaaaa does not exist",
"aaaab does not exist", etc.?

Crazy! Obvious approach:
"We sign each page that exists,
and don't sign anything else."

User asks for nonexistent page.
Receives *unsigned* answer
saying the page doesn't exist.
Has no choice but to trust it.

User asks for nu.nl/econom
Receives unsigned answer,
a response forged by attacke
saying the page doesn't exis
Has no choice but to trust i

Clearly a violation of availab
Sometimes a violation of int
This is not a good approach

What about *nonexistent* files?

Does the server admin
precompute PGP signatures on
"`aaaaa` does not exist",
"`aaaab` does not exist", etc.?

Crazy! Obvious approach:
"We sign each page that exists,
and don't sign anything else."

User asks for nonexistent page.
Receives *unsigned* answer
saying the page doesn't exist.
Has no choice but to trust it.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

What about *nonexistent* files?

Does the server admin
precompute PGP signatures on
"`aaaaa` does not exist",
"`aaaab` does not exist", etc.?

Crazy! Obvious approach:
"We sign each page that exists,
and don't sign anything else."

User asks for nonexistent page.
Receives *unsigned* answer
saying the page doesn't exist.
Has no choice but to trust it.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: "NHTTPSEC". e.g.
`clegg.com/nonex` query returns
"`There are no pages between`
`clegg.com/nick` and
`clegg.com/start`" + signature.

bout *nonexistent* files?

e server admin
bute PGP signatures on
does not exist",
does not exist", etc.?

Obvious approach:
n each page that exists,
't sign anything else."

ks for nonexistent page.
s *unsigned* answer
he page doesn't exist.
choice but to trust it.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: "NHTTPSEC". e.g.
`clegg.com/nonex` query returns
"`There are no pages between`
`clegg.com/nick` and
`clegg.com/start`" + signature.

Try `cleg`
After sev
all `cleg`
`alan`, `al`
`calenda`
`jennife`
`wiki`.

*xistent* files?

dmin
signatures on
exist",
exist", etc.?

pproach:
ge that exists,
ything else."

existent page.
answer
oesn't exist.
to trust it.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: "NHTTPSEC". e.g.
`clegg.com/nonex` query returns
"`There are no pages between`
`clegg.com/nick and`
`clegg.com/start`" + signature.

Try `clegg.com/f`
After several queri
all `clegg.com` nar
`alan`, `alvis`, and
`calendar`, `home`,
`jennifer`, `mail`,
`wiki`.

s?

on

.?

ists,
„

ge.

t.

t.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: "NHTTPSEC". e.g.
`clegg.com/nonex` query returns
"`There are no pages between`
`clegg.com/nick and`
`clegg.com/start`" + signature.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `bria`
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `sta`
`wiki`.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: "NHTTPSEC". e.g.
`clegg.com/nonex` query returns
"`There are no pages between`
`clegg.com/nick` and
`clegg.com/start`" + signature.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `brian`,
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `start`,
`wiki`.

User asks for `nu.nl/economie`.
Receives unsigned answer,
a response forged by attacker,
saying the page doesn't exist.
Has no choice but to trust it.

Clearly a violation of availability.
Sometimes a violation of integrity.
This is not a good approach.

Alternative: "NHTTPSEC". e.g.
`clegg.com/nonex` query returns
"`There are no pages between`
`clegg.com/nick` `and`
`clegg.com/start`" + signature.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `brian`,
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `start`,
`wiki`.

The `clegg.com` administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

ks for nu.nl/economie.
s unsigned answer,
nse forged by attacker,
ne page doesn't exist.
choice but to trust it.

a violation of availability.
nes a violation of integrity.
not a good approach.

ive: "NHTTPSEC". e.g.
com/nonex query returns
are no pages between
com/nick and
com/start" + signature.

Try clegg.com/foo etc.
After several queries have
all clegg.com names:
alan, alvis, andrew, brian,
calendar, home, imogene,
jennifer, mail, nick, start,
wiki.

The clegg.com administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

Summar
all $n$ na
on an N
(with sig
that the
using $n$

nl/economie.
answer,
by attacker,
oesn't exist.
to trust it.

of availability.
tion of integrity.
l approach.

TTPSEC". e.g.
x query returns
ages between
and
t" + signature.

Try clegg.com/foo etc.
After several queries have
all clegg.com names:
alan, alvis, andrew, brian,
calendar, home, imogene,
jennifer, mail, nick, start,
wiki.

The clegg.com administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

Summary: Attacke
all $n$ names of pag
on an NHTTPSEC
(with signatures g
that there are no
using $n$ HTTPSEC

mie.

er,

t.

t.

bility.

tegrity.

.

e.g.

turns

veen

ature.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `brian`,
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `start`,
`wiki`.

The `clegg.com` administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteein
that there are no more)
using $n$ HTTPSEC queries.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `brian`,
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `start`,
`wiki`.

The `clegg.com` administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `brian`,
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `start`,
`wiki`.

