# How cryptographic benchmarking goes wrong

Daniel J. Bernstein

Thanks to NIST 60NANB12D261 for funding this work, and for not reviewing these slides in advance.

---

PRESERVE, ending 2015.06.30, was a European project "Preparing Secure Vehicle-to-X Communication Systems".

Project cost: 5383431 EUR, including 3850000 EUR from the European Commission.



"About PRESERVE": "The mission of PRESERVE is, to *design, implement, and test a secure and scalable V2X Security Subsystem for realistic deployment scenarios*. . . . [Expected Results:] 1. Harmonized V2X Security Architecture. 2. Implementation of V2X Security Subsystem. 3. Cheap and scalable security ASIC for V2X. 4. Testing results VSS under realistic conditions. 5. Research results for deployment challenges."

ptographic benchmarking
ong

. Bernstein

to NIST 60NANB12D261
ing this work, and for not
g these slides in advance.

---

RVE, ending 2015.06.30,
uropean project
ng Secure Vehicle-to-X
nication Systems".

cost: 5383431 EUR,
g 3850000 EUR from
opean Commission.



"About PRESERVE": "The mission of PRESERVE is, to *design, implement, and test a secure and scalable V2X Security Subsystem for realistic deployment scenarios*. . . . [Expected Results:] 1. Harmonized V2X Security Architecture. 2. Implementation of V2X Security Subsystem. 3. Cheap and scalable security ASIC for V2X. 4. Testing results VSS under realistic conditions. 5. Research results for deployment challenges."



Cars alre
Why bui

PRESER
"Securit
Security
"Process
second a
ms can l
hardware
a Pentiu
needs ab
a verifica
cryptogr
likely to

c benchmarking

n

0NANB12D261
ork, and for not
des in advance.

---

ng 2015.06.30,
roject
 Vehicle-to-X
ystems".

3431 EUR,
 EUR from
nmission.



"About PRESERVE": "The mission of PRESERVE is, to *design, implement, and test a secure and scalable V2X Security Subsystem for realistic deployment scenarios*. . . . [Expected Results:] 1. Harmonized V2X Security Architecture. 2. Implementation of V2X Security Subsystem. 3. Cheap and scalable security ASIC for V2X. 4. Testing results VSS under realistic conditions. 5. Research results for deployment challenges."



Cars already incluc
Why build an ASI(

PRESERVE delive
"Security Requirer
Security Architect
"Processing 1,000
second and proces
ms can hardly be
hardware. As disc
a Pentium D 3.4 C
needs about 5 tim
a verification . . . a
cryptographic co-p
likely to be necess

arking

2D261
or not
ance.

5.30,

o-X

.
n

1

“About PRESERVE”: “The mission of PRESERVE is, to *design, implement, and test a secure and scalable V2X Security Subsystem for realistic deployment scenarios*. . . . [Expected Results:] 1. Harmonized V2X Security Architecture. 2. Implementation of V2X Security Subsystem. 3. Cheap and scalable security ASIC for V2X. 4. Testing results VSS under realistic conditions. 5. Research results for deployment challenges.”

2

Cars already include many C
Why build an ASIC?

PRESERVE deliverable 1.1,
“Security Requirements of V
Security Architecture”, 2011
“Processing 1,000 packets p
second and processing each
ms can hardly be met by cu
hardware. As discussed in [3
a Pentium D 3.4 GHz proce
needs about 5 times as long
a verification . . . a dedicate
cryptographic co-processor i
likely to be necessary.”

"About PRESERVE": "The mission of PRESERVE is, to *design, implement, and test a secure and scalable V2X Security Subsystem for realistic deployment scenarios*. . . . [Expected Results:] 1. Harmonized V2X Security Architecture. 2. Implementation of V2X Security Subsystem. 3. Cheap and scalable security ASIC for V2X. 4. Testing results VSS under realistic conditions. 5. Research results for deployment challenges."



Cars already include many CPUs. Why build an ASIC?

PRESERVE deliverable 1.1, "Security Requirements of Vehicle Security Architecture", 2011: "Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary."

PRESERVE": "The
of PRESERVE is,
*n, implement, and*
*ecure and scalable*
*curity Subsystem for*
*deployment scenarios*.
ected Results:] 1.
ized V2X Security
ture. 2. Implementation
Security Subsystem. 3.
nd scalable security ASIC
. 4. Testing results VSS
alistic conditions. 5.
n results for deployment
es."



Cars already include many CPUs. Why build an ASIC?

PRESERVE deliverable 1.1, "Security Requirements of Vehicle Security Architecture", 2011: "Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary."



PRESER
"Deploy
V4", 20
ECC sig
second i
factor fo
environn
4mm$\times$4
technolo
space fo
90nm wi
cores an
more."
max 100

VE": "The
RVE is,
*ent, and*
*scalable*
*system for*
*nt scenarios.*
ults:] 1.
Security
mplementation
ubsystem. 3.
e security ASIC
ng results VSS
ditions. 5.
or deployment



Cars already include many CPUs.
Why build an ASIC?

PRESERVE deliverable 1.1, "Security Requirements of Vehicle Security Architecture", 2011: "Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary."



PRESERVE delive
"Deployment Issue
V4", 2016: "the n
ECC signature ver
second is the key p
factor for ASICs ir
environment . . . [(
4mm×4mm chip]
technology may or
space for one ECC
90nm will allow fo
cores and 55nm w
more." For 180nm
max 100MHz, 100

/
os.
ation
3.
ASIC
VSS
.
ent

2

Cars already include many CPUs. Why build an ASIC?

PRESERVE deliverable 1.1, "Security Requirements of Vehicle Security Architecture", 2011: "Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary."

3

PRESERVE deliverable 5.4, "Deployment Issues Report V4", 2016: "the number of ECC signature verifications second is the key performan factor for ASICs in a C2C environment . . . [On a 4mm$\times$4mm chip] the 180nn technology may only yield e space for one ECC core, whe 90nm will allow for up to te cores and 55nm will allow fo more." For 180nm core says max 100MHz, 100 verif/seco

Cars already include many CPUs.
Why build an ASIC?

PRESERVE deliverable 1.1, "Security Requirements of Vehicle Security Architecture", 2011: "Processing 1,000 packets per second and processing each in 1 ms can hardly be met by current hardware. As discussed in [32], a Pentium D 3.4 GHz processor needs about 5 times as long for a verification . . . a dedicated cryptographic co-processor is likely to be necessary."



PRESERVE deliverable 5.4, "Deployment Issues Report V4", 2016: "the number of ECC signature verifications per second is the key performance factor for ASICs in a C2C environment . . . [On a 4mm×4mm chip] the 180nm technology may only yield enough space for one ECC core, whereas 90nm will allow for up to ten ECC cores and 55nm will allow for even more." For 180nm core says max 100MHz, 100 verif/second.

eady include many CPUs.
ild an ASIC?

RVE deliverable 1.1,
y Requirements of Vehicle
Architecture”, 2011:
sing 1,000 packets per
and processing each in 1
hardly be met by current
e. As discussed in [32],
m D 3.4 GHz processor
bout 5 times as long for
ation . . . a dedicated
aphic co-processor is
be necessary.”



PRESERVE deliverable 5.4, “Deployment Issues Report V4”, 2016: “the number of ECC signature verifications per second is the key performance factor for ASICs in a C2C environment . . . [On a 4mm×4mm chip] the 180nm technology may only yield enough space for one ECC core, whereas 90nm will allow for up to ten ECC cores and 55nm will allow for even more.” For 180nm core says max 100MHz, 100 verif/second.

Compare
IAIK NIS
858 scal
in 11162
at 180nr
technolo
standard
9.3744 $\mu$
conditio
core volt

Signatur
somewha
Still clos
than the

de many CPUs.
C?

rable 1.1,
ments of Vehicle
ure", 2011:
packets per
sing each in 1
met by current
ussed in [32],
GHz processor
es as long for
a dedicated
processor is
ary."



PRESERVE deliverable 5.4, "Deployment Issues Report V4", 2016: "the number of ECC signature verifications per second is the key performance factor for ASICs in a C2C environment . . . [On a 4mm×4mm chip] the 180nm technology may only yield enough space for one ECC core, whereas 90nm will allow for up to ten ECC cores and 55nm will allow for even more." For 180nm core says max 100MHz, 100 verif/second.

Compare to, e.g.,
IAIK NIST P-256
858 scalarmult/se
in 111620 GE at 1
at 180nm ("UMC
technology using
standard cell libra
9.3744 $\mu m^2$/GE;
conditions (tempe
core voltage 1.62V

Signature verificat
somewhat slower t
Still close to 100×
than the PRESER

CPUs.

Vehicle
l:
per
in 1
rrent
32],
ssor
for
d
s



PRESERVE deliverable 5.4, "Deployment Issues Report V4", 2016: "the number of ECC signature verifications per second is the key performance factor for ASICs in a C2C environment . . . [On a 4mm×4mm chip] the 180nm technology may only yield enough space for one ECC core, whereas 90nm will allow for up to ten ECC cores and 55nm will allow for even more." For 180nm core says max 100MHz, 100 verif/second.



Compare to, e.g.,
IAIK NIST P-256 ECC Mod
858 scalarmult/second
in 111620 GE at 192 MHz
at 180nm ("UMC L180GII
technology using Faraday f1
standard cell library (FSA0A
9.3744 $\mu$m$^2$/GE; worst case
conditions (temperature 125
core voltage 1.62V)").

Signature verification will be
somewhat slower than scalar
Still close to 100× more effi
than the PRESERVE estima

PRESERVE deliverable 5.4, "Deployment Issues Report V4", 2016: "the number of ECC signature verifications per second is the key performance factor for ASICs in a C2C environment . . . [On a 4mm×4mm chip] the 180nm technology may only yield enough space for one ECC core, whereas 90nm will allow for up to ten ECC cores and 55nm will allow for even more." For 180nm core says max 100MHz, 100 verif/second.



Compare to, e.g., IAIK NIST P-256 ECC Module: 858 scalarmult/second in 111620 GE at 192 MHz at 180nm ("UMC L180GII technology using Faraday f180 standard cell library (FSA0A_C), 9.3744 $\mu m^2$/GE; worst case conditions (temperature 125°C, core voltage 1.62V)").

Signature verification will be somewhat slower than scalarmult. Still close to 100× more efficient than the PRESERVE estimates.

RVE deliverable 5.4,
ment Issues Report
16: "the number of
nature verifications per
s the key performance
r ASICs in a C2C
nent . . . [On a
mm chip] the 180nm
gy may only yield enough
r one ECC core, whereas
ill allow for up to ten ECC
d 55nm will allow for even
For 180nm core says
MHz, 100 verif/second.



Compare to, e.g.,
IAIK NIST P-256 ECC Module:
858 scalarmult/second
in 111620 GE at 192 MHz
at 180nm ("UMC L180GII
technology using Faraday f180
standard cell library (FSA0A_C),
9.3744 $\mu$m$^2$/GE; worst case
conditions (temperature 125°C,
core voltage 1.62V)").

Signature verification will be
somewhat slower than scalarmult.
Still close to 100× more efficient
than the PRESERVE estimates.

Let's go
core arg

Central
in [32],
processo
(i.e., 17
for signa

[32] is "
Z., 'Ana
overhead
Third Jo
Mobile N
(WMNC

rable 5.4,
es Report
umber of
ifications per
performance
n a C2C
On a
the 180nm
nly yield enough
C core, whereas
r up to ten ECC
ill allow for even
n core says
) verif/second.



Compare to, e.g.,
IAIK NIST P-256 ECC Module:
858 scalarmult/second
in 111620 GE at 192 MHz
at 180nm ("UMC L180GII
technology using Faraday f180
standard cell library (FSA0A_C),
9.3744 $\mu m^2$/GE; worst case
conditions (temperature 125°C,
core voltage 1.62V)").

Signature verification will be
somewhat slower than scalarmult.
Still close to 100× more efficient
than the PRESERVE estimates.

Let's go back to P
core argument for

Central claim: "As
in [32], a Pentium
processor needs ab
(i.e., 17 million CP
for signature verifi

[32] is "Petit, J., M
Z., 'Analysis of au
overhead in vehicu
Third Joint IFIP W
Mobile Networking
(WMNC), 2010."

per
ce

n
nough
ereas
n ECC
or even
s
ond.

Compare to, e.g.,
IAIK NIST P-256 ECC Module:
858 scalarmult/second
in 111620 GE at 192 MHz
at 180nm (“UMC L180GII
technology using Faraday f180
standard cell library (FSA0A_C),
9.3744 $\mu$m$^2$/GE; worst case
conditions (temperature 125°C,
core voltage 1.62V)”).

Signature verification will be
somewhat slower than scalarmult.
Still close to 100× more efficient
than the PRESERVE estimates.



Let’s go back to PRESERVE
core argument for an ASIC.

Central claim: “As discussed
in [32], a Pentium D 3.4 GH
processor needs about” 5ms
(i.e., 17 million CPU cycles)
for signature verification.

[32] is “Petit, J., Mammeri,
Z., ‘Analysis of authenticatio
overhead in vehicular networ
Third Joint IFIP Wireless an
Mobile Networking Conferen
(WMNC), 2010.”

