Cache-timing attacks on AES

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF CCR–9983950
Alfred P. Sloan Foundation

`http://cr.yp.to/papers.html#cachetiming`, 2005:

"This paper reports successful extraction of a complete AES key from a network server on another computer. The targeted server used its key solely to encrypt data using the OpenSSL AES implementation on a Pentium III."

All code included in paper.
Easily reproducible.

cks on AES

is at Chicago
0
undation

`http://cr.yp.to/papers.html#cachetiming`, 2005:

"This paper reports successful
extraction of a complete AES key
from a network server
on another computer.
The targeted server used its key
solely to encrypt data using the
OpenSSL AES implementation
on a Pentium III."

All code included in paper.
Easily reproducible.

Attack extracted t
from *timings* of th

AES selectors (US
"Report on the de
the Advanced Enc
(AES)," 2001:

"In some environm
timing attacks can
against operations
in different amoun
depending on thei

`http://cr.yp.to/papers.html#cachetiming`, 2005:

"This paper reports successful extraction of a complete AES key from a network server on another computer. The targeted server used its key solely to encrypt data using the OpenSSL AES implementation on a Pentium III."

All code included in paper.
Easily reproducible.

Attack extracted the AES key from *timings* of the server.

AES selectors (US NIST), "Report on the development of the Advanced Encryption Standard (AES)," 2001:

"In some environments, timing attacks can be effected against operations that execute in different amounts of time, depending on their arguments.

o/papers.html
005:

ts successful
mplete AES key
rver
ter.
er used its key
lata using the
plementation

in paper.
e.

Attack extracted the AES key
from *timings* of the server.

AES selectors (US NIST),
"Report on the development of
the Advanced Encryption Standard
(AES)," 2001:

"In some environments,
timing attacks can be effected
against operations that execute
in different amounts of time,
depending on their arguments.

"A general defense
timing attacks is t
each encryption an
operation runs in t
of time. . . .

"Table lookup: no
timing attacks . . .

"Multiplication/di
or variable shift/ro
most difficult to d

Attack extracted the AES key
from *timings* of the server.

AES selectors (US NIST),
"Report on the development of
the Advanced Encryption Standard
(AES)," 2001:

"In some environments,
timing attacks can be effected
against operations that execute
in different amounts of time,
depending on their arguments.

"A general defense against
timing attacks is to ensure that
each encryption and decryption
operation runs in the same amount
of time. . . .

"Table lookup: not vulnerable to
timing attacks . . .

"Multiplication/division/squaring
or variable shift/rotation:
most difficult to defend . . .

he AES key
e server.

NIST),
velopment of
ryption Standard

nents,
be effected
that execute
ts of time,
r arguments.

"A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. . . .

"Table lookup: not vulnerable to timing attacks . . .

"Multiplication/division/squaring or variable shift/rotation: most difficult to defend . . .

"Rijndael and Serp
Boolean operation
and fixed shifts/ro
operations are the
against attacks. . .

"Finalist profiles. .
operations used by
among the easiest
power and timing
Rijndael appears t
major speed advan
competitors when
protections are co

"A general defense against timing attacks is to ensure that each encryption and decryption operation runs in the same amount of time. . . .

"Table lookup: not vulnerable to timing attacks . . .

"Multiplication/division/squaring or variable shift/rotation: most difficult to defend . . .

"Rijndael and Serpent use only Boolean operations, table lookups, and fixed shifts/rotations. These operations are the easiest to defend against attacks. . . .

"Finalist profiles. . . . The operations used by Rijndael are among the easiest to defend against power and timing attacks. . . . Rijndael appears to gain a major speed advantage over its competitors when such protections are considered. . . .

e against
o ensure that
nd decryption
the same amount

t vulnerable to

vision/squaring
otation:
efend . . .

"Rijndael and Serpent use only Boolean operations, table lookups, and fixed shifts/rotations. These operations are the easiest to defend against attacks. . . .

"Finalist profiles. . . . The operations used by Rijndael are among the easiest to defend against power and timing attacks. . . . Rijndael appears to gain a major speed advantage over its competitors when such protections are considered. . . .

