Boring crypto

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

Ancient Chinese curse: "May you live in interesting times, so that you have many papers to write."

Related mailing list:
boring-crypto+subscribe@googlegroups.com

Some recent TLS failures

Diginotar CA compromise.
BEAST CBC attack.
Trustwave HTTPS interception.
CRIME compression attack.
Lucky 13 padding/timing attack.
RC4 keystream bias.
TLS truncation.
gotofail signature-verification bug.
Triple Handshake.
Heartbleed buffer overread.
POODLE padding-oracle attack.
Winshock buffer overflow.
FREAK factorization attack.
Logjam discrete-log attack.

rypto

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

Chinese curse: “May you
nteresting times, so that
e many papers to write.”

---

mailing list:
-crypto+subscribe
egroups.com

Some recent TLS failures

Diginotar CA compromise.
BEAST CBC attack.
Trustwave HTTPS interception.
CRIME compression attack.
Lucky 13 padding/timing attack.
RC4 keystream bias.
TLS truncation.
gotofail signature-verification bug.
Triple Handshake.
Heartbleed buffer overread.
POODLE padding-oracle attack.
Winshock buffer overflow.
FREAK factorization attack.
Logjam discrete-log attack.

TLS is n

New att
Disputes
Improve
Propose
Even be
Emerger
Different
New pro

n

is at Chicago &

siteit Eindhoven

---

urse: "May you

times, so that

pers to write."

---

st:

subscribe

com

Some recent TLS failures

Diginotar CA compromise.

BEAST CBC attack.

Trustwave HTTPS interception.

CRIME compression attack.

Lucky 13 padding/timing attack.

RC4 keystream bias.

TLS truncation.

gotofail signature-verification bug.

Triple Handshake.

Heartbleed buffer overread.

POODLE padding-oracle attack.

Winshock buffer overflow.

FREAK factorization attack.

Logjam discrete-log attack.

TLS is not boring

New attacks!

Disputes about se

Improved attacks!

Proposed fixes!

Even better attack

Emergency upgrad

Different attacks!

New protocol vers

ago &
hoven

y you
that
rite.”

e

Some recent TLS failures

Diginotar CA compromise.
BEAST CBC attack.
Trustwave HTTPS interception.
CRIME compression attack.
Lucky 13 padding/timing attack.
RC4 keystream bias.
TLS truncation.
gotofail signature-verification bug.
Triple Handshake.
Heartbleed buffer overread.
POODLE padding-oracle attack.
Winshock buffer overflow.
FREAK factorization attack.
Logjam discrete-log attack.

TLS is not boring crypto.

New attacks!
Disputes about security!
Improved attacks!
Proposed fixes!
Even better attacks!
Emergency upgrades!
Different attacks!
New protocol versions!

Some recent TLS failures

Diginotar CA compromise.

BEAST CBC attack.

Trustwave HTTPS interception.

CRIME compression attack.

Lucky 13 padding/timing attack.

RC4 keystream bias.

TLS truncation.

gotofail signature-verification bug.

Triple Handshake.

Heartbleed buffer overread.

POODLE padding-oracle attack.

Winshock buffer overflow.

FREAK factorization attack.

Logjam discrete-log attack.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Some recent TLS failures

Diginotar CA compromise.

BEAST CBC attack.

Trustwave HTTPS interception.

CRIME compression attack.

Lucky 13 padding/timing attack.

RC4 keystream bias.

TLS truncation.

gotofail signature-verification bug.

Triple Handshake.

Heartbleed buffer overread.

POODLE padding-oracle attack.

Winshock buffer overflow.

FREAK factorization attack.

Logjam discrete-log attack.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Continual excitement;
tons of research papers;
more jobs for cryptographers.

Some recent TLS failures

Diginotar CA compromise.

BEAST CBC attack.

Trustwave HTTPS interception.

CRIME compression attack.

Lucky 13 padding/timing attack.

RC4 keystream bias.

TLS truncation.

gotofail signature-verification bug.

Triple Handshake.

Heartbleed buffer overread.

POODLE padding-oracle attack.

Winshock buffer overflow.

FREAK factorization attack.

Logjam discrete-log attack.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Continual excitement;
tons of research papers;
more jobs for cryptographers.

Let's look at an example.

cent TLS failures

r CA compromise.

CBC attack.

ve HTTPS interception.

compression attack.

3 padding/timing attack.

stream bias.

ncation.

signature-verification bug.

andshake.

ed buffer overread.

E padding-oracle attack.

ck buffer overflow.

factorization attack.

discrete-log attack.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Continual excitement;
tons of research papers;
more jobs for cryptographers.

Let's look at an example.

The RC4

1987: R

Does no

failures

promise.

ck.

S interception.

on attack.

/timing attack.

as.

verification bug.

overread.

-oracle attack.

overflow.

ion attack.

g attack.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Continual excitement;

tons of research papers;

more jobs for cryptographers.

Let's look at an example.

The RC4 stream c

1987: Ron Rivest

Does not publish i

tion.

tack.

n bug.

ack.

.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Continual excitement;
tons of research papers;
more jobs for cryptographers.

Let's look at an example.

The RC4 stream cipher

1987: Ron Rivest designs R
Does not publish it.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

Continual excitement;
tons of research papers;
more jobs for cryptographers.

Let's look at an example.

The RC4 stream cipher

1987: Ron Rivest designs RC4.

Does not publish it.

TLS is not boring crypto.

New attacks!
Disputes about security!
Improved attacks!
Proposed fixes!
Even better attacks!
Emergency upgrades!
Different attacks!
New protocol versions!

Continual excitement;
tons of research papers;
more jobs for cryptographers.

Let's look at an example.

The RC4 stream cipher

1987: Ron Rivest designs RC4.
Does not publish it.

1992: NSA makes a deal with Software Publishers Association.

"NSA allows encryption . . .
The U.S. Department of State will grant export permission to any program that uses the RC2 or RC4 data-encryption algorithm with a key size of less than 40 bits."

not boring crypto.

acks!

s about security!

d attacks!

d fixes!

tter attacks!

ncy upgrades!

t attacks!

tocol versions!

al excitement;

research papers;

os for cryptographers.

ok at an example.

The RC4 stream cipher

1987: Ron Rivest designs RC4. Does not publish it.

1992: NSA makes a deal with Software Publishers Association.

"NSA allows encryption . . . The U.S. Department of State will grant export permission to any program that uses the RC2 or RC4 data-encryption algorithm with a key size of less than 40 bits."

1994: S

posts R

New Yor

dissemin

the long

system .

the de f

for many

program

Window

operatin

Notes. .

was part

be kept

presiden

crypto.

curity!

ks!
les!

ions!

ent;
apers;
tographers.

xample.

The RC4 stream cipher

1987: Ron Rivest designs RC4.
Does not publish it.

1992: NSA makes a deal with
Software Publishers Association.

"NSA allows encryption . . .
The U.S. Department of State
will grant export permission
to any program that uses the
RC2 or RC4 data-encryption
algorithm with a key size
of less than 40 bits."

1994: Someone an
posts RC4 source

New York Times:
dissemination coul
the long-term effe
system . . . [RC4]
the de facto codin
for many popular s
programs including
Windows, Apple's
operating system a
Notes. . . . 'I have
was part of this de
be kept confidenti
president of RSA,

s.

The RC4 stream cipher

1987: Ron Rivest designs RC4. Does not publish it.