Compare to, e.g.,
IAIK NIST P-256 ECC Module:
858 scalarmult/second
in 111620 GE at 192 MHz
at 180nm ("UMC L180GII
technology using Faraday f180
standard cell library (FSA0A_C),
9.3744 $\mu m^2$/GE; worst case
conditions (temperature 125°C,
core voltage 1.62V)").

Signature verification will be
somewhat slower than scalarmult.
Still close to 100× more efficient
than the PRESERVE estimates.



Let's go back to PRESERVE's
core argument for an ASIC.

Central claim: "As discussed
in [32], a Pentium D 3.4 GHz
processor needs about" 5ms
(i.e., 17 million CPU cycles)
for signature verification.

[32] is "Petit, J., Mammeri,
Z., 'Analysis of authentication
overhead in vehicular networks',
Third Joint IFIP Wireless and
Mobile Networking Conference
(WMNC), 2010."

e to, e.g.,
ST P-256 ECC Module:
armult/second
20 GE at 192 MHz
m ("UMC L180GII
gy using Faraday f180
l cell library (FSA0A_C),
$\mu$m$^2$/GE; worst case
ns (temperature 125°C,
tage 1.62V)").

re verification will be
at slower than scalarmult.
se to 100× more efficient
e PRESERVE estimates.



Let's go back to PRESERVE's core argument for an ASIC.

Central claim: "As discussed in [32], a Pentium D 3.4 GHz processor needs about" 5ms (i.e., 17 million CPU cycles) for signature verification.

[32] is "Petit, J., Mammeri, Z., 'Analysis of authentication overhead in vehicular networks', Third Joint IFIP Wireless and Mobile Networking Conference (WMNC), 2010."



[32] says
to the h
economi
from veh
governm
compani
have ma
vehicular
[1]. On
collisions
and 790
United S
economi
[2]. . . .
costing

ECC Module:
cond
.92 MHz
L180GII
Faraday f180
ry (FSA0A_C),
worst case
rature 125°C,
V)").

ion will be
than scalarmult.
x more efficient
VE estimates.



Let's go back to PRESERVE's core argument for an ASIC.

Central claim: "As discussed in [32], a Pentium D 3.4 GHz processor needs about" 5ms (i.e., 17 million CPU cycles) for signature verification.

[32] is "Petit, J., Mammeri, Z., 'Analysis of authentication overhead in vehicular networks', Third Joint IFIP Wireless and Mobile Networking Conference (WMNC), 2010."

[32] says "1. Intro
to the huge life los
economic impacts
from vehicular coll
governments, auto
companies, and in
have made the red
vehicular fatalities
[1]. On average, v
collisions cause 10
and 7900 injuries
United States, lea
economic impact
[2]. . . . [Similar st
costing €160 billio

ule:

80

_C),

°C,

e

rmult.

cient

tes.

5

Let's go back to PRESERVE's core argument for an ASIC.

Central claim: "As discussed in [32], a Pentium D 3.4 GHz processor needs about" 5ms (i.e., 17 million CPU cycles) for signature verification.

[32] is "Petit, J., Mammeri, Z., 'Analysis of authentication overhead in vehicular networks', Third Joint IFIP Wireless and Mobile Networking Conference (WMNC), 2010."

6

[32] says "1. Introduction. I
to the huge life losses and th
economic impacts resulting
from vehicular collisions, ma
governments, automotive
companies, and industry con
have made the reduction of
vehicular fatalities a top prio
[1]. On average, vehicular
collisions cause 102 deaths
and 7900 injuries daily in th
United States, leaving an
economic impact of $230 bil
[2]. . . . [Similar story for EU
costing €160 billion annually

Let's go back to PRESERVE's core argument for an ASIC.

Central claim: "As discussed in [32], a Pentium D 3.4 GHz processor needs about" 5ms (i.e., 17 million CPU cycles) for signature verification.

[32] is "Petit, J., Mammeri, Z., 'Analysis of authentication overhead in vehicular networks', Third Joint IFIP Wireless and Mobile Networking Conference (WMNC), 2010."



[32] says "1. Introduction. Due to the huge life losses and the economic impacts resulting from vehicular collisions, many governments, automotive companies, and industry consortia have made the reduction of vehicular fatalities a top priority [1]. On average, vehicular collisions cause 102 deaths and 7900 injuries daily in the United States, leaving an economic impact of $230 billion [2]. . . . [Similar story for EU:] costing €160 billion annually [3]."

back to PRESERVE's
ument for an ASIC.

claim: "As discussed
a Pentium D 3.4 GHz
r needs about" 5ms
million CPU cycles)
ture verification.

Petit, J., Mammeri,
lysis of authentication
d in vehicular networks',
oint IFIP Wireless and
Networking Conference
C), 2010."



[32] says "1. Introduction. Due to the huge life losses and the economic impacts resulting from vehicular collisions, many governments, automotive companies, and industry consortia have made the reduction of vehicular fatalities a top priority [1]. On average, vehicular collisions cause 102 deaths and 7900 injuries daily in the United States, leaving an economic impact of $230 billion [2]. . . . [Similar story for EU:] costing €160 billion annually [3]."

Vehicles
informat
of IEEE1
support
Signatur
[8] over
P-224 ar
paper, w
and com
the auth
provided
II. Signa
verificati
D 3.4Gh

PRESERVE's
an ASIC.

s discussed
D 3.4 GHz
bout" 5ms
PU cycles)
cation.

Mammeri,
thentication
ular networks',
Wireless and
g Conference



[32] says "1. Introduction. Due to the huge life losses and the economic impacts resulting from vehicular collisions, many governments, automotive companies, and industry consortia have made the reduction of vehicular fatalities a top priority [1]. On average, vehicular collisions cause 102 deaths and 7900 injuries daily in the United States, leaving an economic impact of $230 billion [2]. . . . [Similar story for EU:] costing €160 billion annually [3]."

Vehicles will comn
information. "All
of IEEE1609.2 sta
support the Ellipti
Signature Algorith
[8] over the two N
P-224 and P-256.
paper, we assess t
and communicatio
the authentication
provided by ECDS
II. Signature gener
verification times
D 3.4Ghz workstat

E's

d

Hz

on

rks',

nd

nce

6

[32] says "1. Introduction. Due to the huge life losses and the economic impacts resulting from vehicular collisions, many governments, automotive companies, and industry consortia have made the reduction of vehicular fatalities a top priority [1]. On average, vehicular collisions cause 102 deaths and 7900 injuries daily in the United States, leaving an economic impact of \$230 billion [2]. . . . [Similar story for EU:] costing €160 billion annually [3]."

7

Vehicles will communicate s
information. "All implement
of IEEE1609.2 standard [7] s
support the Elliptic Curve D
Signature Algorithm (ECDS
[8] over the two NIST curve
P-224 and P-256. . . . In thi
paper, we assess the process
and communication overhea
the authentication mechanis
provided by ECDSA. . . . Ta
II. Signature generation and
verification times on a Penti
D 3.4Ghz workstation [10]"

[32] says "1. Introduction. Due to the huge life losses and the economic impacts resulting from vehicular collisions, many governments, automotive companies, and industry consortia have made the reduction of vehicular fatalities a top priority [1]. On average, vehicular collisions cause 102 deaths and 7900 injuries daily in the United States, leaving an economic impact of $230 billion [2]. . . . [Similar story for EU:] costing €160 billion annually [3]."



Vehicles will communicate safety information. "All implementations of IEEE1609.2 standard [7] shall support the Elliptic Curve Digital Signature Algorithm (ECDSA) [8] over the two NIST curves P-224 and P-256. . . . In this paper, we assess the processing and communication overhead of the authentication mechanism provided by ECDSA. . . . Table II. Signature generation and verification times on a Pentium D 3.4Ghz workstation [10]"

s “1. Introduction. Due
uge life losses and the
c impacts resulting
hicular collisions, many
ents, automotive
es, and industry consortia
de the reduction of
r fatalities a top priority
average, vehicular
s cause 102 deaths
0 injuries daily in the
States, leaving an
c impact of $230 billion
[Similar story for EU:]
€160 billion annually [3].”



Vehicles will communicate safety information. “All implementations of IEEE1609.2 standard [7] shall support the Elliptic Curve Digital Signature Algorithm (ECDSA) [8] over the two NIST curves P-224 and P-256. . . . In this paper, we assess the processing and communication overhead of the authentication mechanism provided by ECDSA. . . . Table II. Signature generation and verification times on a Pentium D 3.4Ghz workstation [10]”

[10] (in
J., ‘Anal
Authent
VANETs
Conferen
Mobility
Cairo, D

[10] says
impleme
and follo
For NIS
“Pentiur
2.50ms/
4.97ms/

oduction. Due
sses and the
resulting
lisions, many
omotive
dustry consortia
duction of
a top priority
ehicular
2 deaths
daily in the
ving an
of $230 billion
ory for EU:]
on annually [3]."



Vehicles will communicate safety information. "All implementations of IEEE1609.2 standard [7] shall support the Elliptic Curve Digital Signature Algorithm (ECDSA) [8] over the two NIST curves P-224 and P-256. . . . In this paper, we assess the processing and communication overhead of the authentication mechanism provided by ECDSA. . . . Table II. Signature generation and verification times on a Pentium D 3.4Ghz workstation [10]"

[10] (in [32]) is "P
J., 'Analysis of EC
Authentication Pr
VANETs', 3rd IFIP
Conference on Ne
Mobility and Secu
Cairo, December 2

[10] says "ECDSA
implemented using
and following the
For NIST P-224/P
"Pentium D 3.4GH
2.50ms/3.33ms to
4.97ms/6.63ms to

Due
he
ny
nsortia
ority
e
llion
U:]
y [3]."

Vehicles will communicate safety information. "All implementations of IEEE1609.2 standard [7] shall support the Elliptic Curve Digital Signature Algorithm (ECDSA) [8] over the two NIST curves P-224 and P-256. . . . In this paper, we assess the processing and communication overhead of the authentication mechanism provided by ECDSA. . . . Table II. Signature generation and verification times on a Pentium D 3.4Ghz workstation [10]"



[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing i
VANETs', 3rd IFIP Internati
Conference on New Technol
Mobility and Security (NTM
Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL
and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz worksta
2.50ms/3.33ms to sign,
4.97ms/6.63ms to verify.

Vehicles will communicate safety information. "All implementations of IEEE1609.2 standard [7] shall support the Elliptic Curve Digital Signature Algorithm (ECDSA) [8] over the two NIST curves P-224 and P-256. . . . In this paper, we assess the processing and communication overhead of the authentication mechanism provided by ECDSA. . . . Table II. Signature generation and verification times on a Pentium D 3.4Ghz workstation [10]"



[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing in VANETs', 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz workstation": 2.50ms/3.33ms to sign, 4.97ms/6.63ms to verify.

will communicate safety
tion. "All implementations
1609.2 standard [7] shall
the Elliptic Curve Digital
re Algorithm (ECDSA)
the two NIST curves
nd P-256. . . . In this
ve assess the processing
nmunication overhead of
entication mechanism
by ECDSA. . . . Table
ture generation and
ion times on a Pentium
z workstation [10]"



[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing in VANETs', 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz workstation":
2.50ms/3.33ms to sign,
4.97ms/6.63ms to verify.

Compare
speeds r
of 14nm
("2015 I
https:/

0.015ms
0.049ms

nunicate safety
implementations
ndard [7] shall
c Curve Digital
m (ECDSA)
IST curves
. . . In this
he processing
on overhead of
mechanism
A. . . . Table
ration and
on a Pentium
tion [10]"



[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing in VANETs', 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz workstation":
2.50ms/3.33ms to sign,
4.97ms/6.63ms to verify.

Compare to, e.g.,
speeds reported fo
of 14nm 3.31GHz
("2015 Intel Core
https://bench.c

0.015ms to sign (4
0.049ms to verify

afety
tations
shall
igital
A)
s
s
sing
d of
sm
ble

um

[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing in VANETs', 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz workstation":
2.50ms/3.33ms to sign,
4.97ms/6.63ms to verify.



Compare to, e.g., Ed25519
speeds reported for single co
of 14nm 3.31GHz Skylake
("2015 Intel Core i5-6600")
https://bench.cr.yp.to:

0.015ms to sign (49840 cycl
0.049ms to verify (163206 c

[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing in VANETs', 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz workstation":
2.50ms/3.33ms to sign,
4.97ms/6.63ms to verify.



Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on `https://bench.cr.yp.to`:

0.015ms to sign (49840 cycles),
0.049ms to verify (163206 cycles).

[10] (in [32]) is "Petit J., 'Analysis of ECDSA Authentication Processing in VANETs', 3rd IFIP International Conference on New Technologies, Mobility and Security (NTMS), Cairo, December 2009."

[10] says "ECDSA was implemented using MIRACL and following the Fig.1."
For NIST P-224/P-256 on "Pentium D 3.4GHz workstation":
2.50ms/3.33ms to sign,
4.97ms/6.63ms to verify.



Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles),
0.049ms to verify (163206 cycles).

This chip didn't exist in 2009.
Compare instead to single core of 65nm 2.4GHz Core 2 ("2007 Intel Core 2 Quad Q6600").