"NIST judged Rijn
best overall algorit
Rijndael appears t
consistently good
Its key setup time
its key agility is go
Rijndael's operatio
easiest to defend a
timing attacks. . . .
Rijndael's internal
appears to have go
benefit from instru
parallelism." (Emp

"Rijndael and Serpent use only Boolean operations, table lookups, and fixed shifts/rotations. These operations are the easiest to defend against attacks. . . .

"Finalist profiles. . . . The operations used by Rijndael are among the easiest to defend against power and timing attacks. . . . Rijndael appears to gain a major speed advantage over its competitors when such protections are considered. . . .

"NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be a consistently good performer . . . Its key setup time is excellent, and its key agility is good. . . . *Rijndael's operations are among the easiest to defend against power and timing attacks.* . . . Finally, Rijndael's internal round structure appears to have good potential to benefit from instruction-level parallelism." (Emphasis added.)

pent use only
s, table lookups,
tations. These
easiest to defend
.

.. The
Rijndael are
to defend against
attacks. ...
o gain a
ntage over its
such
nsidered. ...

"NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be a consistently good performer ... Its key setup time is excellent, and its key agility is good. ... *Rijndael's operations are among the easiest to defend against power and timing attacks.* ... Finally, Rijndael's internal round structure appears to have good potential to benefit from instruction-level parallelism." (Emphasis added.)

AES designers (Da
"Resistance agains
attacks: a compar
AES proposals," 1

"Table lookups: T
not susceptible to
attack. ... Favora
that use only logic
table-lookups and
that are therefore
to secure. The alg
group are Crypton
Magenta, Rijndael

"NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be a consistently good performer . . . Its key setup time is excellent, and its key agility is good. . . . *Rijndael's operations are among the easiest to defend against power and timing attacks.* . . . Finally, Rijndael's internal round structure appears to have good potential to benefit from instruction-level parallelism." (Emphasis added.)

AES designers (Daemen, Rijmen), "Resistance against implementation attacks: a comparative study of the AES proposals," 1999:

"Table lookups: This instruction is not susceptible to a timing attack. . . . Favorable: Algorithms that use only logical operations, table-lookups and fixed shifts, and that are therefore relatively easy to secure. The algorithms of this group are Crypton, DEAL, Magenta, Rijndael and Serpent."

ndael to be the
thm for the AES.
o be a
performer . . .
is excellent, and
ood. . . .
ons are among the
against power and
. Finally,
round structure
ood potential to
uction-level
phasis added.)

AES designers (Daemen, Rijmen), "Resistance against implementation attacks: a comparative study of the AES proposals," 1999:

"Table lookups: This instruction is not susceptible to a timing attack. . . . Favorable: Algorithms that use only logical operations, table-lookups and fixed shifts, and that are therefore relatively easy to secure. The algorithms of this group are Crypton, DEAL, Magenta, Rijndael and Serpent."

AES designers, 19
"should take into
measures to be ta
these attacks."

(Amazing change
Timing attacks are
for cryptographic
Bruce Schneier, 20
problem is that sid
are practical again
anything, so it did
into consideration.

AES designers (Daemen, Rijmen), "Resistance against implementation attacks: a comparative study of the AES proposals," 1999:

"Table lookups: This instruction is not susceptible to a timing attack. ... Favorable: Algorithms that use only logical operations, table-lookups and fixed shifts, and that are therefore relatively easy to secure. The algorithms of this group are Crypton, DEAL, Magenta, Rijndael and Serpent."

AES designers, 1999: Speed reports "should take into account the measures to be taken to thwart these attacks."

(Amazing change of position, 2005: Timing attacks are "irrelevant for cryptographic design."

Bruce Schneier, 2005: "The problem is that side-channel attacks are practical against pretty much anything, so it didn't really enter into consideration.")

aemen, Rijmen),
st implementation
rative study of the
999:

This instruction is
a timing
ble: Algorithms
cal operations,
fixed shifts, and
relatively easy
gorithms of this
, DEAL,
and Serpent."

AES designers, 1999: Speed reports "should take into account the measures to be taken to thwart these attacks."

(Amazing change of position, 2005: Timing attacks are "irrelevant for cryptographic design."

Bruce Schneier, 2005: "The problem is that side-channel attacks are practical against pretty much anything, so it didn't really enter into consideration.")