The `clegg.com` administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

Try `clegg.com/foo` etc.
After several queries have
all `clegg.com` names:
`alan`, `alvis`, `andrew`, `brian`,
`calendar`, `home`, `imogene`,
`jennifer`, `mail`, `nick`, `start`,
`wiki`.

The `clegg.com` administrator
disabled HTTP directory indexing
— but then leaked the same data
by installing HTTPSEC
with the default NHTTPSEC.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

HTTPSEC purists disagree:
"It is part of the design
philosophy of the Web
that the data in it is public."
But this notion is so extreme
that it became an HTTPSEC
public-relations problem.

gg.com/foo etc.
veral queries have
g.com names:
lvis, andrew, brian,
ar, home, imogene,
er, mail, nick, start,

egg.com administrator
HTTP directory indexing
hen leaked the same data
ling HTTPSEC
default NHTTPSEC.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

HTTPSEC purists disagree:
"It is part of the design
philosophy of the Web
that the data in it is public."
But this notion is so extreme
that it became an HTTPSEC
public-relations problem.

New HT

1. "NHT
Use a "c
such as
Reveal *h*
instead o
"There
hashes

oo etc.
es have
mes:
rew, brian,
imogene,
nick, start,

dministrator
rectory indexing
d the same data
PSEC
HTTPSEC.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

HTTPSEC purists disagree:
"It is part of the design
philosophy of the Web
that the data in it is public."
But this notion is so extreme
that it became an HTTPSEC
public-relations problem.

New HTTPSEC a

1. "NHTTPSEC3
Use a "one-way ha
such as (iterated s
Reveal *hashes* of n
instead of revealin
"There are no na
hashes between

n,

rt,

or
dexing
e data

C.

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

HTTPSEC purists disagree:
"It is part of the design
philosophy of the Web
that the data in it is public."
But this notion is so extreme
that it became an HTTPSEC
public-relations problem.

New HTTPSEC approach:

1. "NHTTPSEC3" technolo
Use a "one-way hash functio
such as (iterated salted) SH
Reveal *hashes* of names
instead of revealing names.
"`There are no names with`
`hashes between ... and ..`

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

HTTPSEC purists disagree:
"It is part of the design
philosophy of the Web
that the data in it is public."
But this notion is so extreme
that it became an HTTPSEC
public-relations problem.

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with`
`hashes between ... and ...`"

Summary: Attacker learns
all $n$ names of pages
on an NHTTPSEC server
(with signatures guaranteeing
that there are no more)
using $n$ HTTPSEC queries.

This is not a good approach.

HTTPSEC purists disagree:
"It is part of the design
philosophy of the Web
that the data in it is public."
But this notion is so extreme
that it became an HTTPSEC
public-relations problem.

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with`
`hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow
enumeration of the site."

y: Attacker learns
mes of pages
HTTPSEC server
gnatures guaranteeing
re are no more)
HTTPSEC queries.

not a good approach.

EC purists disagree:
rt of the design
hy of the Web
data in it is public."
notion is so extreme
ecame an HTTPSEC
elations problem.

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow
enumeration of the site."

Reality:
by abusi
compute
for many
quickly c
(and kno

er learns
ges
C server
uaranteeing
more)
C queries.

l approach.

disagree:

lesign
Web
is public."
so extreme
HTTPSEC
oblem.

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow
enumeration of the site."

Reality: Attacker
by abusing NHTT
computes the sam
for many different
quickly discovers a
(and knows # mis

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow enumeration of the site."

Reality: Attacker grabs the
by abusing NHTTPSEC3;
computes the same hash fu
for many different name gue
quickly discovers almost all
(and knows # missing name

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow enumeration of the site."

Reality: Attacker grabs the hashes
by abusing NHTTPSEC3;
computes the same hash function
for many different name guesses;
quickly discovers almost all names
(and knows # missing names).

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow enumeration of the site."

Reality: Attacker grabs the hashes
by abusing NHTTPSEC3;
computes the same hash function
for many different name guesses;
quickly discovers almost all names
(and knows # missing names).

HTTPSEC purists: "You could have sent all the same guesses as queries to the server."

New HTTPSEC approach:

1. "NHTTPSEC3" technology:
Use a "one-way hash function"
such as (iterated salted) SHA-1.
Reveal *hashes* of names
instead of revealing names.
"`There are no names with hashes between ... and ...`"

2. Marketing:
Pretend that NHTTPSEC3 is
less damaging than NSEC.

"NHTTPSEC3 does not allow enumeration of the site."

Reality: Attacker grabs the hashes
by abusing NHTTPSEC3;
computes the same hash function
for many different name guesses;
quickly discovers almost all names
(and knows # missing names).

HTTPSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is
under 5000 noisy guesses/sec.
NHTTPSEC3 allows typical
attackers 10000000 to
10000000000 silent guesses/sec.

TTPSEC approach:

TTPSEC3" technology:
one-way hash function"
(iterated salted) SHA-1.
*hashes* of names
of revealing names.
`are no names with`
`between ... and ...`"

eting:
that NHTTPSEC3 is
naging than NSEC.