0.065ms to sign (156843 cycles),
0.232ms to verify (557082 cycles).

[32]) is “Petit
lysis of ECDSA
ication Processing in
s', 3rd IFIP International
nce on New Technologies,
and Security (NTMS),
ecember 2009.”

s “ECDSA was
nted using MIRACL
owing the Fig.1.”
T P-224/P-256 on
n D 3.4GHz workstation”:
3.33ms to sign,
6.63ms to verify.



Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake (“2015 Intel Core i5-6600”) on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles),
0.049ms to verify (163206 cycles).

This chip didn't exist in 2009. Compare instead to single core of 65nm 2.4GHz Core 2 (“2007 Intel Core 2 Quad Q6600”).

0.065ms to sign (156843 cycles),
0.232ms to verify (557082 cycles).

2012 Be
on 720M
0.9ms to

ARM Co
1000MH
in iPad 1
1000MH
in Samsu
1000MH
Motorola
800MHz
Amazon
Today: i
Cortex-A

Petit
CDSA
ocessing in
P International
w Technologies,
rity (NTMS),
2009."

was
g MIRACL
Fig.1."
P-256 on
Hz workstation":
sign,
verify.



Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles), 0.049ms to verify (163206 cycles).

This chip didn't exist in 2009. Compare instead to single core of 65nm 2.4GHz Core 2 ("2007 Intel Core 2 Quad Q6600").

0.065ms to sign (156843 cycles), 0.232ms to verify (557082 cycles).

2012 Bernstein–Sc
on 720MHz ARM
0.9ms to verify (65

ARM Cortex-A8 c
1000MHz Apple A
in iPad 1, iPhone
1000MHz Samsun
in Samsung Galax
1000MHz TI OMA
Motorola Droid X
800MHz Freescale
Amazon Kindle 4
Today: in CPUs c
Cortex-A7 is even

n

ional

ogies,

IS),

ation":



Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles),
0.049ms to verify (163206 cycles).

This chip didn't exist in 2009. Compare instead to single core of 65nm 2.4GHz Core 2 ("2007 Intel Core 2 Quad Q6600").

0.065ms to sign (156843 cycles),
0.232ms to verify (557082 cycles).



2012 Bernstein–Schwabe on 720MHz ARM Cortex-A8
0.9ms to verify (650102 cycl

ARM Cortex-A8 cores were
1000MHz Apple A4
in iPad 1, iPhone 4 (2010);
1000MHz Samsung Exynos
in Samsung Galaxy S (2010
1000MHz TI OMAP3630 in
Motorola Droid X (2010);
800MHz Freescale i.MX50 i
Amazon Kindle 4 (2011); . .
Today: in CPUs costing ≈2
Cortex-A7 is even more pop

Compare to, e.g., Ed25519 speeds reported for single core of 14nm 3.31GHz Skylake ("2015 Intel Core i5-6600") on https://bench.cr.yp.to:

0.015ms to sign (49840 cycles), 0.049ms to verify (163206 cycles).

This chip didn't exist in 2009. Compare instead to single core of 65nm 2.4GHz Core 2 ("2007 Intel Core 2 Quad Q6600").

0.065ms to sign (156843 cycles), 0.232ms to verify (557082 cycles).



2012 Bernstein–Schwabe on 720MHz ARM Cortex-A8: 0.9ms to verify (650102 cycles).

ARM Cortex-A8 cores were in 1000MHz Apple A4 in iPad 1, iPhone 4 (2010); 1000MHz Samsung Exynos 3110 in Samsung Galaxy S (2010); 1000MHz TI OMAP3630 in Motorola Droid X (2010); 800MHz Freescale i.MX50 in Amazon Kindle 4 (2011); . . .
Today: in CPUs costing ≈2 EUR.
Cortex-A7 is even more popular.

e to, e.g., Ed25519
eported for single core
3.31GHz Skylake
ntel Core i5-6600") on
//bench.cr.yp.to:

to sign (49840 cycles),
to verify (163206 cycles).

p didn't exist in 2009.
e instead to single core
2.4GHz Core 2 ("2007
re 2 Quad Q6600").

to sign (156843 cycles),
to verify (557082 cycles).



2012 Bernstein–Schwabe
on 720MHz ARM Cortex-A8:
0.9ms to verify (650102 cycles).

ARM Cortex-A8 cores were in
1000MHz Apple A4
in iPad 1, iPhone 4 (2010);
1000MHz Samsung Exynos 3110
in Samsung Galaxy S (2010);
1000MHz TI OMAP3630 in
Motorola Droid X (2010);
800MHz Freescale i.MX50 in
Amazon Kindle 4 (2011); . . .
Today: in CPUs costing ≈2 EUR.
Cortex-A7 is even more popular.

180nm 3
("2001 I

0.46ms (
for Curv
using flo
Integer r

Nobody
adapting
Would b

3.4GHz
same ba
more ins
Ed2551
on one c

Ed25519
r single core
Skylake
i5-6600") on
cr.yp.to:

49840 cycles),
(163206 cycles).

xist in 2009.
o single core
Core 2 ("2007
Q6600").

156843 cycles),
(557082 cycles).



2012 Bernstein–Schwabe
on 720MHz ARM Cortex-A8:
0.9ms to verify (650102 cycles).

ARM Cortex-A8 cores were in
1000MHz Apple A4
in iPad 1, iPhone 4 (2010);
1000MHz Samsung Exynos 3110
in Samsung Galaxy S (2010);
1000MHz TI OMAP3630 in
Motorola Droid X (2010);
800MHz Freescale i.MX50 in
Amazon Kindle 4 (2011); . . .
Today: in CPUs costing ≈2 EUR.
Cortex-A7 is even more popular.



180nm 32-bit 2GH
("2001 Intel Penti

0.46ms (0.9 millio
for Curve25519 sca
using floating-poin
Integer multiplier i

Nobody has ever b
adapting this to si
Would be ≈0.6ms

3.4GHz Pentium D
same basic microa
more instructions,
Ed25519 would be
on one core than I

ore

on

es),

ycles).

9.

ore

007

cles),

ycles).



2012 Bernstein–Schwabe
on 720MHz ARM Cortex-A8:
0.9ms to verify (650102 cycles).

ARM Cortex-A8 cores were in
1000MHz Apple A4
in iPad 1, iPhone 4 (2010);
1000MHz Samsung Exynos 3110
in Samsung Galaxy S (2010);
1000MHz TI OMAP3630 in
Motorola Droid X (2010);
800MHz Freescale i.MX50 in
Amazon Kindle 4 (2011); . . .
Today: in CPUs costing ≈2 EUR.
Cortex-A7 is even more popular.



180nm 32-bit 2GHz Willame
("2001 Intel Pentium 4"):

0.46ms (0.9 million cycles)
for Curve25519 scalarmult
using floating-point multipli
Integer multiplier is much sl

Nobody has ever bothered
adapting this to signatures.
Would be ≈0.6ms for verify.

3.4GHz Pentium D (dual co
same basic microarchitectur
more instructions, faster clo
Ed25519 would be >10× fa
on one core than Petit's sof

2012 Bernstein–Schwabe
on 720MHz ARM Cortex-A8:
0.9ms to verify (650102 cycles).

ARM Cortex-A8 cores were in
1000MHz Apple A4
in iPad 1, iPhone 4 (2010);
1000MHz Samsung Exynos 3110
in Samsung Galaxy S (2010);
1000MHz TI OMAP3630 in
Motorola Droid X (2010);
800MHz Freescale i.MX50 in
Amazon Kindle 4 (2011); . . .
Today: in CPUs costing ≈2 EUR.
Cortex-A7 is even more popular.



180nm 32-bit 2GHz Willamette
("2001 Intel Pentium 4"):

0.46ms (0.9 million cycles)
for Curve25519 scalarmult
using floating-point multiplier.
Integer multiplier is much slower!

Nobody has ever bothered
adapting this to signatures.
Would be ≈0.6ms for verify.

3.4GHz Pentium D (dual core):
same basic microarchitecture,
more instructions, faster clock.
Ed25519 would be $>10\times$ faster
on one core than Petit's software.

rnstein–Schwabe
MHz ARM Cortex-A8:
o verify (650102 cycles).

ortex-A8 cores were in
Hz Apple A4
1, iPhone 4 (2010);
Hz Samsung Exynos 3110
ung Galaxy S (2010);
Hz TI OMAP3630 in
a Droid X (2010);
z Freescale i.MX50 in
Kindle 4 (2011); . . .
in CPUs costing ≈2 EUR.
A7 is even more popular.



180nm 32-bit 2GHz Willamette
("2001 Intel Pentium 4"):

0.46ms (0.9 million cycles)
for Curve25519 scalarmult
using floating-point multiplier.
Integer multiplier is much slower!

Nobody has ever bothered
adapting this to signatures.
Would be ≈0.6ms for verify.

3.4GHz Pentium D (dual core):
same basic microarchitecture,
more instructions, faster clock.
Ed25519 would be >10× faster
on one core than Petit's software.

Bad ECD
certainly
• can't u
• can't u
• need a
etc. Typ

2000 Br
Menezes
4.0ms/6
cycles) f
inside N

2001 Be
0.7 milli
for NIST

chwabe
Cortex-A8:
50102 cycles).

ores were in
A4
4 (2010);
g Exynos 3110
y S (2010);
AP3630 in
(2010);
i.MX50 in
(2011); . . .
osting ≈2 EUR.
more popular.



180nm 32-bit 2GHz Willamette ("2001 Intel Pentium 4"):

0.46ms (0.9 million cycles) for Curve25519 scalarmult using floating-point multiplier. Integer multiplier is much slower!

Nobody has ever bothered adapting this to signatures. Would be ≈0.6ms for verify.

3.4GHz Pentium D (dual core): same basic microarchitecture, more instructions, faster clock. Ed25519 would be $>10\times$ faster on one core than Petit's software.

Bad ECDSA-NIST
certainly has some
• can't use fastest
• can't use fastest
• need an annoyin
etc. Typical estim

2000 Brown–Hank
Menezes on 400M
4.0ms/6.4ms (1.6/
cycles) for double
inside NIST P-224

2001 Bernstein, ≈
0.7 million cycles
for NIST P-224 sc

3:
les).
in

3110
);

n
.
EUR.
ular.



180nm 32-bit 2GHz Willamette (“2001 Intel Pentium 4”):

0.46ms (0.9 million cycles) for Curve25519 scalarmult using floating-point multiplier. Integer multiplier is much slower!

Nobody has ever bothered adapting this to signatures. Would be ≈0.6ms for verify.

3.4GHz Pentium D (dual core): same basic microarchitecture, more instructions, faster clock. Ed25519 would be >10× faster on one core than Petit's software.

Bad ECDSA-NIST-P-256 de
certainly has some impact:

- can't use fastest mulmods
- can't use fastest curve for
- need an annoying inversio

etc. Typical estimate: 2× sl

2000 Brown–Hankerson–Lóp
Menezes on 400MHz Pentiu
4.0ms/6.4ms (1.6/2.6 millio
cycles) for double scalarmult
inside NIST P-224/P-256 ve

2001 Bernstein, ≈1.6× faste
0.7 million cycles on Pentiu
for NIST P-224 scalarmult.

180nm 32-bit 2GHz Willamette ("2001 Intel Pentium 4"):

0.46ms (0.9 million cycles) for Curve25519 scalarmult using floating-point multiplier. Integer multiplier is much slower!

Nobody has ever bothered adapting this to signatures. Would be $\approx$0.6ms for verify.

3.4GHz Pentium D (dual core): same basic microarchitecture, more instructions, faster clock. Ed25519 would be $>10\times$ faster on one core than Petit's software.



Bad ECDSA-NIST-P-256 design certainly has some impact:

- can't use fastest mulmods;
- can't use fastest curve formulas;
- need an annoying inversion;

etc. Typical estimate: $2\times$ slower.

2000 Brown–Hankerson–López–Menezes on 400MHz Pentium II: 4.0ms/6.4ms (1.6/2.6 million cycles) for double scalarmult inside NIST P-224/P-256 verif.

2001 Bernstein, $\approx 1.6\times$ faster: 0.7 million cycles on Pentium II for NIST P-224 scalarmult.

32-bit 2GHz Willamette
ntel Pentium 4"):

(0.9 million cycles)
e25519 scalarmult
ating-point multiplier.
multiplier is much slower!

has ever bothered
this to signatures.
be ≈0.6ms for verify.

Pentium D (dual core):
sic microarchitecture,
structions, faster clock.
would be >10× faster
core than Petit's software.



Bad ECDSA-NIST-P-256 design certainly has some impact:

- can't use fastest mulmods;
- can't use fastest curve formulas;
- need an annoying inversion;

etc. Typical estimate: 2× slower.

2000 Brown–Hankerson–López–Menezes on 400MHz Pentium II: 4.0ms/6.4ms (1.6/2.6 million cycles) for double scalarmult inside NIST P-224/P-256 verif.