1992: NSA makes a deal with Software Publishers Association.

"NSA allows encryption . . . The U.S. Department of State will grant export permission to any program that uses the RC2 or RC4 data-encryption algorithm with a key size of less than 40 bits."

1994: Someone anonymousl posts RC4 source code.

New York Times: "Widespre dissemination could compro the long-term effectiveness o system . . . [RC4] has becom the de facto coding standar for many popular software programs including Microso Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told was part of this deal that R be kept confidential,' Jim B president of RSA, said."

The RC4 stream cipher

1987: Ron Rivest designs RC4. Does not publish it.

1992: NSA makes a deal with Software Publishers Association.

"NSA allows encryption . . . The U.S. Department of State will grant export permission to any program that uses the RC2 or RC4 data-encryption algorithm with a key size of less than 40 bits."

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

4 stream cipher

on Rivest designs RC4.
t publish it.

SA makes a deal with
e Publishers Association.

lows encryption . . .
. Department of State
nt export permission
program that uses the
RC4 data-encryption
m with a key size
han 40 bits."

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

1994: N
SSL ("S
web brow
RSA Da

SSL sup
RC4 is f

cipher

designs RC4.

t.

a deal with

rs Association.

yption . . .

ent of State

ermission

at uses the

encryption

ey size

s."

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

1994: Netscape in

SSL ("Secure Soc

web browser and s

RSA Data Security

SSL supports man

RC4 is fastest ciph

C4.

th
tion.

ate

e

1994: Someone anonymously posts RC4 source code.

New York Times: “Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple’s Macintosh operating system and Lotus Notes. . . . ‘I have been told it was part of this deal that RC4 be kept confidential,’ Jim Bidzos, president of RSA, said.”

1994: Netscape introduces SSL (“Secure Sockets Layer
web browser and server “bas
RSA Data Security technolo

SSL supports many options.
RC4 is fastest cipher in SSL

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options.
RC4 is fastest cipher in SSL.

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128?

1994: Someone anonymously posts RC4 source code.

New York Times: "Widespread dissemination could compromise the long-term effectiveness of the system . . . [RC4] has become the de facto coding standard for many popular software programs including Microsoft Windows, Apple's Macintosh operating system and Lotus Notes. . . . 'I have been told it was part of this deal that RC4 be kept confidential,' Jim Bidzos, president of RSA, said."

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options.
RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts.
Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable:
1995 Roos shows that RC4 fails a basic definition of cipher security.

omeone anonymously
C4 source code.

rk Times: "Widespread
ation could compromise
-term effectiveness of the
. . . [RC4] has become
acto coding standard
y popular software
s including Microsoft
s, Apple's Macintosh
g system and Lotus
. . . 'I have been told it
t of this deal that RC4
confidential,' Jim Bidzos,
t of RSA, said."

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable: 1995 Roos shows that RC4 fails a basic definition of cipher security.

So the c
throws a
And thro

nonymously
code.

"Widespread
d compromise
ctiveness of the
has become
g standard
software
g Microsoft
Macintosh
and Lotus
been told it
eal that RC4
al,' Jim Bidzos,
said."

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable: 1995 Roos shows that RC4 fails a basic definition of cipher security.

So the crypto com
throws away 40-bi
And throws away

y

ead
mise
of the
he
d

ft
n

it
C4
idzos,

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable: 1995 Roos shows that RC4 fails a basic definition of cipher security.

So the crypto community throws away 40-bit keys? And throws away RC4?

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options.
RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts.
Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable:
1995 Roos shows that RC4 fails a basic definition of cipher security.

So the crypto community throws away 40-bit keys?
And throws away RC4?

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable: 1995 Roos shows that RC4 fails a basic definition of cipher security.

So the crypto community throws away 40-bit keys? And throws away RC4?

Here's what actually happens.

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable: 1995 Roos shows that RC4 fails a basic definition of cipher security.

So the crypto community throws away 40-bit keys? And throws away RC4?

Here's what actually happens.

1997: IEEE standardizes WEP ("Wired Equivalent Privacy") for 802.11 wireless networks.

WEP uses RC4 for encryption.

1994: Netscape introduces SSL ("Secure Sockets Layer") web browser and server "based on RSA Data Security technology".

SSL supports many options. RC4 is fastest cipher in SSL.

1995: Finney posts some examples of SSL ciphertexts. Back–Byers–Young, Doligez, Back–Brooks extract plaintexts.

Fix: RC4-128? Unacceptable: 1995 Roos shows that RC4 fails a basic definition of cipher security.

So the crypto community throws away 40-bit keys? And throws away RC4?

Here's what actually happens.

1997: IEEE standardizes WEP ("Wired Equivalent Privacy") for 802.11 wireless networks.

WEP uses RC4 for encryption.

1999: TLS ("Transport Layer Security"), new version of SSL.

RC4 is fastest cipher in TLS. TLS still supports "export keys".

etscape introduces
ecure Sockets Layer")
wser and server "based on
ta Security technology".

ports many options.
astest cipher in SSL.

inney posts some
s of SSL ciphertexts.
yers–Young, Doligez,
rooks extract plaintexts.

4-128? Unacceptable:
os shows that RC4 fails a
finition of cipher security.

So the crypto community
throws away 40-bit keys?
And throws away RC4?

Here's what actually happens.

1997: IEEE standardizes WEP
("Wired Equivalent Privacy")
for 802.11 wireless networks.

WEP uses RC4 for encryption.

1999: TLS ("Transport Layer
Security"), new version of SSL.

RC4 is fastest cipher in TLS.
TLS still supports "export keys".

More RC

1995 Wa
1997 Go
1998 Kn
Rij
2000 Go
2000 Flu
2001 Ma
2001 Flu
2001 Stu
Ru

RC4 key
⇒ pract

troduces
kets Layer")
server "based on
y technology".

y options.
her in SSL.

s some
ciphertexts.
g, Doligez,
act plaintexts.

nacceptable:
that RC4 fails a
cipher security.

So the crypto community
throws away 40-bit keys?
And throws away RC4?

Here's what actually happens.

1997: IEEE standardizes WEP ("Wired Equivalent Privacy") for 802.11 wireless networks.

WEP uses RC4 for encryption.

1999: TLS ("Transport Layer Security"), new version of SSL.

RC4 is fastest cipher in TLS.
TLS still supports "export keys".

More RC4 cryptan

1995 Wagner,
1997 Golic,
1998 Knudsen–Me
Rijmen–Verd
2000 Golic,
2000 Fluhrer–McG
2001 Mantin–Shar
2001 Fluhrer–Man
2001 Stubblefield–
Rubin.

RC4 key-output c
⇒ practical attack

”)
sed on
gy”.

.

.

,
exts.

e:
fails a
curity.

So the crypto community throws away 40-bit keys? And throws away RC4?

Here’s what actually happens.

1997: IEEE standardizes WEP (“Wired Equivalent Privacy”) for 802.11 wireless networks.

WEP uses RC4 for encryption.

1999: TLS (“Transport Layer Security”), new version of SSL.

RC4 is fastest cipher in TLS. TLS still supports “export keys”.

More RC4 cryptanalysis:

1995 Wagner,
1997 Golic,
1998 Knudsen–Meier–Prene
Rijmen–Verdoolaege,
2000 Golic,
2000 Fluhrer–McGrew,
2001 Mantin–Shamir,
2001 Fluhrer–Mantin–Shami
2001 Stubblefield–Ioannidis–
Rubin.

