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.

TLS is not boring crypto.

New attacks!

Disputes about security!

Improved attacks!

Proposed fixes!

Even better attacks!

Emergency upgrades!

Different attacks!

New protocol versions!

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.

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.

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."

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: 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: 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: 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?

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

Here's what actually happens.

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.

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.

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.

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

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.

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.

Maybe the final straws:

2015 Mantin "Bar Mitzvah",

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

2015 Vanhoef–Piessens "RC4 no more".

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.

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.

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.

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.

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."

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*.

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.

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?

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.

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.

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.

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.

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.

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.

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.

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.

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

Block-cipher implementations typically have no such bugs.

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.

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?

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.

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.