The problem in a
Daemen, Rijmen,
Variable-index tab
vulnerable to timi

AES does many lo
`table[k[3]^n[3]`
that depends on `k`
leaking `k[3]`.

It is extremely diff
to avoid this leak
on Pentium, Athlo
without gigantic s

AES designers, 1999: Speed reports "should take into account the measures to be taken to thwart these attacks."

(Amazing change of position, 2005: Timing attacks are "irrelevant for cryptographic design." Bruce Schneier, 2005: "The problem is that side-channel attacks are practical against pretty much anything, so it didn't really enter into consideration.")

The problem in a nutshell: Daemen, Rijmen, NIST were wrong. Variable-index table lookup *is* vulnerable to timing attacks.

AES does many lookups such as `table[k[3]^n[3]]`, taking time that depends on `k[3]^n[3]`, leaking `k[3]`.

It is extremely difficult to avoid this leak on Pentium, Athlon, etc. without gigantic slowdowns.

99: Speed reports
account the
ken to thwart

of position, 2005:
e "irrelevant
design."
005: "The
de-channel attacks
st pretty much
n't really enter
")

The problem in a nutshell:
Daemen, Rijmen, NIST were wrong.
Variable-index table lookup *is*
vulnerable to timing attacks.

AES does many lookups such as
`table[k[3]^n[3]]`, taking time
that depends on `k[3]^n[3]`,
leaking `k[3]`.

It is extremely difficult
to avoid this leak
on Pentium, Athlon, etc.
without gigantic slowdowns.

Naive reaction:
"Oh, of course.
A table lookup tak
if it misses the cac

Mentioned by 199
2000 Kelsey Schne

Exploited (mostly
by 2002 Page, 200
Suzaki Shigeri Miy

The problem in a nutshell:
Daemen, Rijmen, NIST were wrong.
Variable-index table lookup *is*
vulnerable to timing attacks.

AES does many lookups such as
`table[k[3]^n[3]]`, taking time
that depends on `k[3]^n[3]`,
leaking `k[3]`.

It is extremely difficult
to avoid this leak
on Pentium, Athlon, etc.
without gigantic slowdowns.

Naive reaction:
"Oh, of course.
A table lookup takes more time
if it misses the cache."

Mentioned by 1996 Kocher,
2000 Kelsey Schneier Wagner Hall.

Exploited (mostly for DES)
by 2002 Page, 2003 Tsunoo Saito
Suzaki Shigeri Miyauchi.

nutshell:
NIST were wrong.
le lookup *is*
ng attacks.

ookups such as
]], taking time
[3]^n[3],

icult

on, etc.
lowdowns.

Naive reaction:
"Oh, of course.
A table lookup takes more time
if it misses the cache."

Mentioned by 1996 Kocher,
2000 Kelsey Schneier Wagner Hall.

Exploited (mostly for DES)
by 2002 Page, 2003 Tsunoo Saito
Suzaki Shigeri Miyauchi.

Naive reaction, co
"But this isn't har
Eliminate cache m
all the AES tables
before the AES en

Problem 1: Elimin
misses is actually
difficult than this,
achieved by any ex
high-speed AES so

Problem 2: Even i
table-lookup time

Naive reaction:
"Oh, of course.
A table lookup takes more time
if it misses the cache."

Mentioned by 1996 Kocher,
2000 Kelsey Schneier Wagner Hall.

Exploited (mostly for DES)
by 2002 Page, 2003 Tsunoo Saito
Suzaki Shigeri Miyauchi.

Naive reaction, continued:
"But this isn't hard to fix.
Eliminate cache misses by loading
all the AES tables into cache
before the AES encryption."

Problem 1: Eliminating cache
misses is actually much more
difficult than this, and is not
achieved by any existing
high-speed AES software.

Problem 2: Even in cache,
table-lookup time is variable.

kes more time
che."

6 Kocher,
eier Wagner Hall.

for DES)
03 Tsunoo Saito
vauchi.

Naive reaction, continued:
"But this isn't hard to fix.
Eliminate cache misses by loading
all the AES tables into cache
before the AES encryption."

Problem 1: Eliminating cache
misses is actually much more
difficult than this, and is not
achieved by any existing
high-speed AES software.

Problem 2: Even in cache,
table-lookup time is variable.