2001 Bernstein, ≈1.6× faster: 0.7 million cycles on Pentium II for NIST P-224 scalarmult.

2000 Br
Menezes
cycles o

e.g., P-2
1.2 milli
2.7 milli

2001 Be
0.7 milli
0.8 milli
0.9 milli
using co

OpenSS
2.0 milli

lz Willamette
um 4"):

n cycles)
alarmult
nt multiplier.
is much slower!

oothered
gnatures.
for verify.

D (dual core):
rchitecture,
faster clock.
e >10× faster
Petit's software.



Bad ECDSA-NIST-P-256 design
certainly has some impact:

- can't use fastest mulmods;
- can't use fastest curve formulas;
- need an annoying inversion;

etc. Typical estimate: 2× slower.

2000 Brown–Hankerson–López–
Menezes on 400MHz Pentium II:
4.0ms/6.4ms (1.6/2.6 million
cycles) for double scalarmult
inside NIST P-224/P-256 verif.

2001 Bernstein, ≈1.6× faster:
0.7 million cycles on Pentium II
for NIST P-224 scalarmult.

2000 Brown–Hank
Menezes software
cycles on P4 than

e.g., P-224 scalarr
1.2 million cycles
2.7 million cycles

2001 Bernstein P-
0.7 million cycles
0.8 million cycles
0.9 million cycles
using compressed

OpenSSL 1.0.1, P-
2.0 million cycles

ette

er.

ower!

re):

e,

ck.

ster

tware.



Bad ECDSA-NIST-P-256 design
certainly has some impact:

- can't use fastest mulmods;
- can't use fastest curve formulas;
- need an annoying inversion;

etc. Typical estimate: $2\times$ slower.

2000 Brown–Hankerson–López–Menezes on 400MHz Pentium II: 4.0ms/6.4ms (1.6/2.6 million cycles) for double scalarmult inside NIST P-224/P-256 verif.

2001 Bernstein, $\approx 1.6\times$ faster: 0.7 million cycles on Pentium II for NIST P-224 scalarmult.



2000 Brown–Hankerson–Lóp
Menezes software uses many
cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentiur
2.7 million cycles on Pentiur

2001 Bernstein P-224 scalar
0.7 million cycles on Pentiur
0.8 million cycles on Pentiur
0.9 million cycles on Pentiur
using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentiur

Bad ECDSA-NIST-P-256 design certainly has some impact:

- can't use fastest mulmods;
- can't use fastest curve formulas;
- need an annoying inversion;

etc. Typical estimate: $2\times$ slower.

2000 Brown–Hankerson–López–Menezes on 400MHz Pentium II: 4.0ms/6.4ms (1.6/2.6 million cycles) for double scalarmult inside NIST P-224/P-256 verif.

2001 Bernstein, $\approx 1.6\times$ faster: 0.7 million cycles on Pentium II for NIST P-224 scalarmult.



2000 Brown–Hankerson–López–Menezes software uses many more cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentium II.
2.7 million cycles on Pentium 4.

2001 Bernstein P-224 scalarmult:
0.7 million cycles on Pentium II.
0.8 million cycles on Pentium 4.
0.9 million cycles on Pentium 4 using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentium D.

DSA-NIST-P-256 design
has some impact:
use fastest mulmods;
use fastest curve formulas;
n annoying inversion;
ical estimate: 2× slower.

own–Hankerson–López–
on 400MHz Pentium II:
.4ms (1.6/2.6 million
or double scalarmult
IST P-224/P-256 verif.

rnstein, ≈1.6× faster:
on cycles on Pentium II
P-224 scalarmult.



2000 Brown–Hankerson–López–Menezes software uses many more cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentium II.
2.7 million cycles on Pentium 4.

2001 Bernstein P-224 scalarmult:
0.7 million cycles on Pentium II.
0.8 million cycles on Pentium 4.
0.9 million cycles on Pentium 4 using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentium D.

How did
17 millio
22 millio

Presuma
bad mul

Why did
ECDSA,
underlyi

Why did
previous

Why did

Why did

-P-256 design
impact:
mulmods;
curve formulas;
g inversion;
ate: 2× slower.

erson–López–
Hz Pentium II:
/2.6 million
scalarmult
/P-256 verif.

1.6× faster:
on Pentium II
alarmult.



2000 Brown–Hankerson–López–Menezes software uses many more cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentium II.
2.7 million cycles on Pentium 4.

2001 Bernstein P-224 scalarmult:
0.7 million cycles on Pentium II.
0.8 million cycles on Pentium 4.
0.9 million cycles on Pentium 4 using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentium D.

How did Petit ma
17 million cycles f
22 million cycles f

Presumably some
bad mulmod and

Why did Petit rei
ECDSA, using MI
underlying arithm

Why did Petit not
previous speed lit

Why did Petit cho

Why did BHLM c

esign

;

mulas;

n;

lower.

pez–

m II:

n

t

erif.

er:

m II



2000 Brown–Hankerson–López–Menezes software uses many more cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentium II.
2.7 million cycles on Pentium 4.

2001 Bernstein P-224 scalarmult:
0.7 million cycles on Pentium II.
0.8 million cycles on Pentium 4.
0.9 million cycles on Pentium 4 using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentium D.

How did Petit manage to us
17 million cycles for P-224 v
22 million cycles for P-256 v

Presumably some combinati
bad mulmod and bad curve

Why did Petit reimplement
ECDSA, using MIRACL for t
underlying arithmetic?

Why did Petit not simply cit
previous speed literature?

Why did Petit choose Pentiu

Why did BHLM choose PII?

2000 Brown–Hankerson–López–Menezes software uses many more cycles on P4 than on PII.

e.g., P-224 scalarmult:
1.2 million cycles on Pentium II.
2.7 million cycles on Pentium 4.

2001 Bernstein P-224 scalarmult:
0.7 million cycles on Pentium II.
0.8 million cycles on Pentium 4.
0.9 million cycles on Pentium 4 using compressed keys.

OpenSSL 1.0.1, P-224 verif:
2.0 million cycles on Pentium D.



How did Petit manage to use 17 million cycles for P-224 verif, 22 million cycles for P-256 verif?

Presumably some combination of bad mulmod and bad curve ops.

Why did Petit reimplement ECDSA, using MIRACL for the underlying arithmetic?

Why did Petit not simply cite previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?

own–Hankerson–López–
s software uses many more
n P4 than on PII.

224 scalarmult:
on cycles on Pentium II.
on cycles on Pentium 4.

rnstein P-224 scalarmult:
on cycles on Pentium II.
on cycles on Pentium 4.
on cycles on Pentium 4
mpressed keys.

L 1.0.1, P-224 verif:
on cycles on Pentium D.



How did Petit manage to use
17 million cycles for P-224 verif,
22 million cycles for P-256 verif?

Presumably some combination of
bad mulmod and bad curve ops.

Why did Petit reimplement
ECDSA, using MIRACL for the
underlying arithmetic?

Why did Petit not simply cite
previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?

Petit: "
cryptogr
OpenSS
Authors
comparis
that MIR
performa
elliptic c

kerson–López–
uses many more
on PII.

mult:
on Pentium II.
on Pentium 4.

224 scalarmult:
on Pentium II.
on Pentium 4.
on Pentium 4
keys.

-224 verif:
on Pentium D.



How did Petit manage to use 17 million cycles for P-224 verif, 22 million cycles for P-256 verif?

Presumably some combination of bad mulmod and bad curve ops.

Why did Petit reimplement ECDSA, using MIRACL for the underlying arithmetic?

Why did Petit not simply cite previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?

Petit: “There are
cryptographic libra
OpenSSL and Cry
Authors in [21] pro
comparison and co
that MIRACL has
performance for op
elliptic curves over

ez–
more

m II.
m 4.

mult:
m II.
m 4.
m 4

m D.

How did Petit manage to use 17 million cycles for P-224 verif, 22 million cycles for P-256 verif?

Presumably some combination of bad mulmod and bad curve ops.

Why did Petit reimplement ECDSA, using MIRACL for the underlying arithmetic?

Why did Petit not simply cite previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?



Petit: "There are three mai
cryptographic libraries: MIR
OpenSSL and Crypto++.
Authors in [21] proposed a
comparison and concluded
that MIRACL has the best
performance for operations
elliptic curves over binary fie

How did Petit manage to use 17 million cycles for P-224 verif, 22 million cycles for P-256 verif?

Presumably some combination of bad mulmod and bad curve ops.

Why did Petit reimplement ECDSA, using MIRACL for the underlying arithmetic?

Why did Petit not simply cite previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?



Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

How did Petit manage to use 17 million cycles for P-224 verif, 22 million cycles for P-256 verif?

Presumably some combination of bad mulmod and bad curve ops.

Why did Petit reimplement ECDSA, using MIRACL for the underlying arithmetic?

Why did Petit not simply cite previous speed literature?

Why did Petit choose Pentium D?

Why did BHLM choose PII?



Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

But NIST P-224 and NIST P-256 are defined over prime fields!

[21] says "For elliptic curves over prime fields, OpenSSL has the best performance under all platforms."

Petit manage to use
on cycles for P-224 verif,
on cycles for P-256 verif?

ably some combination of
mod and bad curve ops.

Petit reimplement
using MIRACL for the
ng arithmetic?

Petit not simply cite
speed literature?

Petit choose Pentium D?

BHLM choose PII?



Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

But NIST P-224 and NIST P-256 are defined over prime fields!

[21] says "For elliptic curves over prime fields, OpenSSL has the best performance under all platforms."

More ge
Paper ar
crypto u

*If* the cr
Why is t
Why sho

*If* the cr
Paper is
Look, he
More lik
More lik
funding

nage to use
or P-224 verif,
or P-256 verif?

combination of
oad curve ops.

mplement
RACL for the
etic?

simply cite
erature?

ose Pentium D?

hoose PII?



Petit: “There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields.”

But NIST P-224 and NIST P-256 are defined over prime fields!

[21] says “For elliptic curves over prime fields, OpenSSL has the best performance under all platforms.”

More general situa
Paper analyzes im
crypto upon an ap

*If* the crypto soun
Why is the paper
Why should it be

*If* the crypto soun
Paper is more inte
Look, here’s a spe
More likely to be
More likely to mot
funding to fix the

se
verif,
verif?

on of
ops.

the

te

um D?
s



Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

But NIST P-224 and NIST P-256 are defined over prime fields!

[21] says "For elliptic curves over prime fields, OpenSSL has the best performance under all platforms."

More general situation:
Paper analyzes impact of crypto upon an application.

*If* the crypto sounds fast:
Why is the paper interesting
Why should it be published?

*If* the crypto sounds slower:
Paper is more interesting.
Look, here's a speed problem
More likely to be published.
More likely to motivate funding to fix the problem.

Petit: "There are three main cryptographic libraries: MIRACL, OpenSSL and Crypto++. Authors in [21] proposed a comparison and concluded that MIRACL has the best performance for operations on elliptic curves over binary fields."

But NIST P-224 and NIST P-256 are defined over prime fields!

[21] says "For elliptic curves over prime fields, OpenSSL has the best performance under all platforms."



More general situation:
Paper analyzes impact of crypto upon an application.

*If* the crypto sounds fast:
Why is the paper interesting?
Why should it be published?

*If* the crypto sounds slower:
Paper is more interesting.
Look, here's a speed problem!
More likely to be published.
More likely to motivate funding to fix the problem.

There are three main
raphic libraries: MIRACL,
L and Crypto++.
in [21] proposed a
son and concluded
RACL has the best
ance for operations on
curves over binary fields."

T P-224 and NIST P-256
ned over prime fields!

s "For elliptic curves
me fields, OpenSSL has
performance under all
s."



More general situation:
Paper analyzes impact of
crypto upon an application.

*If* the crypto sounds fast:
Why is the paper interesting?
Why should it be published?

*If* the crypto sounds slower:
Paper is more interesting.
Look, here's a speed problem!
More likely to be published.
More likely to motivate
funding to fix the problem.

Obvious
applicati
deploym

Many ra
answerin
CPU to
literatur
mulmod

Slowest,
are most

Situatior
random
There's
*deliberat*

three main
aries: MIRACL,
pto++.
oposed a
oncluded
the best
perations on
binary fields."
nd NIST P-256
rime fields!
otic curves
OpenSSL has
nce under all



More general situation:
Paper analyzes impact of
crypto upon an application.

*If* the crypto sounds fast:
Why is the paper interesting?
Why should it be published?

*If* the crypto sounds slower:
Paper is more interesting.
Look, here's a speed problem!
More likely to be published.
More likely to motivate
funding to fix the problem.



Obvious question
application conside
deployment: "Is it

Many random met
answering this que
CPU to test? Wh
literature and libra
mulmod, or curve

Slowest, least com
are most likely to

Situation is fully e
randomness + nat
There's no evidenc
*deliberately* slowe

n

ACL,

on

elds.”

P-256

!

has

all



More general situation:
Paper analyzes impact of
crypto upon an application.

*If* the crypto sounds fast:
Why is the paper interesting?
Why should it be published?

*If* the crypto sounds slower:
Paper is more interesting.
Look, here’s a speed problem!
More likely to be published.
More likely to motivate
funding to fix the problem.