PSEC3 does not allow
tion of the site."

Reality: Attacker grabs the hashes by abusing NHTTPSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

HTTPSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 5000 noisy guesses/sec. NHTTPSEC3 allows typical attackers 10000000 to 10000000000 silent guesses/sec.

Another

Each HT
is anoth
Often yo
of keys f
Could be

HTTPSE
by accep
and send
through

Much lo

pproach:

" technology:
ash function"
salted) SHA-1.
names
g names.
ames with
... and ..."

TTPSEC3 is
n NSEC.

es not allow
e site."

Reality: Attacker grabs the hashes by abusing NHTTPSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

HTTPSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 5000 noisy guesses/sec. NHTTPSEC3 allows typical attackers 10000000 to 10000000000 silent guesses/sec.

Another HTTPSE

Each HTTPSEC k
is another file to r
Often your browse
of keys from sever
Could be a serious

HTTPSEC speeds
by accepting reque
and sending respo
through UDP pack

Much lower overhe

gy:
on"
A-1.

."

is

ow

Reality: Attacker grabs the hashes by abusing NHTTPSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

HTTPSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 5000 noisy guesses/sec. NHTTPSEC3 allows typical attackers 10000000 to 10000000000 silent guesses/sec.

## Another HTTPSEC optimiza

Each HTTPSEC key/signatu is another file to retrieve. Often your browser needs a of keys from several servers. Could be a serious slowdown

HTTPSEC speeds this up by accepting requests and sending responses through UDP packets.

Much lower overhead than T

Reality: Attacker grabs the hashes by abusing NHTTPSEC3; computes the same hash function for many different name guesses; quickly discovers almost all names (and knows # missing names).

HTTPSEC purists: "You could have sent all the same guesses as queries to the server."

4Mbps flood of queries is under 5000 noisy guesses/sec. NHTTPSEC3 allows typical attackers 10000000 to 10000000000 silent guesses/sec.

## Another HTTPSEC optimization

Each HTTPSEC key/signature is another file to retrieve. Often your browser needs a chain of keys from several servers. Could be a serious slowdown.

HTTPSEC speeds this up by accepting requests and sending responses through UDP packets.

Much lower overhead than TCP.

Attacker grabs the hashes
ng NHTTPSEC3;
es the same hash function
y different name guesses;
discovers almost all names
ows # missing names).

EC purists: "You could
t all the same guesses
es to the server."

lood of queries is
000 noisy guesses/sec.
SEC3 allows typical
s 10000000 to
00000 silent guesses/sec.

## Another HTTPSEC optimization

Each HTTPSEC key/signature
is another file to retrieve.
Often your browser needs a chain
of keys from several servers.
Could be a serious slowdown.

HTTPSEC speeds this up
by accepting requests
and sending responses
through UDP packets.

Much lower overhead than TCP.

The bad
HTTPS
much, m
than HT

Attacker
UDP req
victim's
to many

The HT
blast the
much lar
taking v

grabs the hashes
PSEC3;
e hash function
name guesses;
almost all names
ssing names).

: "You could
ame guesses
server."

ueries is
guesses/sec.
ws typical
0 to
t guesses/sec.

## Another HTTPSEC optimization

Each HTTPSEC key/signature
is another file to retrieve.
Often your browser needs a chain
of keys from several servers.
Could be a serious slowdown.

HTTPSEC speeds this up
by accepting requests
and sending responses
through UDP packets.

Much lower overhead than TCP.

The bad news:
HTTPSEC respon
much, much, muc
than HTTPSEC re

Attacker forges ma
UDP request pack
victim's IP address
to many HTTPSE

The HTTPSEC se
blast the victim w
much larger volum
taking victim off t

hashes

nction
esses;
names
es).

uld
ses

ec.

/sec.

## Another HTTPSEC optimization

Each HTTPSEC key/signature
is another file to retrieve.
Often your browser needs a chain
of keys from several servers.
Could be a serious slowdown.

HTTPSEC speeds this up
by accepting requests
and sending responses
through UDP packets.

Much lower overhead than TCP.

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Interne

## Another HTTPSEC optimization

Each HTTPSEC key/signature
is another file to retrieve.
Often your browser needs a chain
of keys from several servers.
Could be a serious slowdown.

HTTPSEC speeds this up
by accepting requests
and sending responses
through UDP packets.

Much lower overhead than TCP.

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

## HTTPSEC optimization

TTPSEC key/signature
er file to retrieve.
our browser needs a chain
from several servers.
e a serious slowdown.

EC speeds this up
pting requests
ding responses
UDP packets.

wer overhead than TCP.

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

The RFC
provides
against

C optimization

key/signature
etrieve.
er needs a chain
al servers.
s slowdown.

this up
ests
nses
kets.

ead than TCP.

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

The RFC says "HT
provides no protec
against denial of s

ation

ure

chain

n.

TCP.

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

The RFC says "HTTPSEC
provides no protection
against denial of service atta

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

The bad news:
HTTPSEC responses are
much, much, much larger
than HTTPSEC requests.