Obstacles to writi
that looks up a ta
in time independe

- Cache is faster t
- L1 cache is faste
- Cache associativ
- Code can be int
- Stores can interf
- Cache-bank thro

Perhaps there are
Most CPU designe
adequately docum

Naive reaction, continued:
"But this isn't hard to fix.
Eliminate cache misses by loading
all the AES tables into cache
before the AES encryption."

Problem 1: Eliminating cache misses is actually much more difficult than this, and is not achieved by any existing high-speed AES software.

Problem 2: Even in cache, table-lookup time is variable.

Obstacles to writing code
that looks up a table entry
in time independent of index:

- Cache is faster than DRAM.
- L1 cache is faster than L2 cache.
- Cache associativity is limited.
- Code can be interrupted.
- Stores can interfere with loads.
- Cache-bank throughput is limited.

Perhaps there are more obstacles.
Most CPU designers fail to
adequately document CPU speed!

ntinued:
d to fix.
isses by loading
into cache
cryption."

ating cache
much more
and is not
kisting
oftware.

n cache,
is variable.

Obstacles to writing code
that looks up a table entry
in time independent of index:

- Cache is faster than DRAM.
- L1 cache is faster than L2 cache.
- Cache associativity is limited.
- Code can be interrupted.
- Stores can interfere with loads.
- Cache-bank throughput is limited.

Perhaps there are more obstacles.
Most CPU designers fail to
adequately document CPU speed!

Cache associativity

AES software uses
tables, input, key,

On (e.g.) Athlon,
overlap modulo 32
two arrays can kn
another array out

Fix: squeeze varia
limited number of
reload tables befor

Obstacles to writing code
that looks up a table entry
in time independent of index:

- Cache is faster than DRAM.
- L1 cache is faster than L2 cache.
- Cache associativity is limited.
- Code can be interrupted.
- Stores can interfere with loads.
- Cache-bank throughput is limited.

Perhaps there are more obstacles.
Most CPU designers fail to
adequately document CPU speed!

Cache associativity is limited

AES software uses several arrays:
tables, input, key, etc.

On (e.g.) Athlon, if array positions
overlap modulo 32768, accessing
two arrays can knock an entry in
another array out of cache.

Fix: squeeze variables into a
limited number of arrays;
reload tables before every AES call.

ng code
ble entry
nt of index:

han DRAM.
er than L2 cache.
ity is limited.
errupted.
fere with loads.
oughput is limited.

more obstacles.
ers fail to
ent CPU speed!

Cache associativity is limited

AES software uses several arrays:
tables, input, key, etc.

On (e.g.) Athlon, if array positions
overlap modulo 32768, accessing
two arrays can knock an entry in
another array out of cache.

Fix: squeeze variables into a
limited number of arrays;
reload tables before every AES call.

Code can be interr

Another process ca
an array entry out

Consider hyperthre
(2005 Osvik Sham
independently 200
But problem exists
even without hype

Fix: put AES into
kernel, disabling in
call takes time; re
code changes; but

Cache associativity is limited

AES software uses several arrays: tables, input, key, etc.

On (e.g.) Athlon, if array positions overlap modulo 32768, accessing two arrays can knock an entry in another array out of cache.

Fix: squeeze variables into a limited number of arrays; reload tables before every AES call.

Code can be interrupted

Another process can knock an array entry out of cache.

Consider hyperthreading attacks (2005 Osvik Shamir Tromer, independently 2005 Percival). But problem exists even without hyperthreading.

Fix: put AES into operating-system kernel, disabling interrupts. Kernel call takes time; requires massive code changes; but should work.

y is limited

several arrays:
etc.

if array positions
2768, accessing
ock an entry in
of cache.

bles into a
arrays;
re every AES call.

Code can be interrupted

Another process can knock
an array entry out of cache.

Consider hyperthreading attacks
(2005 Osvik Shamir Tromer,
independently 2005 Percival).
But problem exists
even without hyperthreading.

Fix: put AES into operating-system
kernel, disabling interrupts. Kernel
call takes time; requires massive
code changes; but should work.

Stores can interfer

On (e.g.) Pentium
load from L1 cach
slightly slower if it
same cache line m
as a recent store.

Timing variation h
even if all loads ar

Fix: compress AES
table positions mo

Code can be interrupted

Another process can knock
an array entry out of cache.