RC4 key-output correlations
⇒ practical attacks on WEF

So the crypto community
throws away 40-bit keys?
And throws away RC4?

Here's what actually happens.

1997: IEEE standardizes WEP
("Wired Equivalent Privacy")
for 802.11 wireless networks.

WEP uses RC4 for encryption.

1999: TLS ("Transport Layer
Security"), new version of SSL.

RC4 is fastest cipher in TLS.
TLS still supports "export keys".

More RC4 cryptanalysis:

1995 Wagner,
1997 Golic,
1998 Knudsen–Meier–Preneel–Rijmen–Verdoolaege,
2000 Golic,
2000 Fluhrer–McGrew,
2001 Mantin–Shamir,
2001 Fluhrer–Mantin–Shamir,
2001 Stubblefield–Ioannidis–Rubin.

RC4 key-output correlations
$\Rightarrow$ practical attacks on WEP.

rypto community
way 40-bit keys?
ows away RC4?

vhat actually happens.

EEE standardizes WEP
Equivalent Privacy")
11 wireless networks.

es RC4 for encryption.

LS ("Transport Layer
"), new version of SSL.

astest cipher in TLS.
l supports "export keys".

More RC4 cryptanalysis:

1995 Wagner,
1997 Golic,
1998 Knudsen–Meier–Preneel–Rijmen–Verdoolaege,
2000 Golic,
2000 Fluhrer–McGrew,
2001 Mantin–Shamir,
2001 Fluhrer–Mantin–Shamir,
2001 Stubblefield–Ioannidis–Rubin.

RC4 key-output correlations
⇒ practical attacks on WEP.

2001 Riv

"Applica
the encr
using ha
discard t
pseudo-r
be consi
proposed
of RC4 i
and extr
random
likely to
choice fo
embedde

mmunity
t keys?
RC4?

lly happens.

ardizes WEP
nt Privacy")
s networks.

r encryption.

nsport Layer
ersion of SSL.

her in TLS.
"export keys".

More RC4 cryptanalysis:

1995 Wagner,
1997 Golic,
1998 Knudsen–Meier–Preneel–
Rijmen–Verdoolaege,
2000 Golic,
2000 Fluhrer–McGrew,
2001 Mantin–Shamir,
2001 Fluhrer–Mantin–Shamir,
2001 Stubblefield–Ioannidis–
Rubin.

RC4 key-output correlations
⇒ practical attacks on WEP.

2001 Rivest respo

"Applications whic
the encryption key
using hashing and/
discard the first 25
pseudo-random ou
be considered secu
proposed attacks.
of RC4 is its excep
and extremely effic
random generator.
likely to remain th
choice for many ap
embedded systems

ns.

EP
”)
.

on.

er
SL.

.
eys”.

More RC4 cryptanalysis:

1995 Wagner,
1997 Golic,
1998 Knudsen–Meier–Preneel–
Rijmen–Verdoolaege,
2000 Golic,
2000 Fluhrer–McGrew,
2001 Mantin–Shamir,
2001 Fluhrer–Mantin–Shamir,
2001 Stubblefield–Ioannidis–
Rubin.

RC4 key-output correlations
⇒ practical attacks on WEP.

2001 Rivest response: TLS

“Applications which pre-proc
the encryption key and IV by
using hashing and/or which
discard the first 256 bytes of
pseudo-random output shou
be considered secure from th
proposed attacks. . . . The ‘
of RC4 is its exceptionally si
and extremely efficient pseud
random generator. . . . RC4
likely to remain the algorithn
choice for many applications
embedded systems.”

More RC4 cryptanalysis:

1995 Wagner,
1997 Golic,
1998 Knudsen–Meier–Preneel–Rijmen–Verdoolaege,
2000 Golic,
2000 Fluhrer–McGrew,
2001 Mantin–Shamir,
2001 Fluhrer–Mantin–Shamir,
2001 Stubblefield–Ioannidis–Rubin.

RC4 key-output correlations
$\Rightarrow$ practical attacks on WEP.

2001 Rivest response: TLS is ok.

"Applications which pre-process the encryption key and IV by using hashing and/or which discard the first 256 bytes of pseudo-random output should be considered secure from the proposed attacks. . . . The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

C4 cryptanalysis:

agner,
lic,
nudsen–Meier–Preneel–
men–Verdoolaege,
lic,
uhrer–McGrew,
antin–Shamir,
uhrer–Mantin–Shamir,
ubblefield–Ioannidis–
bin.

-output correlations
ical attacks on WEP.

2001 Rivest response: TLS is ok.

“Applications which pre-process the encryption key and IV by using hashing and/or which discard the first 256 bytes of pseudo-random output should be considered secure from the proposed attacks. . . . The ‘heart’ of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems.”

Even mo

2002 Hu
2002 Mi
2002 Pu
2003 Bit
2003 Pu
2004 Pa
2004 Ko
2004 De
2005 Ma
2005 Ma
2005 d’O
2006 Kle
2006 Do
2006 Ch

alysis:

eier–Preneel–

loolaege,

Grew,

mir,

tin–Shamir,

-Ioannidis–

orrelations

ks on WEP.

2001 Rivest response: TLS is ok.

"Applications which pre-process the encryption key and IV by using hashing and/or which discard the first 256 bytes of pseudo-random output should be considered secure from the proposed attacks. . . . The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

Even more RC4 cr

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Prenee
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d'Otreppe,
2006 Klein,
2006 Doroshenko–
2006 Chaabouni.

el–

ir,

-

P.

2001 Rivest response: TLS is ok.

"Applications which pre-process the encryption key and IV by using hashing and/or which discard the first 256 bytes of pseudo-random output should be considered secure from the proposed attacks. . . . The 'heart' of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems."

Even more RC4 cryptanalysi

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d'Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

2001 Rivest response: TLS is ok.

“Applications which pre-process the encryption key and IV by using hashing and/or which discard the first 256 bytes of pseudo-random output should be considered secure from the proposed attacks. . . . The ‘heart’ of RC4 is its exceptionally simple and extremely efficient pseudo-random generator. . . . RC4 is likely to remain the algorithm of choice for many applications and embedded systems.”

Even more RC4 cryptanalysis:

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d’Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

vest response: TLS is ok.

ations which pre-process
yption key and IV by
shing and/or which
the first 256 bytes of
random output should
dered secure from the
d attacks. . . . The 'heart'
s its exceptionally simple
remely efficient pseudo-
generator. . . . RC4 is
remain the algorithm of
or many applications and
ed systems."

Even more RC4 cryptanalysis:

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d'Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

WEP bla
million c
T. J. Ma
settled f

nse: TLS is ok.

ch pre-process
y and IV by
/or which
56 bytes of
tput should
ure from the
. . . The ‘heart’
ptionally simple
cient pseudo-
. . . RC4 is
e algorithm of
pplications and
s.”

Even more RC4 cryptanalysis:

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d’Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

WEP blamed for 2
million credit-card
T. J. Maxx. Subse
settled for $40900

is ok.

cess
y

f
ld
ne
heart'
imple
do-
is
m of
s and

Even more RC4 cryptanalysis:

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d'Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

WEP blamed for 2007 theft
million credit-card numbers
T. J. Maxx. Subsequent law
settled for $40900000.