Attacker forges many
UDP request packets from
victim's IP address
to many HTTPSEC servers.

The HTTPSEC servers
blast the victim with
much larger volume of data,
taking victim off the Internet.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

news:
EC responses are
nuch, much larger
TTPSEC requests.

r forges many
quest packets from
IP address
HTTPSEC servers.

TTPSEC servers
e victim with
rger volume of data,
ictim off the Internet.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

The wor

ses are
h larger
equests.

any
kets from
s
C servers.

ervers
ith
ne of data,
he Internet.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

The worst part of

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

The worst part of HTTPSE

t.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

## The worst part of HTTPSEC

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

HTTPSEC signs only
*routing* information:
specifically, 30x HTTP redirects.

The RFC says "HTTPSEC
provides no protection
against denial of service attacks."

The RFC doesn't say
"HTTPSEC is a pool of
remote-controlled attack drones,
the worst DDoS amplifier
on the Internet."

Exericse: investigate
other types of DoS attacks.
e.g. HTTPSEC advertising says
zero server-CPU-time cost.
How much server CPU time
can attackers actually consume?

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

HTTPSEC signs only
*routing* information:
specifically, 30x HTTP redirects.

The HTTPSEC excuse for this:
signing redirects
is simpler than
signing the final web page.

C says "HTTPSEC
no protection
denial of service attacks."

C doesn't say
SEC is a pool of
controlled attack drones,
st DDoS amplifier
nternet."

: investigate
pes of DoS attacks.
TPSEC advertising says
ver-CPU-time cost.
ch server CPU time
ckers actually consume?

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

HTTPSEC signs only
*routing* information:
specifically, 30x HTTP redirects.

The HTTPSEC excuse for this:
signing redirects
is simpler than
signing the final web page.

```
$ telne
Trying
Connect
Escape
GET / H
Host: g

HTTP/1.
Locatio
...
```

---

HTTPS
on the "
www.goo

TTPSEC
tion
ervice attacks."

say
ool of
attack drones,
mplifier

ate
S attacks.
dvertising says
ime cost.
CPU time
ually consume?

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

HTTPSEC signs only
*routing* information:
specifically, 30x HTTP redirects.

The HTTPSEC excuse for this:
signing redirects
is simpler than
signing the final web page.

```
$ telnet google.
Trying 173.194.6
Connected to goo
Escape character
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Mov
Location: http:/
...
```

---

HTTPSEC allows
on the "google.c
www.google.com"

acks."

ones,

says

me?

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

HTTPSEC signs only
*routing* information:
specifically, 30x HTTP redirects.

The HTTPSEC excuse for this:
signing redirects
is simpler than
signing the final web page.

```
$ telnet google.com 80
Trying 173.194.66.102...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Perman
Location: http://www.goog
...
```

---

HTTPSEC allows a signatur
on the "`google.com` →
`www.google.com`" redirect.

## The worst part of HTTPSEC

The data signed by HTTPSEC
doesn't actually include
*the web pages that*
*the browser shows to the user*.

HTTPSEC signs only
*routing* information:
specifically, 30x HTTP redirects.

The HTTPSEC excuse for this:
signing redirects
is simpler than
signing the final web page.

```
$ telnet google.com 80
Trying 173.194.66.102...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
...
```

---

HTTPSEC allows a signature
on the "`google.com` →
`www.google.com`" redirect.

st part of HTTPSEC

a signed by HTTPSEC
actually include
*pages that*
*vser shows to the user*.

EC signs only
information:
lly, 30x HTTP redirects.

TPSEC excuse for this:
redirects
er than
the final web page.

```
$ telnet google.com 80
Trying 173.194.66.102...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
...
```

---

HTTPSEC allows a signature
on the "`google.com` →
`www.google.com`" redirect.

```
$ telne
Trying
Connect
Escape
GET / H
Host: w

HTTP/1.
Locatio
...
```

---

HTTPS
on the
`www.goo`

HTTPSEC

y HTTPSEC
clude
*t*
*s to the user*.

nly
on:
TTP redirects.

xcuse for this:

eb page.

```
$ telnet google.com 80
Trying 173.194.66.102...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
...
```

---

HTTPSEC allows a signature on the "`google.com` → `www.google.com`" redirect.

```
$ telnet www.goo
Trying 173.194.6
Connected to www
Escape character
GET / HTTP/1.1
Host: www.google

HTTP/1.1 302 Fou
Location: http:/
...
```

---

HTTPSEC allows
on the "`www.goog`
`www.google.nl`"

C

EC

*er.*

ects.

his:

```
$ telnet google.com 80
Trying 173.194.66.102...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
...
```

HTTPSEC allows a signature on the "google.com → www.google.com" redirect.

```
$ telnet www.google.com 8
Trying 173.194.67.104...
Connected to www.google.c
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 302 Found
Location: http://www.goog
...
```