Consider hyperthreading attacks
(2005 Osvik Shamir Tromer,
independently 2005 Percival).
But problem exists
even without hyperthreading.

Fix: put AES into operating-system kernel, disabling interrupts. Kernel call takes time; requires massive code changes; but should work.

Stores can interfere with loads

On (e.g.) Pentium III,
load from L1 cache is
slightly slower if it involves
same cache line modulo 4096
as a recent store.

Timing variation happens
even if all loads are from L1 cache!

Fix: compress AES tables; control table positions modulo 4096.

rupted

an knock
of cache.

eading attacks
ir Tromer,
5 Percival).
s

erthreading.

operating-system
nterrupts. Kernel
quires massive
should work.

Stores can interfere with loads

On (e.g.) Pentium III,
load from L1 cache is
slightly slower if it involves
same cache line modulo 4096
as a recent store.

Timing variation happens
even if all loads are from L1 cache!

Fix: compress AES tables; control
table positions modulo 4096.

Cache-bank throug

On (e.g.) Athlon,
can perform two lo
from L1 cache eve
Exception: Second
waits for a cycle if
are from same cac

Fix: very careful a
to ensure that two
never happen in th
My AES software

Stores can interfere with loads

On (e.g.) Pentium III,
load from L1 cache is
slightly slower if it involves
same cache line modulo 4096
as a recent store.

Timing variation happens
even if all loads are from L1 cache!

Fix: compress AES tables; control
table positions modulo 4096.

Cache-bank throughput is limited

On (e.g.) Athlon,
can perform two loads
from L1 cache every cycle.
Exception: Second load
waits for a cycle if loads
are from same cache "bank."

Fix: very careful asm programming
to ensure that two loads
never happen in the same cycle.
My AES software tries to do this.

re with loads

III,
e is
involves
odulo 4096

happens
e from L1 cache!

S tables; control
odulo 4096.

Cache-bank throughput is limited

On (e.g.) Athlon,
can perform two loads
from L1 cache every cycle.
Exception: Second load
waits for a cycle if loads
are from same cache "bank."

Fix: very careful asm programming
to ensure that two loads
never happen in the same cycle.
My AES software tries to do this.

Do we want to kee

Variable-index tab
are dangerous.

Clearly impossible
today's advertised
with constant-time

I believe it's possib
achieve tolerable s
but it's extremely
and it's fragile: ne
will allow more att

Cache-bank throughput is limited

On (e.g.) Athlon,
can perform two loads
from L1 cache every cycle.
Exception: Second load
waits for a cycle if loads
are from same cache "bank."

Fix: very careful asm programming
to ensure that two loads
never happen in the same cycle.
My AES software tries to do this.

Do we want to keep AES?

Variable-index table lookups
are dangerous.

Clearly impossible to achieve
today's advertised AES speeds
with constant-time software.

I believe it's possible to
achieve tolerable speeds,
but it's extremely difficult,
and it's fragile: new CPUs
will allow more attacks.

ghput is limited

oads

ery cycle.

d load

loads

he "bank."

sm programming

loads

ne same cycle.

tries to do this.

Do we want to keep AES?

Variable-index table lookups are dangerous.

Clearly impossible to achieve today's advertised AES speeds with constant-time software.

I believe it's possible to achieve tolerable speeds, but it's extremely difficult, and it's fragile: new CPUs will allow more attacks.

Why not switch to
that avoids these p

2005 Schneier: "P
any encryption alg
susceptible to timi
so choosing on tha
doesn't make that

But some fast ciph
susceptible to timi
Can build fast ciph
add, constant-dista
Examples: TEA, H

Do we want to keep AES?

Variable-index table lookups are dangerous.

Clearly impossible to achieve today's advertised AES speeds with constant-time software.

I believe it's possible to achieve tolerable speeds, but it's extremely difficult, and it's fragile: new CPUs will allow more attacks.

Why not switch to a cipher that avoids these problems?

2005 Schneier: "Pretty much any encryption algorithm is susceptible to timing attacks, so choosing on that regard doesn't make that much sense."

But some fast ciphers are *not* susceptible to timing attacks! Can build fast cipher from xor, add, constant-distance rotation. Examples: TEA, Helix, Salsa20.