Even more RC4 cryptanalysis:

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d'Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Even more RC4 cryptanalysis:

2002 Hulton,
2002 Mironov,
2002 Pudovkina,
2003 Bittau,
2003 Pudovkina,
2004 Paul–Preneel,
2004 KoreK,
2004 Devine,
2005 Maximov,
2005 Mantin,
2005 d'Otreppe,
2006 Klein,
2006 Doroshenko–Ryabko,
2006 Chaabouni.

WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Cryptanalysis continues:

2007 Paul–Maitra–Srivastava,
2007 Paul–Rathi–Maitra,
2007 Paul–Maitra,
2007 Vaudenay–Vuagnoux,
2007 Tews–Weinmann–Pyshkin,
2007 Tomasevic–Bojanic–Nieto-Taladriz,
2007 Maitra–Paul,
2008 Basu–Ganguly–Maitra–Paul.

ore RC4 cryptanalysis:

ulton,
ronov,
dovkina,
ttau,
dovkina,
ul–Preneel,
oreK,
evine,
aximov,
antin,
Otreppe,
ein,
oroshenko–Ryabko,
aabouni.

WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Cryptanalysis continues:

2007 Paul–Maitra–Srivastava,
2007 Paul–Rathi–Maitra,
2007 Paul–Maitra,
2007 Vaudenay–Vuagnoux,
2007 Tews–Weinmann–Pyshkin,
2007 Tomasevic–Bojanic–Nieto-Taladriz,
2007 Maitra–Paul,
2008 Basu–Ganguly–Maitra–Paul.

And mor

2008 Bi
2008 Go
2008 Ma
2008 Ak
2008 Ma
2008 Be
2009 Ba
2010 Se
Vu
2010 Vu
2011 Ma
2011 Se
Sa
2011 Pa

ryptanalysis:

l,

-Ryabko,

WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Cryptanalysis continues:

2007 Paul–Maitra–Srivastava,
2007 Paul–Rathi–Maitra,
2007 Paul–Maitra,
2007 Vaudenay–Vuagnoux,
2007 Tews–Weinmann–Pyshkin,
2007 Tomasevic–Bojanic–Nieto-Taladriz,
2007 Maitra–Paul,
2008 Basu–Ganguly–Maitra–Paul.

And more:

2008 Biham–Carm
2008 Golic–Morga
2008 Maximov–Kh
2008 Akgun–Kava
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra
2010 Sepehrdad–V Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul-
2011 Sen Gupta–M Sarkar,
2011 Paul–Maitra

s:

WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Cryptanalysis continues:

2007 Paul–Maitra–Srivastava,
2007 Paul–Rathi–Maitra,
2007 Paul–Maitra,
2007 Vaudenay–Vuagnoux,
2007 Tews–Weinmann–Pyshkin,
2007 Tomasevic–Bojanic–Nieto-Taladriz,
2007 Maitra–Paul,
2008 Basu–Ganguly–Maitra–Paul.

And more:

2008 Biham–Carmeli,
2008 Golic–Morgari,
2008 Maximov–Khovratovic
2008 Akgun–Kavak–Demirci
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra–Paul–Ta
2010 Sepehrdad–Vaudenay–Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul–Sen Gupt
2011 Sen Gupta–Maitra–Pa Sarkar,
2011 Paul–Maitra book.

WEP blamed for 2007 theft of 45 million credit-card numbers from T. J. Maxx. Subsequent lawsuit settled for $40900000.

Cryptanalysis continues:

2007 Paul–Maitra–Srivastava,
2007 Paul–Rathi–Maitra,
2007 Paul–Maitra,
2007 Vaudenay–Vuagnoux,
2007 Tews–Weinmann–Pyshkin,
2007 Tomasevic–Bojanic–Nieto-Taladriz,
2007 Maitra–Paul,
2008 Basu–Ganguly–Maitra–Paul.

And more:

2008 Biham–Carmeli,
2008 Golic–Morgari,
2008 Maximov–Khovratovich,
2008 Akgun–Kavak–Demirci,
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra–Paul–Talukdar,
2010 Sepehrdad–Vaudenay–Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul–Sen Gupta,
2011 Sen Gupta–Maitra–Paul–Sarkar,
2011 Paul–Maitra book.

amed for 2007 theft of 45
credit-card numbers from
axx. Subsequent lawsuit
for $40900000.

alysis continues:

ul–Maitra–Srivastava,
ul–Rathi–Maitra,
ul–Maitra,
udenay–Vuagnoux,
ws–Weinmann–Pyshkin,
masevic–Bojanic–
eto-Taladriz,
aitra–Paul,
su–Ganguly–Maitra–Paul.

And more:

2008 Biham–Carmeli,
2008 Golic–Morgari,
2008 Maximov–Khovratovich,
2008 Akgun–Kavak–Demirci,
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra–Paul–Talukdar,
2010 Sepehrdad–Vaudenay–Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul–Sen Gupta,
2011 Sen Gupta–Maitra–Paul–Sarkar,
2011 Paul–Maitra book.

2012 Ak
"Up to 7
web site
BEAST]
is the cu
use and
v1.0. . . .
prefer th
TLS v1.
128 is fa
processo
15% of
on the A
RC4 . . .
support

2007 theft of 45
numbers from
equent lawsuit
000.

tinues:

–Srivastava,
Maitra,
,
uagnoux,
nann–Pyshkin,
Bojanic–
iz,
,
ly–Maitra–Paul.

And more:

2008 Biham–Carmeli,
2008 Golic–Morgari,
2008 Maximov–Khovratovich,
2008 Akgun–Kavak–Demirci,
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra–Paul–Talukdar,
2010 Sepehrdad–Vaudenay–Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul–Sen Gupta,
2011 Sen Gupta–Maitra–Paul–Sarkar,
2011 Paul–Maitra book.

2012 Akamai blog
"Up to 75% of SS
web sites are vulne
BEAST] . . . Open
is the current vers
use and it only sup
v1.0. . . . the inter
prefer the RC4-128
TLS v1.0 and SSL
128 is faster and c
processor time . . .
15% of SSL/TLS
on the Akamai pla
RC4 . . . most brow
support the RC4 f

of 45
from
vsuit

a,

nkin,

-Paul.

And more:

2008 Biham–Carmeli,
2008 Golic–Morgari,
2008 Maximov–Khovratovich,
2008 Akgun–Kavak–Demirci,
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra–Paul–Talukdar,
2010 Sepehrdad–Vaudenay–Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul–Sen Gupta,
2011 Sen Gupta–Maitra–Paul–Sarkar,
2011 Paul–Maitra book.

2012 Akamai blog entry:
"Up to 75% of SSL-enabled
web sites are vulnerable [to
BEAST] . . . OpenSSL v0.9.
is the current version in broa
use and it only supports TLS
v1.0. . . . the interim fix is to
prefer the RC4-128 cipher fo
TLS v1.0 and SSL v3. . . . R
128 is faster and cheaper in
processor time . . . approxim
15% of SSL/TLS negotiatio
on the Akamai platform use
RC4 . . . most browsers can
support the RC4 fix for BEA

And more:

2008 Biham–Carmeli,
2008 Golic–Morgari,
2008 Maximov–Khovratovich,
2008 Akgun–Kavak–Demirci,
2008 Maitra–Paul.
2008 Beck–Tews,
2009 Basu–Maitra–Paul–Talukdar,
2010 Sepehrdad–Vaudenay–Vuagnoux,
2010 Vuagnoux,
2011 Maitra–Paul–Sen Gupta,
2011 Sen Gupta–Maitra–Paul–Sarkar,
2011 Paul–Maitra book.