HTTPSEC allows a signatur on the "www.google.com → www.google.nl" redirect.

```
$ telnet google.com 80
Trying 173.194.66.102...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: google.com

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
...
```

---

HTTPSEC allows a signature on the "`google.com` → `www.google.com`" redirect.

```
$ telnet www.google.com 80
Trying 173.194.67.104...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 302 Found
Location: http://www.google.nl/
...
```

---

HTTPSEC allows a signature on the "`www.google.com` → `www.google.nl`" redirect.

```
t google.com 80
173.194.66.102...
ed to google.com.
character is '^]'.
TTP/1.1
oogle.com

1 301 Moved Permanently
n: http://www.google.com/
```

EC allows a signature
"google.com →
ogle.com" redirect.

```
$ telnet www.google.com 80
Trying 173.194.67.104...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 302 Found
Location: http://www.google.nl/
...
```

HTTPSEC allows a signature
on the "www.google.com →
www.google.nl" redirect.

```
$ telne
Trying
Connecte
Escape
GET / H
Host: w

HTTP/1.
...
```

The resp
Google v

HTTPS
HTTPS

```
com 80
6.102...
gle.com.
 is '^]'.
```

```
ed Permanently
/www.google.com/
```

a signature
com →
" redirect.

```
$ telnet www.google.com 80
Trying 173.194.67.104...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 302 Found
Location: http://www.google.nl/
...
```

HTTPSEC allows a signature on the "`www.google.com` → `www.google.nl`" redirect.

```
$ telnet www.goo
Trying 173.194.6
Connected to www
Escape character
GET / HTTP/1.1
Host: www.google

HTTP/1.1 200 OK
...
```

The response cont
Google web page.

HTTPSEC *does n*
HTTPSEC signs o

```
ently
gle.com/
```

---

re

```
$ telnet www.google.com 80
Trying 173.194.67.104...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 302 Found
Location: http://www.google.nl/
...
```

---

HTTPSEC allows a signature on the "www.google.com → www.google.nl" redirect.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.n
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

---

The response contains the a Google web page.

HTTPSEC *does not sign th* HTTPSEC signs only redire

```
$ telnet www.google.com 80
Trying 173.194.67.104...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com

HTTP/1.1 302 Found
Location: http://www.google.nl/
...
```

---

HTTPSEC allows a signature on the "`www.google.com` → `www.google.nl`" redirect.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

```
t www.google.com 80
173.194.67.104...
ed to www.google.com.
character is '^]'.
TTP/1.1
ww.google.com

1 302 Found
n: http://www.google.nl/
```

EC allows a signature
"www.google.com →
ogle.nl" redirect.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You ma
and you'

```
gle.com 80
7.104...
.google.com.
 is '^]'.

.com

nd
/www.google.nl/
```

---

a signature
`gle.com` →
redirect.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You may say this
and you're not the

80

com.

gle.nl/

re

→

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You may say this is stupid,
and you're not the only one.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK

...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You may say this is stupid, and you're not the only one."

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK

...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You may say this is stupid, and you're not the only one."

If final web page isn't signed, what is the security benefit of signing the redirects?
Attacker simply forges the page.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

```
$ telnet www.google.nl 80
Trying 173.194.66.94...
Connected to www.google.nl.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.nl

HTTP/1.1 200 OK
...
```

---

The response contains the actual Google web page.

HTTPSEC *does not sign this*.
HTTPSEC signs only redirects.

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

Redirects can benefit from
*availability* and *confidentiality*,
but HTTPSEC doesn't provide
availability and confidentiality.

```
t www.google.nl 80
173.194.66.94...
ed to www.google.nl.
character is '^]'.
TTP/1.1
ww.google.nl


1 200 OK
```

---

ponse contains the actual
web page.

EC *does not sign this*.
EC signs only redirects.

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

Redirects can benefit from
*availability* and *confidentiality*,
but HTTPSEC doesn't provide
availability and confidentiality.

HTTPSE

After yea
>100 pe
tens of r
regulatio
from gov
direct pa
*please* in

```
gle.nl 80
6.94...
.google.nl.
 is '^]'.


.nl
```

tains the actual

*ot sign this*.

nly redirects.

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

Redirects can benefit from
*availability* and *confidentiality*,
but HTTPSEC doesn't provide
availability and confidentiality.

HTTPSEC vs. HT

After years of deve
>100 people, gran
tens of millions of
regulations requiri
from government a
direct payments to
*please* install HTT

)

ll.

---

ctual

*is*.

cts.

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

Redirects can benefit from
*availability* and *confidentiality*,
but HTTPSEC doesn't provide
availability and confidentiality.

## HTTPSEC vs. HTTPS

After years of development b
>100 people, grants totallin
tens of millions of EUR, U.S
regulations requiring HTTPS
from government agencies, a
direct payments to admins t
*please* install HTTPSEC:

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

Redirects can benefit from
*availability* and *confidentiality*,
but HTTPSEC doesn't provide
availability and confidentiality.