Obvious question whenever
application considers crypto
deployment: “Is it fast enou

Many random methodologie
answering this question. Wh
CPU to test? What to take
literature and libraries? Reu
mulmod, or curve ops, or m

Slowest, least competent an
are most likely to be publish

Situation is fully explainable
randomness + natural select
There’s no evidence that Pe
*deliberately* slowed down cry

More general situation:
Paper analyzes impact of
crypto upon an application.

*If* the crypto sounds fast:
Why is the paper interesting?
Why should it be published?

*If* the crypto sounds slower:
Paper is more interesting.
Look, here's a speed problem!
More likely to be published.
More likely to motivate
funding to fix the problem.



Obvious question whenever an
application considers crypto
deployment: "Is it fast enough?"

Many random methodologies for
answering this question. Which
CPU to test? What to take from
literature and libraries? Reuse
mulmod, or curve ops, or more?

Slowest, least competent answers
are most likely to be published.

Situation is fully explainable by
randomness + natural selection.
There's no evidence that Petit
*deliberately* slowed down crypto.

neral situation:
nalyzes impact of
pon an application.

rypto sounds fast:
the paper interesting?
ould it be published?

rypto sounds slower:
more interesting.
ere's a speed problem!
ely to be published.
ely to motivate
to fix the problem.



Obvious question whenever an application considers crypto deployment: "Is it fast enough?"

Many random methodologies for answering this question. Which CPU to test? What to take from literature and libraries? Reuse mulmod, or curve ops, or more?

Slowest, least competent answers are most likely to be published.

Situation is fully explainable by randomness + natural selection. There's no evidence that Petit *deliberately* slowed down crypto.

Paper in
software
incentive
slow, an
report it

Paper w
function
lengths,
timing n
maximiz
from old

This is n
what ma

ation:
pact of
plication.

ds fast:
interesting?
published?

ds slower:
eresting.
ed problem!
published.
tivate
problem.



Obvious question whenever an application considers crypto deployment: “Is it fast enough?”

Many random methodologies for answering this question. Which CPU to test? What to take from literature and libraries? Reuse mulmod, or curve ops, or more?

Slowest, least competent answers are most likely to be published.

Situation is fully explainable by randomness + natural selection. There’s no evidence that Petit *deliberately* slowed down crypto.

Paper introducing
software or hardwa
incentive to report
slow, and analogou
report its own cryp

Paper will naturall
functions, paramet
lengths, platforms,
timing mechanism
maximize reported
from old to new.

This is not the sar
what matters mos

g?

?

m!



Obvious question whenever an application considers crypto deployment: “Is it fast enough?”

Many random methodologies for answering this question. Which CPU to test? What to take from literature and libraries? Reuse mulmod, or curve ops, or more?

Slowest, least competent answers are most likely to be published.

Situation is fully explainable by randomness + natural selection. There’s no evidence that Petit *deliberately* slowed down crypto.



Paper introducing new crypt
software or hardware has sa
incentive to report older cry
slow, and analogous incentiv
report its own crypto as fast

Paper will naturally select
functions, parameters, input
lengths, platforms, I/O form
timing mechanism, etc. that
maximize reported improven
from old to new.

This is not the same as sele
what matters most for the u

Obvious question whenever an application considers crypto deployment: "Is it fast enough?"

Many random methodologies for answering this question. Which CPU to test? What to take from literature and libraries? Reuse mulmod, or curve ops, or more?

Slowest, least competent answers are most likely to be published.

Situation is fully explainable by randomness + natural selection. There's no evidence that Petit *deliberately* slowed down crypto.



Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.

question whenever an
ion considers crypto
ent: “Is it fast enough?”

ndom methodologies for
g this question. Which
test? What to take from
e and libraries? Reuse
, or curve ops, or more?

least competent answers
t likely to be published.

n is fully explainable by
ness + natural selection.
no evidence that Petit
*tely* slowed down crypto.



Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.

Bit oper
(assumin
as listed

| key | ops |
|---|---|
| 128 | 88 |
| 128 | 100 |
| 128 | 117 |
| 256 | 144 |
| 128 | 147 |
| 256 | 156 |
| 128 | 162 |
| 128 | 202 |
| 256 | 283 |

whenever an
ers crypto
fast enough?"

thodologies for
estion. Which
at to take from
aries? Reuse
ops, or more?

petent answers
be published.

xplainable by
tural selection.
ce that Petit
d down crypto.



Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.

Bit operations per
(assuming precom
as listed in recent

| key | ops/bit | ciph |
|---|---|---|
| 128 | 88 | Simo |
| 128 | 100 | NOE |
| 128 | 117 | Skin |
| 256 | 144 | Simo |
| 128 | 147.2 | PRE |
| 256 | 156 | Skin |
| 128 | 162.75 | Picc |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

an

gh?”

s for
hich
from
se
ore?

swers
ed.

by
tion.
tit
ypto.



Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.

Bit operations per bit of pla
(assuming precomputed sub
as listed in recent Skinny pa

| key | ops/bit | cipher |
|---|---|---|
| 128 | 88 | Simon: 60 ops |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 144 | Simon: 106 op |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.



Bit operations per bit of plaintext (assuming precomputed subkeys), as listed in recent Skinny paper:

| key | ops/bit | cipher |
|---|---|---|
| 128 | 88 | Simon: 60 ops broken |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 144 | Simon: 106 ops broken |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

Paper introducing new crypto software or hardware has same incentive to report older crypto as slow, and analogous incentive to report its own crypto as fast.

Paper will naturally select functions, parameters, input lengths, platforms, I/O format, timing mechanism, etc. that maximize reported improvement from old to new.

This is not the same as selecting what matters most for the users.



Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

| key | ops/bit | cipher |
|---|---|---|
| 256 | 54 | Salsa20/8 |
| 256 | 78 | Salsa20/12 |
| 128 | 88 | Simon: 60 ops broken |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 126 | Salsa20 |
| 256 | 144 | Simon: 106 ops broken |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

troducing new crypto
or hardware has same
e to report older crypto as
d analogous incentive to
s own crypto as fast.

ill naturally select
s, parameters, input
platforms, I/O format,
nechanism, etc. that
e reported improvement
l to new.

not the same as selecting
atters most for the users.



Bit operations per bit of plaintext
(assuming precomputed subkeys),
not entirely listed in Skinny paper:

| key | ops/bit | cipher |
|---|---|---|
| 256 | 54 | Salsa20/8 |
| 256 | 78 | Salsa20/12 |
| 128 | 88 | Simon: 60 ops broken |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 126 | Salsa20 |
| 256 | 144 | Simon: 106 ops broken |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

Many ba
backed b

e.g. Do
optimize
the olde
Rely on

"We cor
most arc
do much
complete
heuristic
get little
where th
slightly

new crypto
are has same
older crypto as
us incentive to
oto as fast.

y select
ters, input
, I/O format,
, etc. that
l improvement

me as selecting
t for the users.



Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

| key | ops/bit | cipher |
|---|---|---|
| 256 | 54 | Salsa20/8 |
| 256 | 78 | Salsa20/12 |
| 128 | 88 | Simon: 60 ops broken |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 126 | Salsa20 |
| 256 | 144 | Simon: 106 ops broken |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

Many bad example
backed by tons of

e.g. Do we bother
optimized impleme
the older crypto?
Rely on "optimizin

"We come so close
most architectures
do much more wit
complete algorithm
heuristics. We can
get little niggles h
where the heuristic
slightly wrong ans

to
me
pto as
ve to
t.

at,

nent

cting
users.



Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

| key | ops/bit | cipher |
|---|---|---|
| 256 | 54 | Salsa20/8 |
| 256 | 78 | Salsa20/12 |
| 128 | 88 | Simon: 60 ops broken |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 126 | Salsa20 |
| 256 | 144 | Simon: 106 ops broken |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |

Many bad examples to imita
backed by tons of misinform

e.g. Do we bother searching
optimized implementations o
the older crypto? Take any
Rely on "optimizing" compil

"We come so close to optim
most architectures that we c
do much more without using
complete algorithms instead
heuristics. We can only try t
get little niggles here and th
where the heuristics get
slightly wrong answers."

Bit operations per bit of plaintext (assuming precomputed subkeys), not entirely listed in Skinny paper:

| key | ops/bit | cipher |
|---|---|---|
| 256 | 54 | Salsa20/8 |
| 256 | 78 | Salsa20/12 |
| 128 | 88 | Simon: 60 ops broken |
| 128 | 100 | NOEKEON |
| 128 | 117 | Skinny |
| 256 | 126 | Salsa20 |
| 256 | 144 | Simon: 106 ops broken |
| 128 | 147.2 | PRESENT |
| 256 | 156 | Skinny |
| 128 | 162.75 | Piccolo |
| 128 | 202.5 | AES |
| 256 | 283.5 | AES |



Many bad examples to imitate, backed by tons of misinformation.

e.g. Do we bother searching for optimized implementations of the older crypto? Take any code! Rely on "optimizing" compiler!

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

ations per bit of plaintext
ng precomputed subkeys),
rely listed in Skinny paper:

| s/bit | cipher |
|---|---|
| 4 | Salsa20/8 |
| 8 | Salsa20/12 |
| 8 | Simon: 60 ops broken |
| 0 | NOEKEON |
| 7 | Skinny |
| 6 | Salsa20 |
| 4 | Simon: 106 ops broken |
| 7.2 | PRESENT |
| 6 | Skinny |
| 2.75 | Piccolo |
| 2.5 | AES |
| 3.5 | AES |



Many bad examples to imitate,
backed by tons of misinformation.

e.g. Do we bother searching for optimized implementations of the older crypto? Take any code! Rely on “optimizing” compiler!

“We come so close to optimal on most architectures that we can’t do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers.”

Reality i

bit of plaintext
puted subkeys),
in Skinny paper:

er

a20/8
a20/12
on: 60 ops broken
EKEON
ny
a20
on: 106 ops broken
SENT
ny
olo

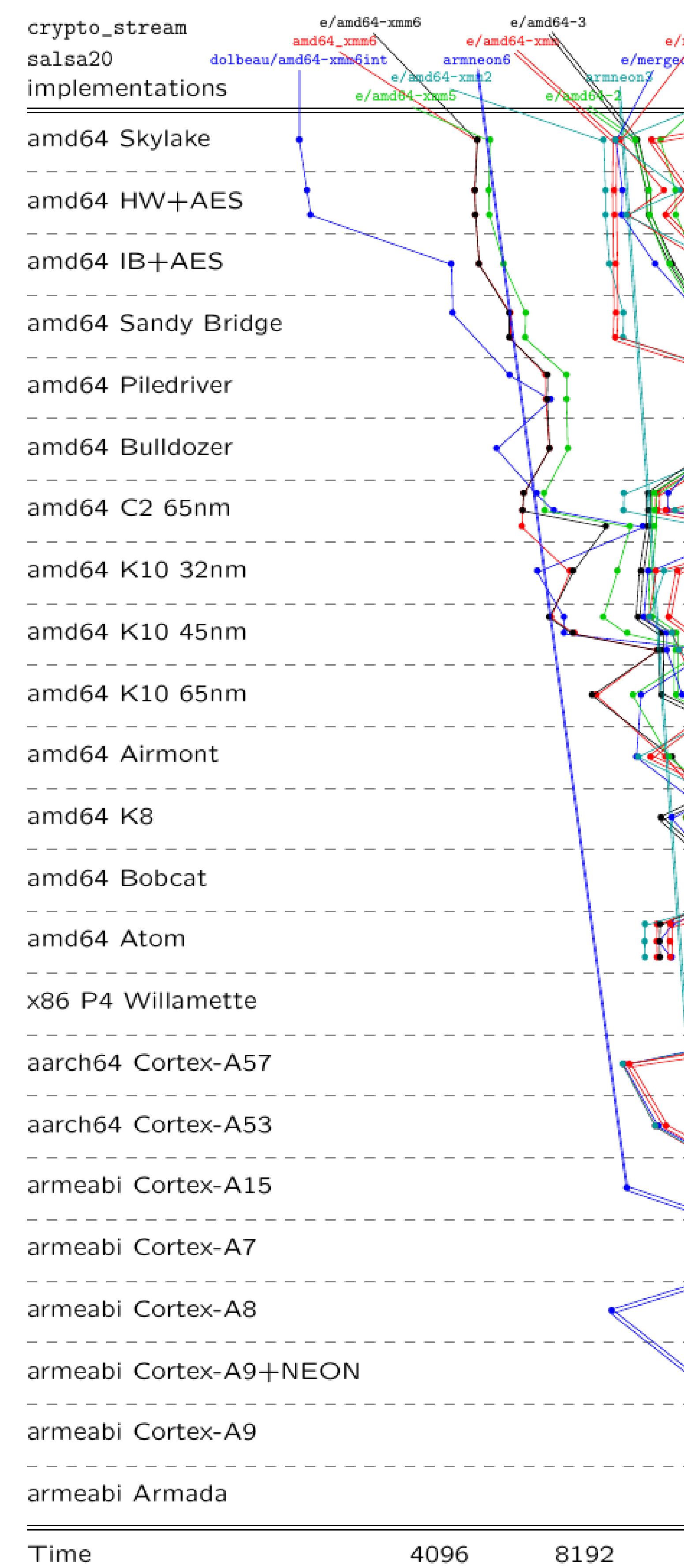


Many bad examples to imitate, backed by tons of misinformation.

e.g. Do we bother searching for optimized implementations of the older crypto? Take any code! Rely on "optimizing" compiler!