2012 Akamai blog entry:
"Up to 75% of SSL-enabled web sites are vulnerable [to BEAST] . . . OpenSSL v0.9.8w is the current version in broad use and it only supports TLS v1.0. . . . the interim fix is to prefer the RC4-128 cipher for TLS v1.0 and SSL v3. . . . RC4-128 is faster and cheaper in processor time . . . approximately 15% of SSL/TLS negotiations on the Akamai platform use RC4 . . . most browsers can support the RC4 fix for BEAST."

re:

ham–Carmeli,
olic–Morgari,
aximov–Khovratovich,
gun–Kavak–Demirci,
aitra–Paul.
ck–Tews,
su–Maitra–Paul–Talukdar,
pehrdad–Vaudenay–
agnoux,
agnoux,
aitra–Paul–Sen Gupta,
n Gupta–Maitra–Paul–
rkar,
ul–Maitra book.

2012 Akamai blog entry:
"Up to 75% of SSL-enabled web sites are vulnerable [to BEAST] . . . OpenSSL v0.9.8w is the current version in broad use and it only supports TLS v1.0. . . . the interim fix is to prefer the RC4-128 cipher for TLS v1.0 and SSL v3. . . . RC4-128 is faster and cheaper in processor time . . . approximately 15% of SSL/TLS negotiations on the Akamai platform use RC4 . . . most browsers can support the RC4 fix for BEAST."

RC4 cry

2013 Lv-
2013 Lv-
2013 Se
Pa
2013 Sa
Ma
2013 Iso
Mo
2013 Al
Pa
Sc
2014 Pa
2015 Se
Vu

heli,
ri,
hovratovich,
k–Demirci,
.

–Paul–Talukdar,
Vaudenay–

–Sen Gupta,
Maitra–Paul–

book.

2012 Akamai blog entry:
"Up to 75% of SSL-enabled web sites are vulnerable [to BEAST] . . . OpenSSL v0.9.8w is the current version in broad use and it only supports TLS v1.0. . . . the interim fix is to prefer the RC4-128 cipher for TLS v1.0 and SSL v3. . . . RC4-128 is faster and cheaper in processor time . . . approximately 15% of SSL/TLS negotiations on the Akamai platform use RC4 . . . most browsers can support the RC4 fix for BEAST."

RC4 cryptanalysis

2013 Lv–Zhang–L
2013 Lv–Lin,
2013 Sen Gupta–M
Paul–Sarkar,
2013 Sarkar–Sen G
Maitra,
2013 Isobe–Ohigas
Morii,
2013 AlFardan–Be
Paterson–Po
Schuldt,
2014 Paterson–Str
2015 Sepehrdad–S
Vuagnoux.

h,
i,
ukdar,
ta,
ul–

2012 Akamai blog entry:
"Up to 75% of SSL-enabled web sites are vulnerable [to BEAST] . . . OpenSSL v0.9.8w is the current version in broad use and it only supports TLS v1.0. . . . the interim fix is to prefer the RC4-128 cipher for TLS v1.0 and SSL v3. . . . RC4-128 is faster and cheaper in processor time . . . approximately 15% of SSL/TLS negotiations on the Akamai platform use RC4 . . . most browsers can support the RC4 fix for BEAST."

RC4 cryptanalysis continues

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Me
Paul–Sarkar,
2013 Sarkar–Sen Gupta–Pau
Maitra,
2013 Isobe–Ohigashi–Watan
Morii,
2013 AlFardan–Bernstein–
Paterson–Poettering–
Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaud
Vuagnoux.

2012 Akamai blog entry:
"Up to 75% of SSL-enabled web sites are vulnerable [to BEAST] . . . OpenSSL v0.9.8w is the current version in broad use and it only supports TLS v1.0. . . . the interim fix is to prefer the RC4-128 cipher for TLS v1.0 and SSL v3. . . . RC4-128 is faster and cheaper in processor time . . . approximately 15% of SSL/TLS negotiations on the Akamai platform use RC4 . . . most browsers can support the RC4 fix for BEAST."

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

kamai blog entry:
75% of SSL-enabled
s are vulnerable [to
. . . OpenSSL v0.9.8w
urrent version in broad
it only supports TLS
the interim fix is to
he RC4-128 cipher for
0 and SSL v3. . . . RC4-
ster and cheaper in
r time . . . approximately
SSL/TLS negotiations
Akamai platform use
most browsers can
the RC4 fix for BEAST."

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

Maybe t

2015 Ma
2015 Ga
var
"R
2015 Va
"R

entry:
L-enabled
erable [to
SSL v0.9.8w
ion in broad
pports TLS
im fix is to
8 cipher for
v3. . . . RC4-
cheaper in
approximately
negotiations
tform use
wsers can
ix for BEAST."

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

Maybe the final st

2015 Mantin "Bar
2015 Garman–Pat
van der Merv
"RC4 must c
2015 Vanhoef–Pie
"RC4 no mo

8w
ad
S
o
or
RC4-

ately
ns

AST."

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",
2015 Garman–Paterson–van der Merwe "RC4 must die",
2015 Vanhoef–Piessens "RC4 no more".

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",
2015 Garman–Paterson–van der Merwe "RC4 must die",
2015 Vanhoef–Piessens "RC4 no more".

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",
2015 Garman–Paterson–van der Merwe "RC4 must die",
2015 Vanhoef–Piessens "RC4 no more".

Meanwhile IETF publishes RFC 7465 ("RC4 die die die"), prohibiting RC4 in TLS.

RC4 cryptanalysis continues:

2013 Lv–Zhang–Lin,
2013 Lv–Lin,
2013 Sen Gupta–Maitra–Meier–Paul–Sarkar,
2013 Sarkar–Sen Gupta–Paul–Maitra,
2013 Isobe–Ohigashi–Watanabe–Morii,
2013 AlFardan–Bernstein–Paterson–Poettering–Schuldt,
2014 Paterson–Strefler,
2015 Sepehrdad–Sušil–Vaudenay–Vuagnoux.

Maybe the final straws:

2015 Mantin “Bar Mitzvah”,
2015 Garman–Paterson–van der Merwe “RC4 must die”,
2015 Vanhoef–Piessens “RC4 no more”.

Meanwhile IETF publishes RFC 7465 (“RC4 die die die”), prohibiting RC4 in TLS.

2015.09.01: Google, Microsoft, Mozilla say that in 2016 their browsers will no longer allow RC4.

ptanalysis continues:

–Zhang–Lin,

–Lin,

n Gupta–Maitra–Meier–

ul–Sarkar,

rkar–Sen Gupta–Paul–

aitra,

be–Ohigashi–Watanabe–

orii,

Fardan–Bernstein–

terson–Poettering–

huldt,

terson–Strefler,

pehrdad–Sušil–Vaudenay–

agnoux.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",

2015 Garman–Paterson–van der Merwe "RC4 must die",

2015 Vanhoef–Piessens "RC4 no more".

Meanwhile IETF publishes RFC 7465 ("RC4 die die die"), prohibiting RC4 in TLS.

2015.09.01: Google, Microsoft, Mozilla say that in 2016 their browsers will no longer allow RC4.