## HTTPSEC vs. HTTPS

After years of development by
>100 people, grants totalling
tens of millions of EUR, U.S.
regulations requiring HTTPSEC
from government agencies, and
direct payments to admins to
*please* install HTTPSEC:

"You may say this is stupid,
and you're not the only one."

If final web page isn't signed,
what is the security benefit of
signing the redirects?
Attacker simply forges the page.

If final web page *is* signed,
what is the security benefit of
signing the redirects?
Attacker can't forge the page.

Redirects can benefit from
*availability* and *confidentiality*,
but HTTPSEC doesn't provide
availability and confidentiality.

## HTTPSEC vs. HTTPS

After years of development by
>100 people, grants totalling
tens of millions of EUR, U.S.
regulations requiring HTTPSEC
from government agencies, and
direct payments to admins to
*please* install HTTPSEC:

HTTPSEC is running on a few
thousand Internet servers.

Network World, 2013.01.29:
"HTTPSEC adoption stalls
outside of federal government"

ay say this is stupid,
're not the only one."

web page isn't signed,
the security benefit of
the redirects?
r simply forges the page.

web page *is* signed,
the security benefit of
the redirects?
r can't forge the page.

s can benefit from
*ity* and *confidentiality*,
TPSEC doesn't provide
ity and confidentiality.

## HTTPSEC vs. HTTPS

After years of development by >100 people, grants totalling tens of millions of EUR, U.S. regulations requiring HTTPSEC from government agencies, and direct payments to admins to *please* install HTTPSEC:

HTTPSEC is running on a few thousand Internet servers.

Network World, 2013.01.29:
"HTTPSEC adoption stalls outside of federal government"

There's

HTTPS
*and* con
*for the c*

HTTPS
web too

HTTPS
doesn't
with nor
tries to a
isn't a h

is stupid,
e only one."

sn't signed,
ty benefit of
cts?
rges the page.

*s* signed,
ty benefit of
cts?
ge the page.

efit from
*onfidentiality*,
esn't provide
nfidentiality.

## HTTPSEC vs. HTTPS

After years of development by >100 people, grants totalling tens of millions of EUR, U.S. regulations requiring HTTPSEC from government agencies, and direct payments to admins to *please* install HTTPSEC:

HTTPSEC is running on a few thousand Internet servers.

Network World, 2013.01.29: "HTTPSEC adoption stalls outside of federal government"

There's competitio

HTTPS aims for i
*and* confidentiality
*for the complete w*

HTTPS works wit
web tools and dyn

HTTPS doesn't al
doesn't have any p
with nonexistent f
tries to avoid leaki
isn't a huge DDoS

## HTTPSEC vs. HTTPS

After years of development by >100 people, grants totalling tens of millions of EUR, U.S. regulations requiring HTTPSEC from government agencies, and direct payments to admins to *please* install HTTPSEC:

HTTPSEC is running on a few thousand Internet servers.

Network World, 2013.01.29: "HTTPSEC adoption stalls outside of federal government"

There's competition: HTTP

HTTPS aims for integrity *and* confidentiality *for the complete web pages*.

HTTPS works with existing web tools and dynamic data

HTTPS doesn't allow replay doesn't have any problems with nonexistent files; tries to avoid leaking data; isn't a huge DDoS amplifier

## HTTPSEC vs. HTTPS

After years of development by
>100 people, grants totalling
tens of millions of EUR, U.S.
regulations requiring HTTPSEC
from government agencies, and
direct payments to admins to
*please* install HTTPSEC:

HTTPSEC is running on a few
thousand Internet servers.

Network World, 2013.01.29:
"HTTPSEC adoption stalls
outside of federal government"

## There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

## EC vs. HTTPS

ars of development by
eople, grants totalling
millions of EUR, U.S.
ons requiring HTTPSEC
vernment agencies, and
ayments to admins to
nstall HTTPSEC:

EC is running on a few
d Internet servers.

World, 2013.01.29:
SEC adoption stalls
of federal government"

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What th
say abou

"HTTPS
to be co

TTPS

elopment by
nts totalling
EUR, U.S.
ng HTTPSEC
agencies, and
o admins to
TPSEC:

ning on a few
servers.

013.01.29:
tion stalls
government"

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPS
say about HTTPS

"HTTPS requires
to be constantly o

by
g
.
SEC
and
o

ew

nt”

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC propon
say about HTTPS:

“HTTPS requires keys
to be constantly online.”

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC proponents
say about HTTPS:

"HTTPS requires keys
to be constantly online."

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC proponents
say about HTTPS:

"HTTPS requires keys
to be constantly online."

Yes, it does; so what?

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC proponents
say about HTTPS:

"HTTPS requires keys
to be constantly online."

Yes, it does; so what?

"HTTPS requires servers
to use per-query crypto."

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC proponents
say about HTTPS:

"HTTPS requires keys
to be constantly online."

Yes, it does; so what?

"HTTPS requires servers
to use per-query crypto."

Yes, it does; so what?

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC proponents
say about HTTPS:

"HTTPS requires keys
to be constantly online."

Yes, it does; so what?