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

Reality is more co



intext
keys),
paper:

broken

s broken

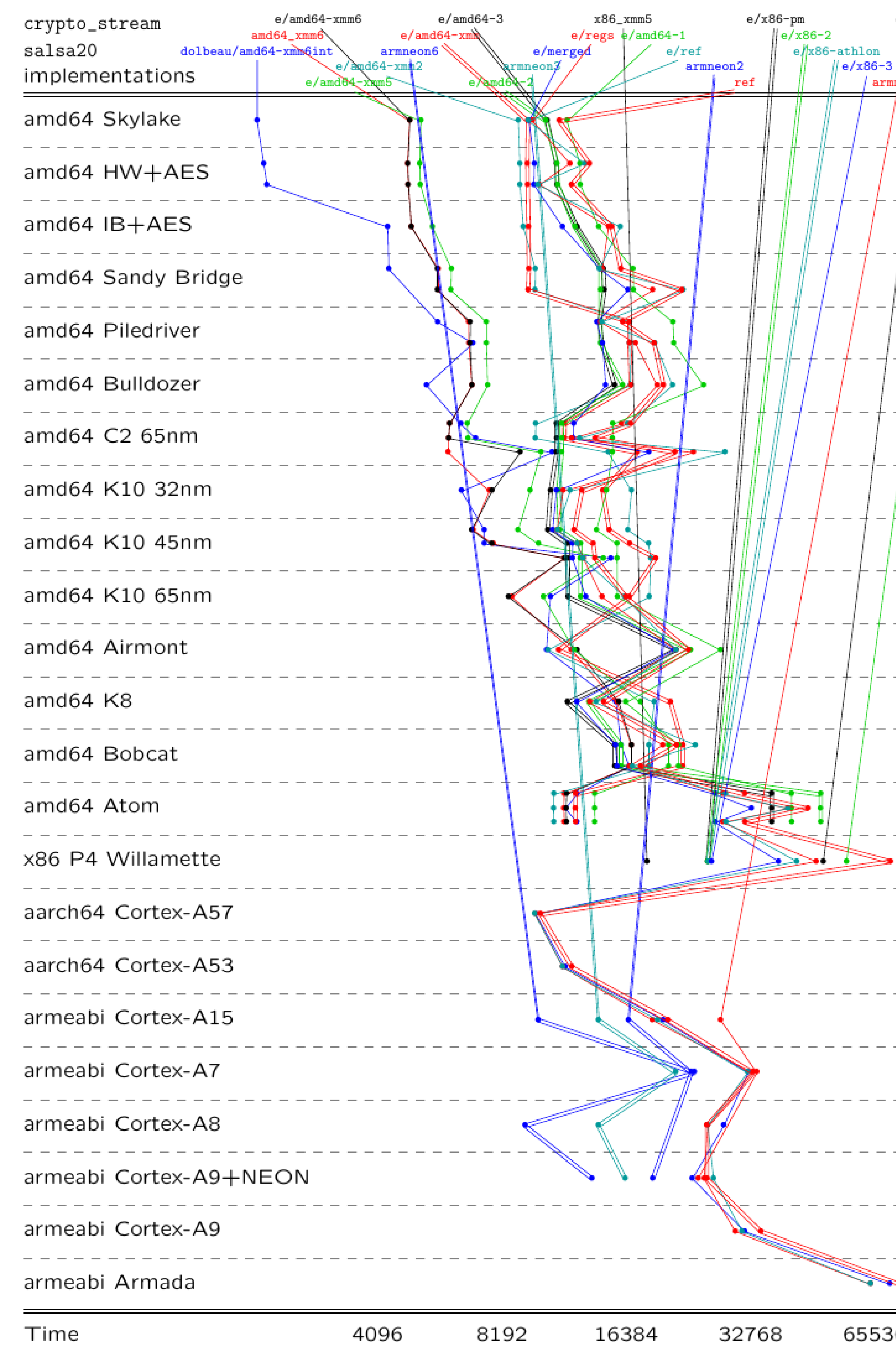
Many bad examples to imitate, backed by tons of misinformation.

e.g. Do we bother searching for optimized implementations of the older crypto? Take any code! Rely on "optimizing" compiler!

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."



Reality is more complicated:



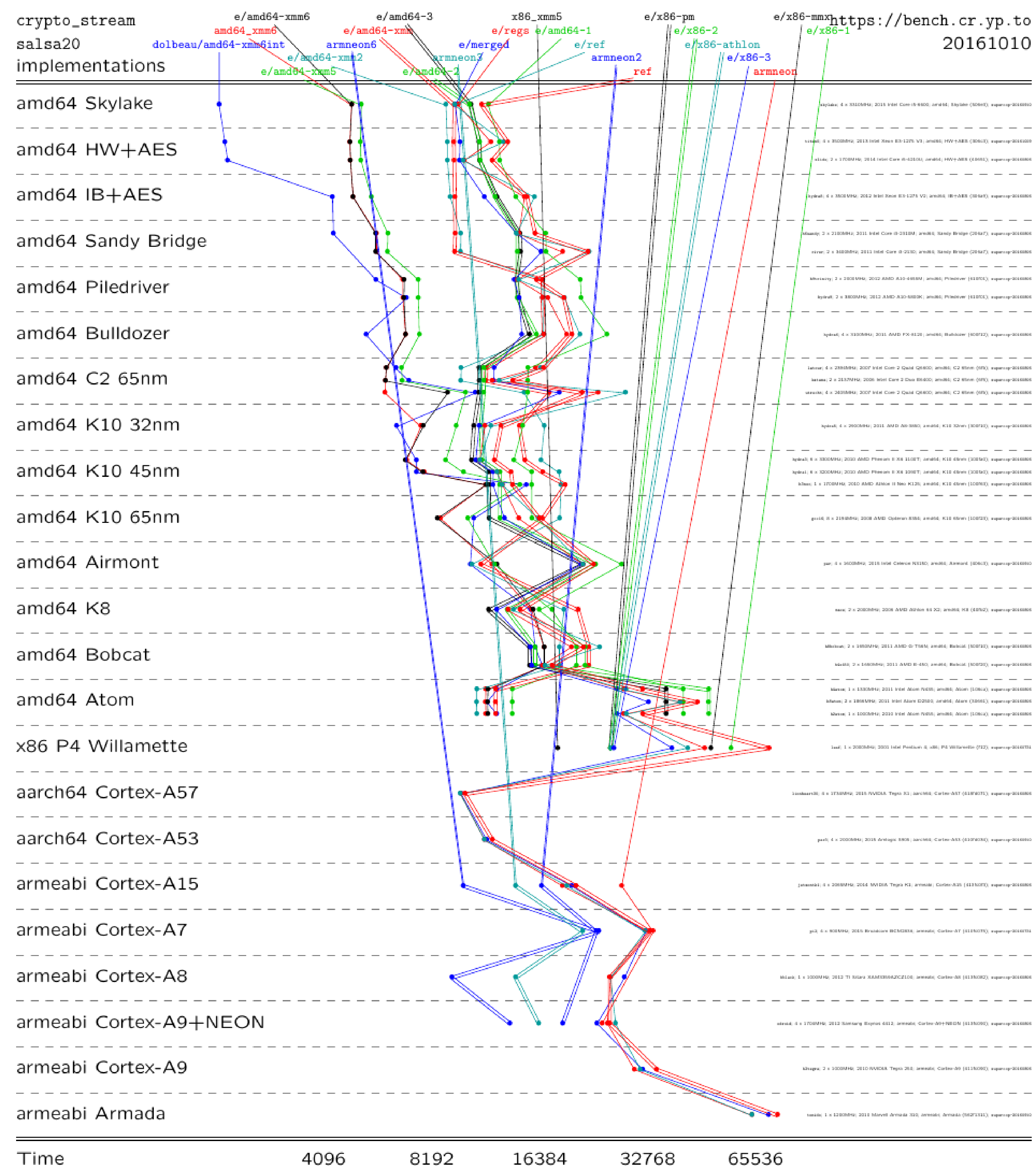
Many bad examples to imitate,
backed by tons of misinformation.

e.g. Do we bother searching for optimized implementations of the older crypto? Take any code! Rely on "optimizing" compiler!

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."



Reality is more complicated:



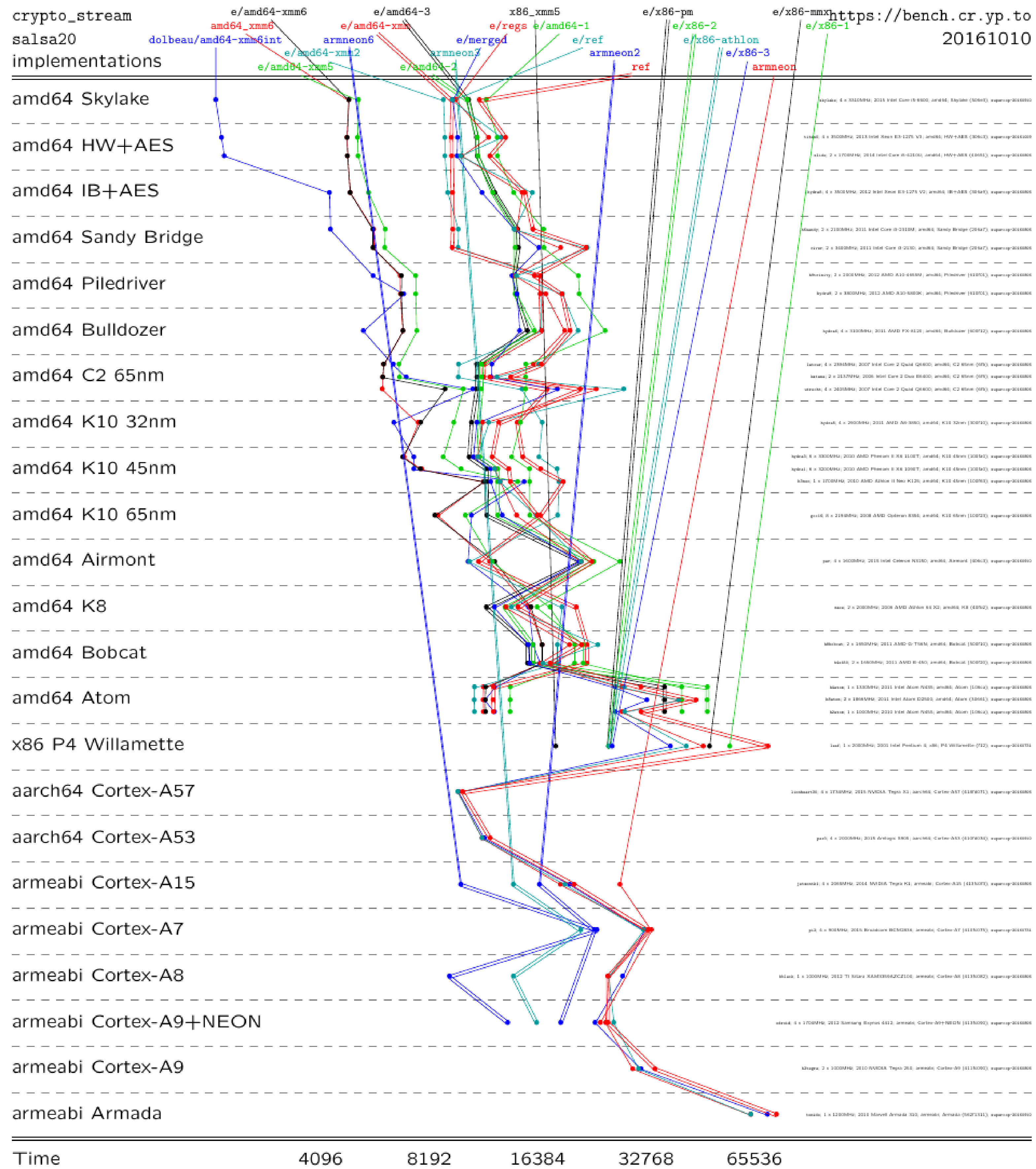
ad examples to imitate,
by tons of misinformation.

we bother searching for
ed implementations of
r crypto? Take any code!
"optimizing" compiler!

me so close to optimal on
chitectures that we can't
n more without using NP
e algorithms instead of
s. We can only try to
e niggles here and there
he heuristics get
wrong answers."