Another

2005 Tr

65ms to

used for

Attack p

but with

continues:

in,

Maitra–Meier–

Gupta–Paul–

shi–Watanabe–

ernstein–
ettering–

refler,
Sušil–Vaudenay–

Maybe the final straws:

2015 Mantin “Bar Mitzvah”,
2015 Garman–Paterson–
van der Merwe
“RC4 must die”,
2015 Vanhoef–Piessens
“RC4 no more”.

Meanwhile IETF publishes
RFC 7465 (“RC4 die die die”),
prohibiting RC4 in TLS.

2015.09.01: Google, Microsoft,
Mozilla say that in 2016 their
browsers will no longer allow RC4.

Another example:

2005 Tromer–Osvi
65ms to steal Linu
used for hard-disk
Attack process on
but without privile

:

eier–

ul–

nabe–

enay–

Maybe the final straws:

2015 Mantin "Bar Mitzvah",
2015 Garman–Paterson–
van der Merwe
"RC4 must die",
2015 Vanhoef–Piessens
"RC4 no more".

Meanwhile IETF publishes RFC 7465 ("RC4 die die die"), prohibiting RC4 in TLS.

2015.09.01: Google, Microsoft, Mozilla say that in 2016 their browsers will no longer allow RC4.

Another example: timing at

2005 Tromer–Osvik–Shamir:
65ms to steal Linux AES key
used for hard-disk encryption
Attack process on same CPU
but without privileges.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",
2015 Garman–Paterson–
van der Merwe
"RC4 must die",
2015 Vanhoef–Piessens
"RC4 no more".

Meanwhile IETF publishes
RFC 7465 ("RC4 die die die"),
prohibiting RC4 in TLS.

2015.09.01: Google, Microsoft,
Mozilla say that in 2016 their
browsers will no longer allow RC4.

Another example: timing attacks

2005 Tromer–Osvik–Shamir:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",
2015 Garman–Paterson–van der Merwe "RC4 must die",
2015 Vanhoef–Piessens "RC4 no more".

Meanwhile IETF publishes RFC 7465 ("RC4 die die die"), prohibiting RC4 in TLS.

2015.09.01: Google, Microsoft, Mozilla say that in 2016 their browsers will no longer allow RC4.

Another example: timing attacks

2005 Tromer–Osvik–Shamir: 65ms to steal Linux AES key used for hard-disk encryption. Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

he final straws:

antin “Bar Mitzvah”,
rman–Paterson–
n der Merwe
C4 must die”,
nhoef–Piessens
C4 no more”.

ile IETF publishes
65 (“RC4 die die die”),
ng RC4 in TLS.

01: Google, Microsoft,
say that in 2016 their
s will no longer allow RC4.

Another example: timing attacks

2005 Tromer–Osvik–Shamir:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel’s secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms: compute key from timings.

2011 Br
minutes
machine
Secret b
influence

raws:

Mitzvah",
erson–
we
die",
ssens
re".

ublishes
die die die"),
TLS.

le, Microsoft,
2016 their
nger allow RC4.

Another example: timing attacks

2005 Tromer–Osvik–Shamir:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms: compute key from timings.

2011 Brumley–Tu
minutes to steal a
machine's OpenS
Secret branch con
influence timings.

,

”),

oft,
ir
/ RC4.

Another example: timing attacks

2005 Tromer–Osvik–Shamir:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel’s secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms: compute key from timings.

2011 Brumley–Tuveri:
minutes to steal another
machine’s OpenSSL ECDSA
Secret branch conditions
influence timings.

Another example: timing attacks

2005 Tromer–Osvik–Shamir: 65ms to steal Linux AES key used for hard-disk encryption. Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Another example: timing attacks

2005 Tromer–Osvik–Shamir:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms: compute key from timings.

2011 Brumley–Tuveri:
minutes to steal another
machine's OpenSSL ECDSA key.
Secret branch conditions
influence timings.

Most cryptographic software
has many more small-scale
variations in timing:
e.g., `memcmp` for IPsec MACs.

Another example: timing attacks

2005 Tromer–Osvik–Shamir: 65ms to steal Linux AES key used for hard-disk encryption. Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms: compute key from timings.

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

example: timing attacks

omer–Osvik–Shamir:
steal Linux AES key
hard-disk encryption.
process on same CPU
out privileges.

all AES implementations
lookup tables.
secret AES key
es table-load addresses,
ng CPU cache state,
ng measurable timings
ttack process.
ompute key from timings.

2011 Brumley–Tuveri:
minutes to steal another
machine's OpenSSL ECDSA key.
Secret branch conditions
influence timings.

Most cryptographic software
has many more small-scale
variations in timing:
e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g.
2014 van de Pol–Smart–Yarom
extracted Bitcoin secret keys
from 25 OpenSSL signatures.

2008 RF
Layer Se
Version
small tin
performa
extent o
fragmen
be large
due to t
existing
of the ti

timing attacks

ik–Shamir:
ux AES key
encryption.
same CPU
eges.

nplementations
bles.
ES key
ad addresses,
ache state,
rable timings
ess.
ey from timings.

2011 Brumley–Tuveri:
minutes to steal another
machine's OpenSSL ECDSA key.
Secret branch conditions
influence timings.

Most cryptographic software
has many more small-scale
variations in timing:
e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g.
2014 van de Pol–Smart–Yarom
extracted Bitcoin secret keys
from 25 OpenSSL signatures.

2008 RFC 5246 "T
Layer Security (TL
Version 1.2": "Th
small timing chann
performance depen
extent on the size
fragment, but it is
be large enough to
due to the large bl
existing MACs and
of the timing signa

tacks

t
y
n.
U

tions

ses,
e,
ngs

mings.

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

2008 RFC 5246 "The Trans
Layer Security (TLS) Protoc
Version 1.2": "This leaves a
small timing channel, since
performance depends to som
extent on the size of the dat
fragment, but it is not belie
be large enough to be exploi
due to the large block size o
existing MACs and the smal
of the timing signal."

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2011 Brumley–Tuveri: minutes to steal another machine's OpenSSL ECDSA key. Secret branch conditions influence timings.

Most cryptographic software has many more small-scale variations in timing: e.g., `memcmp` for IPsec MACs.

Many more timing attacks: e.g. 2014 van de Pol–Smart–Yarom extracted Bitcoin secret keys from 25 OpenSSL signatures.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

umley–Tuveri:
to steal another
's OpenSSL ECDSA key.
ranch conditions
e timings.

yptographic software
y more small-scale
s in timing:
ncmp for IPsec MACs.

ore timing attacks: e.g.
n de Pol–Smart–Yarom
d Bitcoin secret keys
OpenSSL signatures.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

Interesti

All of th
wonderfu

veri:
nother
SL ECDSA key.
ditions

ic software
nall-scale
g:
Psec MACs.

; attacks: e.g.
Smart–Yarom
secret keys
signatures.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

Interesting vs. bor

All of this excitem
wonderful for cryp

key.

e

s.

e.g.

om

s

s.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researc*

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*: continually forced to panic, vulnerable to attacks, uncertain what to do next.

2008 RFC 5246 "The Transport Layer Security (TLS) Protocol, Version 1.2": "This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal."

2013 AlFardan–Paterson "Lucky Thirteen: breaking the TLS and DTLS record protocols": exploit these timings; steal plaintext.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*:
continually forced to panic,
vulnerable to attacks,
uncertain what to do next.

The crypto users' fantasy
is **boring crypto**:
crypto that simply works,
solidly resists attacks,
never needs any upgrades.