"HTTPS requires servers
to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the
channel, not the data. It doesn't
provide end-to-end security."

There's competition: HTTPS!

HTTPS aims for integrity
*and* confidentiality
*for the complete web pages*.

HTTPS works with existing
web tools and dynamic data.

HTTPS doesn't allow replays;
doesn't have any problems
with nonexistent files;
tries to avoid leaking data;
isn't a huge DDoS amplifier.

What the HTTPSEC proponents
say about HTTPS:

"HTTPS requires keys
to be constantly online."

Yes, it does; so what?

"HTTPS requires servers
to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the
channel, not the data. It doesn't
provide end-to-end security."

Huh? What does this mean?

competition: HTTPS!

aims for integrity
fidentiality
*complete web pages*.

works with existing
ls and dynamic data.

doesn't allow replays;
have any problems
nexistent files;
avoid leaking data;
uge DDoS amplifier.

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the s
signed d
laptop t
which gi
which ve
then the
HTTPS

on: HTTPS!

ntegrity
/
*web pages*.

h existing
amic data.

llow replays;
problems
iles;
ing data;
S amplifier.

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the site owner
signed data from h
laptop to an untru
which gives it to y
which verifies the
then the server ca
HTTPS lets the se

PS!

.

ys;

.

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the site owner copies PG
signed data from his trusted
laptop to an untrusted serve
which gives it to your brows
which verifies the signed dat
then the server can't change
HTTPS lets the server chan

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

"HTTPS destroys the caching layer. This Matters."

What the HTTPSEC proponents say about HTTPS:

"HTTPS requires keys to be constantly online."

Yes, it does; so what?

"HTTPS requires servers to use per-query crypto."

Yes, it does; so what?

"HTTPS protects only the channel, not the data. It doesn't provide end-to-end security."

Huh? What does this mean?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

"HTTPS destroys the caching layer. This Matters."

Yeah, sure it does. Film at 11: Internet Destroyed By HTTPS.

e HTTPSEC proponents
ut HTTPS:

S requires keys
nstantly online."

oes; so what?

S requires servers
er-query crypto."

oes; so what?

S protects only the
not the data. It doesn't
end-to-end security."

/hat does this mean?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

"HTTPS destroys the caching layer. This Matters."

Yeah, sure it does. Film at 11: Internet Destroyed By HTTPS.

The DN

I've beer
data sen
includin

`google.`

`www.goo`
`www.goo`

But ther
many m
Domain

EC proponents

:

keys
nline."

nat?

servers
rypto."

nat?

only the
ata. It doesn't
d security."

this mean?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

"HTTPS destroys the caching layer. This Matters."

Yeah, sure it does. Film at 11: Internet Destroyed By HTTPS.

The DNS security

I've been describin
data sent to your
including two HTT

`google.com` → `ww`

`www.google.com`
`www.google.nl`

`www.google.nl` w

But there are actu
many more redirec
Domain Name Sys

nents

esn't
,

?

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

"HTTPS destroys the caching layer. This Matters."

Yeah, sure it does. Film at 11: Internet Destroyed By HTTPS.

## The DNS security mess

I've been describing data sent to your browser, including two HTTP redirec

`google.com` → `www.googl`

`www.google.com` → `www.google.nl`

`www.google.nl` web page

But there are actually many more redirection steps Domain Name System looku

"If the site owner copies PGP-signed data from his trusted laptop to an untrusted server, which gives it to your browser, which verifies the signed data, then the server can't change it. HTTPS lets the server change it."

Yes, of course, but why is the site owner putting his data on an untrusted server?

"HTTPS destroys the caching layer. This Matters."

Yeah, sure it does. Film at 11: Internet Destroyed By HTTPS.

## The DNS security mess

I've been describing data sent to your browser, including two HTTP redirects:

`google.com` → `www.google.com`

`www.google.com` → `www.google.nl`

`www.google.nl` web page

But there are actually many more redirection steps: Domain Name System lookups.

ite owner copies PGP-
ata from his trusted
o an untrusted server,
ves it to your browser,
erifies the signed data,
e server can't change it.
lets the server change it."

course, but why is the
er putting his data on
usted server?

S destroys the caching
his Matters."

re it does. Film at 11:
Destroyed By HTTPS.

## The DNS security mess

I've been describing
data sent to your browser,
including two HTTP redirects:

`google.com` → `www.google.com`

`www.google.com` →
`www.google.nl`

`www.google.nl` web page

But there are actually
many more redirection steps:
Domain Name System lookups.

```
com NS
google.
google.
google.
www.goo
173.194.
www.goo
www.goo
nl NS 1
google.
www.goo
www.goo
```

copies PGP-
his trusted
usted server,
your browser,
signed data,
n't change it.
erver change it."

t why is the
his data on
er?

the caching
rs."

. Film at 11:
l By HTTPS.

## The DNS security mess

I've been describing
data sent to your browser,
including two HTTP redirects:

`google.com` → `www.google.com`

`www.google.com` →
`www.google.nl`

`www.google.nl` web page

But there are actually
many more redirection steps:
Domain Name System lookups.