Reality is more complicated:




SUPERC
includes
of 595 c
>20 imp

Haswell:
impleme
`gcc -O3`
is 6.15×
Salsa20

`merged`
with "m
optimiza
compiler

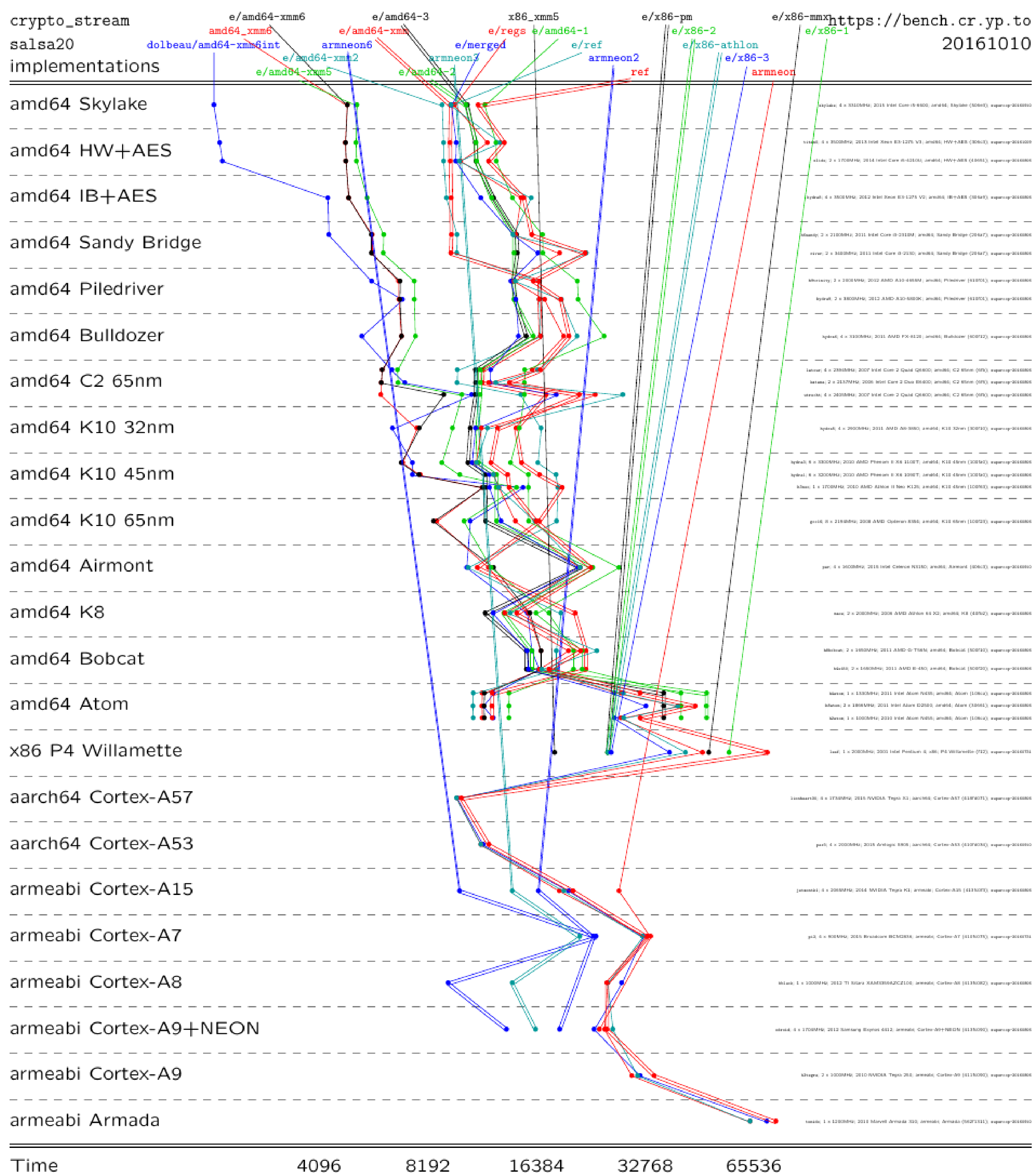
es to imitate,
misinformation.

searching for
entations of
Take any code!
ng" compiler!

e to optimal on
s that we can't
hout using NP
ns instead of
n only try to
ere and there
cs get
wers."



Reality is more complicated:




SUPERCOP benc
includes 2155 imp
of 595 cryptograp
>20 implementati

Haswell: Reasona
implementation c
gcc -O3 -fomit-
is 6.15× slower th
Salsa20 implemen

merged implemen
with "machine-ind
optimizations and
compiler options:

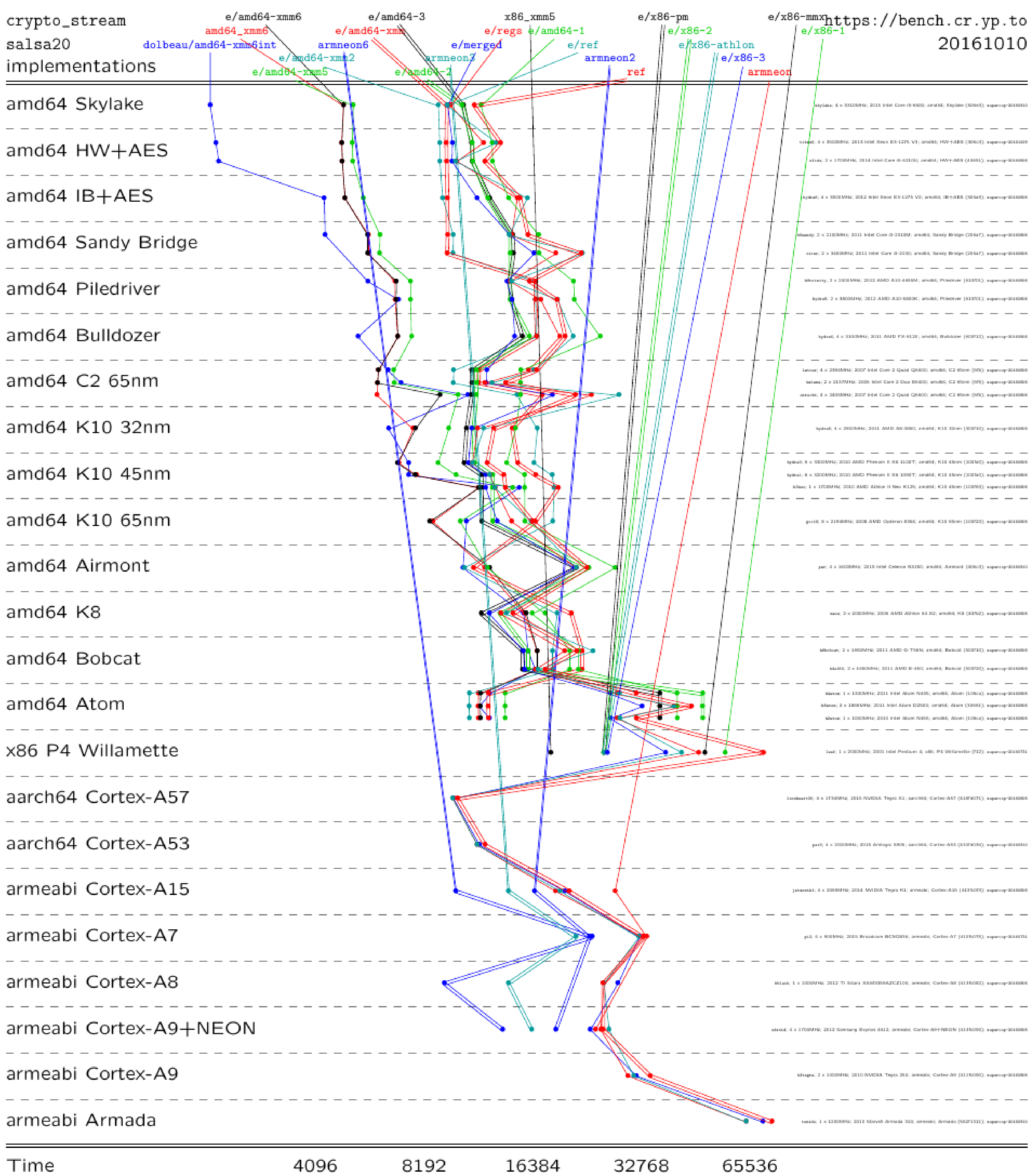
ate,
ation.

for
of
code!
ler!

al on
can't
g NP
of
to
ere



Reality is more complicated:

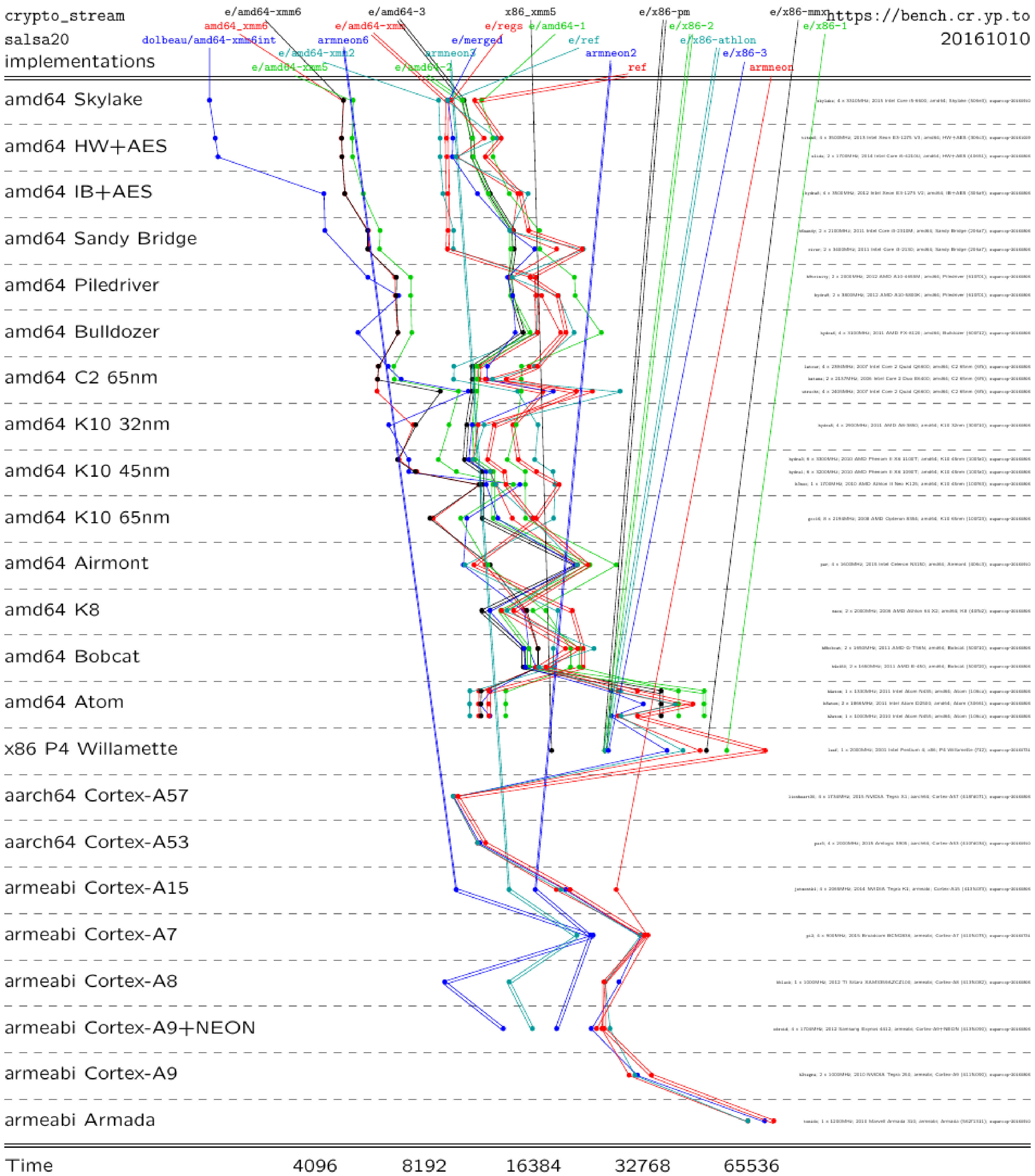



SUPERCOP benchmarking t
includes 2155 implementatio
of 595 cryptographic primiti
>20 implementations of Sal

Haswell: Reasonably simple
implementation compiled wi
`gcc -O3 -fomit-frame-po`
is 6.15× slower than fastest
Salsa20 implementation.