FC 5246 "The Transport
ecurity (TLS) Protocol,
1.2": "This leaves a
ning channel, since MAC
ance depends to some
n the size of the data
t, but it is not believed to
enough to be exploitable,
he large block size of
MACs and the small size
ming signal."

Fardan–Paterson "Lucky
: breaking the TLS and
ecord protocols": exploit
mings; steal plaintext.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*:
continually forced to panic,
vulnerable to attacks,
uncertain what to do next.

The crypto users' fantasy is **boring crypto**:
crypto that simply works,
solidly resists attacks,
never needs any upgrades.

What wi
the cryp
some cry
actually

The Transport
LS) Protocol,
is leaves a
nel, since MAC
nds to some
of the data
not believed to
be exploitable,
lock size of
d the small size
al."

terson "Lucky
the TLS and
ocols": exploit
al plaintext.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*:
continually forced to panic,
vulnerable to attacks,
uncertain what to do next.

The crypto users' fantasy is **boring crypto**:
crypto that simply works,
solidly resists attacks,
never needs any upgrades.

What will happen
the crypto users c
some crypto resea
actually create bor

port
col,
n
MAC
ne
ta
ved to
itable,
of
l size

ucky
and
ploit
t.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*: continually forced to panic, vulnerable to attacks, uncertain what to do next.

The crypto users' fantasy is **boring crypto**: crypto that simply works, solidly resists attacks, never needs any upgrades.

What will happen if the crypto users convince some crypto researchers to actually create boring crypto

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*: continually forced to panic, vulnerable to attacks, uncertain what to do next.

The crypto users' fantasy is **boring crypto**: crypto that simply works, solidly resists attacks, never needs any upgrades.

What will happen if the crypto users convince some crypto researchers to actually create boring crypto?

Interesting vs. boring crypto

All of this excitement is
wonderful for crypto *researchers*.

The only people suffering
are the crypto *users*:
continually forced to panic,
vulnerable to attacks,
uncertain what to do next.

The crypto users' fantasy
is **boring crypto**:
crypto that simply works,
solidly resists attacks,
never needs any upgrades.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

Interesting vs. boring crypto

All of this excitement is
wonderful for crypto *researchers*.

The only people suffering
are the crypto *users*:
continually forced to panic,
vulnerable to attacks,
uncertain what to do next.

The crypto users' fantasy
is **boring crypto**:
crypto that simply works,
solidly resists attacks,
never needs any upgrades.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

This is an existential threat
against future crypto research.

Interesting vs. boring crypto

All of this excitement is wonderful for crypto *researchers*.

The only people suffering are the crypto *users*: continually forced to panic, vulnerable to attacks, uncertain what to do next.

The crypto users' fantasy is **boring crypto**: crypto that simply works, solidly resists attacks, never needs any upgrades.

What will happen if the crypto users convince some crypto researchers to actually create boring crypto?

No more real-world attacks. No more emergency upgrades. Limited audience for any minor attack improvements and for replacement crypto.

This is an existential threat against future crypto research.

Is this the real life?
Is this just fantasy?

ng vs. boring crypto

is excitement is
ul for crypto *researchers*.

y people suffering
crypto *users*:
lly forced to panic,
le to attacks,
n what to do next.

pto users' fantasy
g crypto:
hat simply works,
esists attacks,
eeds any upgrades.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

This is an existential threat
against future crypto research.

Is this the real life?
Is this just fantasy?

Crypto c

Again co

Many in
How do
How car
How car
to influe
affect tir

ing crypto

ent is
to *researchers*.

uffering
rs:
to panic,
cks,
do next.

fantasy

works,
cks,
pgrades.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

This is an existential threat
against future crypto research.

Is this the real life?
Is this just fantasy?

Crypto can be bor

Again consider tim

Many interesting c
How do secrets aff
How can attacker
How can attacker
to influence how s
affect timings? Et

hers.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

This is an existential threat
against future crypto research.

Is this the real life?
Is this just fantasy?

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timing
How can attacker see timing
How can attacker choose inp
to influence how secrets
affect timings? Et cetera.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

This is an existential threat
against future crypto research.

Is this the real life?
Is this just fantasy?

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs
to influence how secrets
affect timings? Et cetera.

What will happen if
the crypto users convince
some crypto researchers to
actually create boring crypto?

No more real-world attacks.
No more emergency upgrades.
Limited audience for any
minor attack improvements
and for replacement crypto.

This is an existential threat
against future crypto research.

Is this the real life?
Is this just fantasy?

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs
to influence how secrets
affect timings? Et cetera.

The boring-crypto alternative:
crypto software is built from
instructions that have no data
flow from inputs to timings.
Obviously constant time.

ill happen if
to users convince
ypto researchers to
create boring crypto?

e real-world attacks.
e emergency upgrades.
audience for any
ttack improvements
replacement crypto.

an existential threat
future crypto research.

he real life?
ust fantasy?

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs to influence how secrets affect timings? Et cetera.

The boring-crypto alternative: crypto software is built from instructions that have no data flow from inputs to timings. Obviously constant time.

Another
"2^{80} sec

2^{80} mult
about 2^{2}
Bluffdale

if
onvince
rchers to
ring crypto?

d attacks.
cy upgrades.
for any
ovements
nt crypto.

tial threat
pto research.

?
y?

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs to influence how secrets affect timings? Et cetera.

The boring-crypto alternative: crypto software is built from instructions that have no data flow from inputs to timings. Obviously constant time.

Another example:
"2^{80} security" is in

2^{80} mults on mass
about 2^{22} watt-yea
Bluffdale: 2^{26} wat

o?

es.

ch.

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs
to influence how secrets
affect timings? Et cetera.

The boring-crypto alternative:
crypto software is built from
instructions that have no data
flow from inputs to timings.
Obviously constant time.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market G
about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs to influence how secrets affect timings? Et cetera.

The boring-crypto alternative: crypto software is built from instructions that have no data flow from inputs to timings. Obviously constant time.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs: about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs to influence how secrets affect timings? Et cetera.

The boring-crypto alternative: crypto software is built from instructions that have no data flow from inputs to timings. Obviously constant time.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs: about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

Crypto can be boring

Again consider timing leaks.

Many interesting questions:
How do secrets affect timings?
How can attacker see timings?
How can attacker choose inputs to influence how secrets affect timings? Et cetera.

The boring-crypto alternative: crypto software is built from instructions that have no data flow from inputs to timings. Obviously constant time.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs: about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

"2^{128} security" is boring.

can be boring

onsider timing leaks.

teresting questions:
 secrets affect timings?
n attacker see timings?
n attacker choose inputs
nce how secrets
mings? Et cetera.

ing-crypto alternative:
oftware is built from
ons that have no data
m inputs to timings.
ly constant time.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs:
about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than
single-precision mults? Easier?
Can the attack cost be shared
across targets, as in Logjam?
Every speedup is important.

"2^{128} security" is boring.

NIST EC
see, e.g.
in PS3 E

Ed2551

ing

ning leaks.

questions:
fect timings?
see timings?
choose inputs
ecrets
cetera.

alternative:
built from
ave no data
o timings.
t time.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs:
about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

"2^{128} security" is boring.

NIST ECC is inter
see, e.g., how keys
in PS3 ECDSA an

Ed25519 and X25

gs?
gs?
puts

e:

ta

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs:
about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

"2^{128} security" is boring.

NIST ECC is interesting:
see, e.g., how keys were exp
in PS3 ECDSA and Java EC

Ed25519 and X25519: borin

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs: about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

"2^{128} security" is boring.