`com` NS 192.5.6.30

`google.com` NS 2

`google.com` A 74

`google.com` → `ww`

`www.google.com`
173.194.66.99

`www.google.com`

`www.google.nl`

`nl` NS 192.5.4.1

`google.nl` NS 21

`www.google.nl` A

`www.google.nl` w

P-

er,
er,
ta,
e it.
ge it."

ne
on

ng

11:
PS.

## The DNS security mess

I've been describing
data sent to your browser,
including two HTTP redirects:

`google.com` → `www.google.com`

`www.google.com` →
`www.google.nl`

`www.google.nl` web page

But there are actually
many more redirection steps:
Domain Name System lookups.

`com` NS 192.5.6.30

`google.com` NS 216.239.34

`google.com` A 74.125.136.1

`google.com` → `www.googl`

`www.google.com` A
173.194.66.99

`www.google.com` →
`www.google.nl`

`nl` NS 192.5.4.1

`google.nl` NS 216.239.34.1

`www.google.nl` A 74.125.1

`www.google.nl` web page

## The DNS security mess

I've been describing
data sent to your browser,
including two HTTP redirects:

`google.com` → `www.google.com`

`www.google.com` →
`www.google.nl`

`www.google.nl` web page

But there are actually
many more redirection steps:
Domain Name System lookups.

`com` NS 192.5.6.30

`google.com` NS 216.239.34.10

`google.com` A 74.125.136.100

`google.com` → `www.google.com`

`www.google.com` A
173.194.66.99

`www.google.com` →
`www.google.nl`

`nl` NS 192.5.4.1

`google.nl` NS 216.239.34.10

`www.google.nl` A 74.125.132.94

`www.google.nl` web page

S security mess

n describing
t to your browser,
g two HTTP redirects:

.com → www.google.com

ogle.com →
ogle.nl

ogle.nl web page

re are actually
ore redirection steps:
Name System lookups.

com NS 192.5.6.30

google.com NS 216.239.34.10

google.com A 74.125.136.100

google.com → www.google.com

www.google.com A
173.194.66.99

www.google.com →
www.google.nl

www.google.nl

nl NS 192.5.4.1

google.nl NS 216.239.34.10

www.google.nl A 74.125.132.94

www.google.nl web page

DNSSE
in very
HTTPS

All the
are shar
includin
almost a
cryptogr

## mess

ng
browser,
TP redirects:

ww.google.com

→

veb page

ually
ction steps:
stem lookups.

com NS 192.5.6.30

google.com NS 216.239.34.10

google.com A 74.125.136.100

google.com → www.google.com

www.google.com A
173.194.66.99

www.google.com →
www.google.nl

nl NS 192.5.4.1

google.nl NS 216.239.34.10

www.google.nl A 74.125.132.94

www.google.nl web page

DNSSEC signs DN
in very much the s
HTTPSEC signs H

All the problems o
are shared by DNS
including lack of d
almost all DNS pa
cryptographically u

ts:

e.com

:

ups.

com NS 192.5.6.30

google.com NS 216.239.34.10

google.com A 74.125.136.100

google.com → www.google.com

www.google.com A
173.194.66.99

www.google.com →
www.google.nl

nl NS 192.5.4.1

google.nl NS 216.239.34.10

www.google.nl A 74.125.132.94

www.google.nl web page

DNSSEC signs DNS redirect
in very much the same way
HTTPSEC signs HTTP redi

All the problems of HTTPSE
are shared by DNSSEC,
including lack of deployment
almost all DNS packets are
cryptographically unprotecte

`com` NS 192.5.6.30

`google.com` NS 216.239.34.10

`google.com` A 74.125.136.100

`google.com` → `www.google.com`

`www.google.com` A
173.194.66.99

`www.google.com` →
`www.google.nl`

`nl` NS 192.5.4.1

`google.nl` NS 216.239.34.10

`www.google.nl` A 74.125.132.94

`www.google.nl` web page

DNSSEC signs DNS redirects
in very much the same way that
HTTPSEC signs HTTP redirects.

All the problems of HTTPSEC
are shared by DNSSEC,
including lack of deployment:
almost all DNS packets are
cryptographically unprotected.

`com` NS 192.5.6.30

`google.com` NS 216.239.34.10

`google.com` A 74.125.136.100

`google.com` → `www.google.com`

`www.google.com` A
173.194.66.99

`www.google.com` →
`www.google.nl`

`nl` NS 192.5.4.1

`google.nl` NS 216.239.34.10

`www.google.nl` A 74.125.132.94

`www.google.nl` web page

DNSSEC signs DNS redirects
in very much the same way that
HTTPSEC signs HTTP redirects.

All the problems of HTTPSEC
are shared by DNSSEC,
including lack of deployment:
almost all DNS packets are
cryptographically unprotected.

Actually, HTTPSEC is an
imaginary imitation of DNSSEC,
not a real proposal.
But DNSSEC is a real proposal,
and has all of these problems.