NIST ECC is interesting:
see, e.g., how keys were exposed in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Another example:
"2^{80} security" is interesting.

2^{80} mults on mass-market GPUs: about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is "2^{80} security" really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

"2^{128} security" is boring.

NIST ECC is interesting: see, e.g., how keys were exposed in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting: expands the attack surface, complicates implementations, complicates security analysis.

One True Cipher Suite: boring.

Another example:
“2^{80} security” is interesting.

2^{80} mults on mass-market GPUs: about 2^{22} watt-years.
Bluffdale: 2^{26} watts.

Is “2^{80} security” really 2^{85}? 2^{75}?
Are the individual ops harder than single-precision mults? Easier?
Can the attack cost be shared across targets, as in Logjam?
Every speedup is important.

“2^{128} security” is boring.

NIST ECC is interesting: see, e.g., how keys were exposed in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto “agility” is interesting: expands the attack surface, complicates implementations, complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers actually ensure correctness?

example:
urity" is interesting.

ts on mass-market GPUs:
^{22} watt-years.
e: 2^{26} watts.

security" really 2^{85}? 2^{75}?
individual ops harder than
recision mults? Easier?
attack cost be shared
argets, as in Logjam?
peedup is important.

curity" is boring.

NIST ECC is interesting:
see, e.g., how keys were exposed
in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting:
expands the attack surface,
complicates implementations,
complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers
actually ensure correctness?

Bugs tri
usually a

nteresting.

s-market GPUs:
ars.
ts.

eally 2^{85}? 2^{75}?
ops harder than
ults? Easier?
st be shared
in Logjam?
mportant.

boring.

NIST ECC is interesting:
see, e.g., how keys were exposed
in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting:
expands the attack surface,
complicates implementations,
complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers
actually ensure correctness?

Bugs triggered by
usually aren't caug

GPUs:

2^{75}?

r than

er?

ed

?

NIST ECC is interesting:
see, e.g., how keys were exposed in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting:
expands the attack surface,
complicates implementations,
complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers actually ensure correctness?

Bugs triggered by very rare
usually aren't caught by test

NIST ECC is interesting:
see, e.g., how keys were exposed
in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting:
expands the attack surface,
complicates implementations,
complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers
actually ensure correctness?

Bugs triggered by very rare inputs
usually aren't caught by testing.

NIST ECC is interesting:
see, e.g., how keys were exposed
in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting:
expands the attack surface,
complicates implementations,
complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers
actually ensure correctness?

Bugs triggered by very rare inputs
usually aren't caught by testing.

Block-cipher implementations
typically have no such bugs.

NIST ECC is interesting:
see, e.g., how keys were exposed in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting: expands the attack surface, complicates implementations, complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers actually ensure correctness?

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

NIST ECC is interesting:
see, e.g., how keys were exposed in PS3 ECDSA and Java ECDH.

Ed25519 and X25519: boring.

Crypto "agility" is interesting: expands the attack surface, complicates implementations, complicates security analysis.

One True Cipher Suite: boring.

Incorrect software: interesting.

Correct software: boring.
Can boring-crypto researchers actually ensure correctness?

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

CC is interesting:
, how keys were exposed
ECDSA and Java ECDH.

9 and X25519: boring.

'agility" is interesting:
the attack surface,
ates implementations,
ates security analysis.

e Cipher Suite: boring.

t software: interesting.

software: boring.
ing-crypto researchers
ensure correctness?

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

Typically
are caug
Can this

esting:
s were exposed
d Java ECDH.

519: boring.

interesting:
k surface,
mentations,
ty analysis.

Suite: boring.

interesting.

boring.
researchers
rrectness?

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

Typically these lim
are caught by care
Can this be autom

osed
CDH.

g.

g:

s,
s.

ng.

ng.

rs

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

Typically these limb overflow
are caught by careful audits.
Can this be automated?

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

Typically these limb overflows are caught by careful audits. Can this be automated?

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

Typically these limb overflows are caught by careful audits. Can this be automated?

2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": proof of correctness of thousands of lines of asm for X25519 main loop.

Bugs triggered by very rare inputs usually aren't caught by testing.

Block-cipher implementations typically have no such bugs.

Much bigger issue for bigint software. Integers are split into "limbs" stored in CPU words; typical tests fail to find extreme values of limbs, fail to catch slight overflows inside arithmetic.

2011 Brumley–Barbosa–Page–Vercauteren exploited a limb overflow in OpenSSL.

Typically these limb overflows are caught by careful audits. Can this be automated?

2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": proof of correctness of thousands of lines of asm for X25519 main loop.

Still very far from automatic: huge portion of proof was *checked* by computer but *written* by hand.

Per proof: many hours of CPU time; many hours of human time.

ggered by very rare inputs
aren't caught by testing.

pher implementations
have no such bugs.

gger issue for bigint
. Integers are split into
stored in CPU words;
ests fail to find extreme
f limbs, fail to catch slight
s inside arithmetic.

umley–Barbosa–Page–
eren exploited a
erflow in OpenSSL.

Typically these limb overflows are caught by careful audits. Can this be automated?

2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": proof of correctness of thousands of lines of asm for X25519 main loop.

Still very far from automatic: huge portion of proof was *checked* by computer but *written* by hand.

Per proof: many hours of CPU time; many hours of human time.

2015 Be
gfverif
far less t
Usable p
process

Latest n
correctn
impleme

CPU tim
141 seco

Human
annotati
Working

very rare inputs
ght by testing.

ementations
such bugs.

for bigint
are split into
CPU words;
find extreme
il to catch slight
rithmetic.

rbosa–Page–
ited a
OpenSSL.

Typically these limb overflows are caught by careful audits. Can this be automated?

2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": proof of correctness of thousands of lines of asm for X25519 main loop.

Still very far from automatic: huge portion of proof was *checked* by computer but *written* by hand.

Per proof: many hours of CPU time; many hours of human time.

2015 Bernstein–Sc
`gfverif`, in progr
far less time per p
Usable part of dev
process for ECC sc

Latest news: finish
correctness for `re`
implementation of

CPU time per proo
141 seconds on my

Human time per p
annotations for ea
Working on autom

inputs
ting.

ns

nto
s;
eme
slight

e–

Typically these limb overflows are caught by careful audits. Can this be automated?

2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": proof of correctness of thousands of lines of asm for X25519 main loop.

Still very far from automatic: huge portion of proof was *checked* by computer but *written* by hand.

Per proof: many hours of CPU time; many hours of human time.

2015 Bernstein–Schwabe `gfverif`, in progress: far less time per proof. Usable part of development process for ECC software.

Latest news: finished provin correctness for `ref10` implementation of X25519.

CPU time per proof: 141 seconds on my laptop.

Human time per proof: annotations for each field op Working on automating this

Typically these limb overflows are caught by careful audits. Can this be automated?

2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": proof of correctness of thousands of lines of asm for X25519 main loop.

Still very far from automatic: huge portion of proof was *checked* by computer but *written* by hand.

Per proof: many hours of CPU time; many hours of human time.

2015 Bernstein–Schwabe `gfverif`, in progress: far less time per proof. Usable part of development process for ECC software.

Latest news: finished proving correctness for `ref10` implementation of X25519.

CPU time per proof: 141 seconds on my laptop.

Human time per proof: annotations for each field op. Working on automating this.