`merged` implementation
with "machine-independent"
optimizations and best of 12
compiler options: 4.52× slo

# Reality is more complicated:



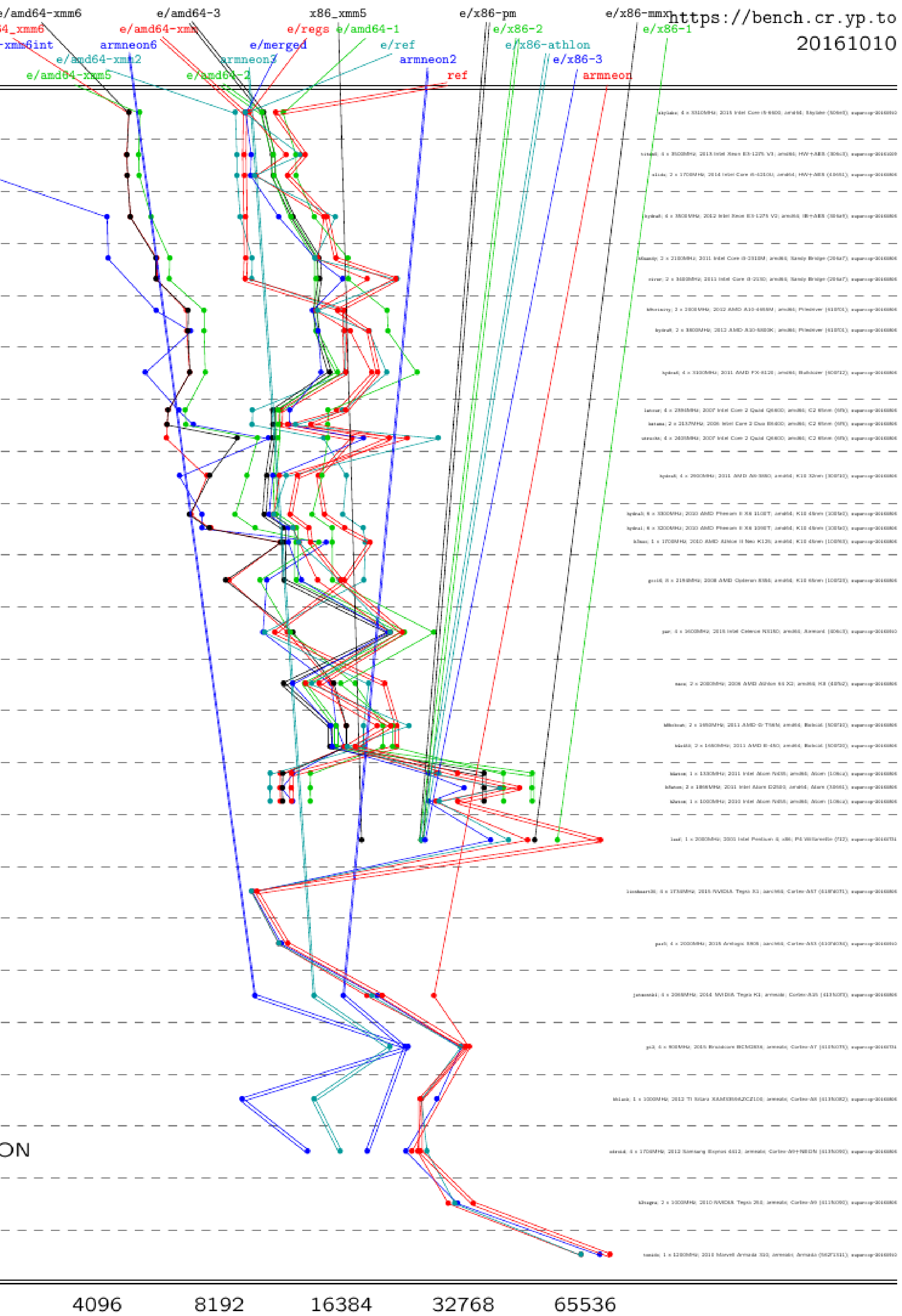



SUPERCOP benchmarking toolkit includes 2155 implementations of 595 cryptographic primitives. >20 implementations of Salsa20.

Haswell: Reasonably simple `ref` implementation compiled with `gcc -O3 -fomit-frame-pointer` is 6.15× slower than fastest Salsa20 implementation.

`merged` implementation with "machine-independent" optimizations and best of 121 compiler options: 4.52× slower.

s more complicated:






SUPERCOP benchmarking toolkit includes 2155 implementations of 595 cryptographic primitives. >20 implementations of Salsa20.

Haswell: Reasonably simple `ref` implementation compiled with `gcc -O3 -fomit-frame-pointer` is 6.15× slower than fastest Salsa20 implementation.

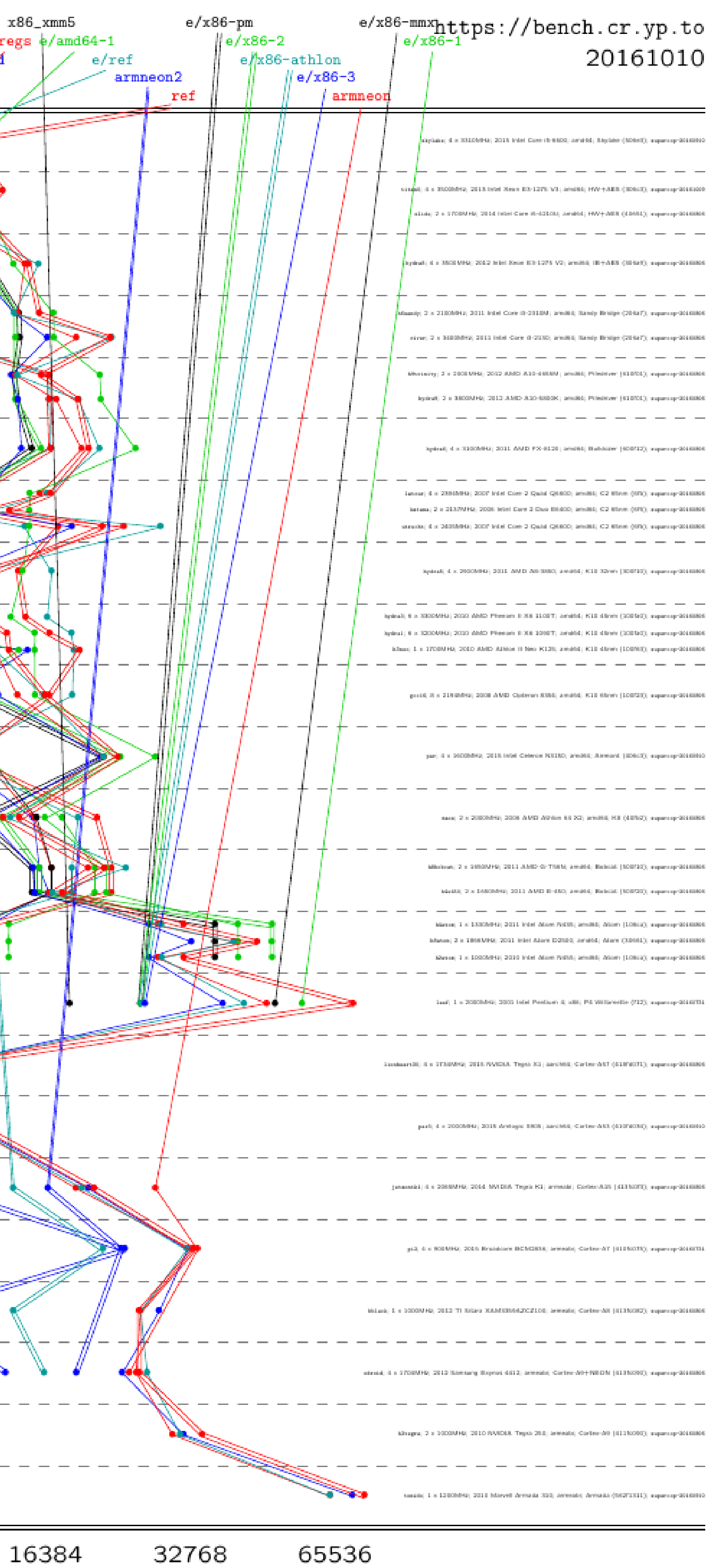
`merged` implementation with "machine-independent" optimizations and best of 121 compiler options: 4.52× slower.

Another
lattice-b
means g
of rando

2017.03
Valencia
Regazzo
sources
discrete
benchma

Qualitat
choice o
sampling

mplicated:






SUPERCOP benchmarking toolkit includes 2155 implementations of 595 cryptographic primitives. >20 implementations of Salsa20.

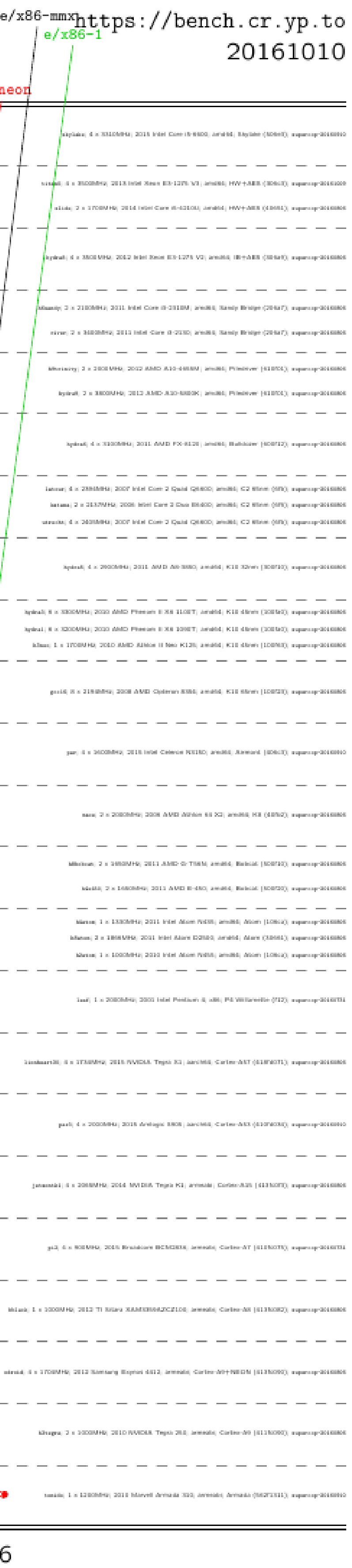
Haswell: Reasonably simple `ref` implementation compiled with `gcc -O3 -fomit-frame-pointer` is 6.15× slower than fastest Salsa20 implementation.

`merged` implementation with "machine-independent" optimizations and best of 121 compiler options: 4.52× slower.

Another interestin
lattice-based signi
means generating
of random Gaussia

2017.03 Brannigar
Valencia–O'Sulliva
Regazzoni "An inv
sources of random
discrete Gaussian
benchmarks for RN

Qualitatively large
choice of RNG ⇒
sampling ⇒ cost






SUPERCOP benchmarking toolkit includes 2155 implementations of 595 cryptographic primitives. >20 implementations of Salsa20.

Haswell: Reasonably simple `ref` implementation compiled with `gcc -O3 -fomit-frame-pointer` is 6.15× slower than fastest Salsa20 implementation.

`merged` implementation with "machine-independent" optimizations and best of 121 compiler options: 4.52× slower.

Another interesting example
lattice-based signing typicall
means generating a huge nu
of random Gaussian samples

2017.03 Brannigan–Smyth–
Valencia–O'Sullivan–Güneys
Regazzoni "An investigation
sources of randomness withi
discrete Gaussian sampling"
benchmarks for RNGs, samp

Qualitatively large impacts:
choice of RNG ⇒ cost of
sampling ⇒ cost of signing.

SUPERCOP benchmarking toolkit includes 2155 implementations of 595 cryptographic primitives. >20 implementations of Salsa20.

Haswell: Reasonably simple `ref` implementation compiled with `gcc -O3 -fomit-frame-pointer` is 6.15× slower than fastest Salsa20 implementation.

`merged` implementation with "machine-independent" optimizations and best of 121 compiler options: 4.52× slower.



Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG ⇒ cost of sampling ⇒ cost of signing.

COP benchmarking toolkit
2155 implementations
ryptographic primitives.
plementations of Salsa20.

Reasonably simple `ref`
ntation compiled with
`-fomit-frame-pointer`
slower than fastest
implementation.

implementation
achine-independent"
tions and best of 121
options: 4.52× slower.



Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG ⇒ cost of sampling ⇒ cost of signing.

Two exa
in this 2
Skylake

383.69 M
cycles/b
using AE
(32 cycle

nmarking toolkit
lementations
hic primitives.
ons of Salsa20.

ply simple `ref`
ompiled with
`frame-pointer`
an fastest
tation.

tation
lependent"
best of 121
4.52× slower.



Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG ⇒ cost of sampling ⇒ cost of signing.

Two examples of s
in this 2017 paper
Skylake (Intel Cor

383.69 MByte/sec
cycles/byte) for A
using AES-NI; 106
(32 cycles/byte) fo

toolkit
ons
ves.
sa20.

ref
th
inter

'
21
wer.



Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG $\Rightarrow$ cost of sampling $\Rightarrow$ cost of signing.

Two examples of speed repo
in this 2017 paper for a 3.4
Skylake (Intel Core i7-6700)

383.69 MByte/sec (8.86
cycles/byte) for AES CTR-D
using AES-NI; 106.07 MByt
(32 cycles/byte) for ChaCha

Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG ⇒ cost of sampling ⇒ cost of signing.



Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

Another interesting example: lattice-based signing typically means generating a huge number of random Gaussian samples.

2017.03 Brannigan–Smyth–Oder–Valencia–O'Sullivan–Güneysu–Regazzoni "An investigation of sources of randomness within discrete Gaussian sampling": benchmarks for RNGs, samplers.

Qualitatively large impacts: choice of RNG $\Rightarrow$ cost of sampling $\Rightarrow$ cost of signing.



Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

But wait. eBACS reports 0.92 cycles/byte for AES-256-CTR, 1.18 cycles/byte for ChaCha20.

Author non-response: "essential for us to examine standard open implementations". Slow ones?

interesting example:
ased signing typically
enerating a huge number
m Gaussian samples.

Brannigan–Smyth–Oder–
–O'Sullivan–Güneysu–
ni "An investigation of
of randomness within
Gaussian sampling":
arks for RNGs, samplers.

ively large impacts:
f RNG ⇒ cost of
g ⇒ cost of signing.

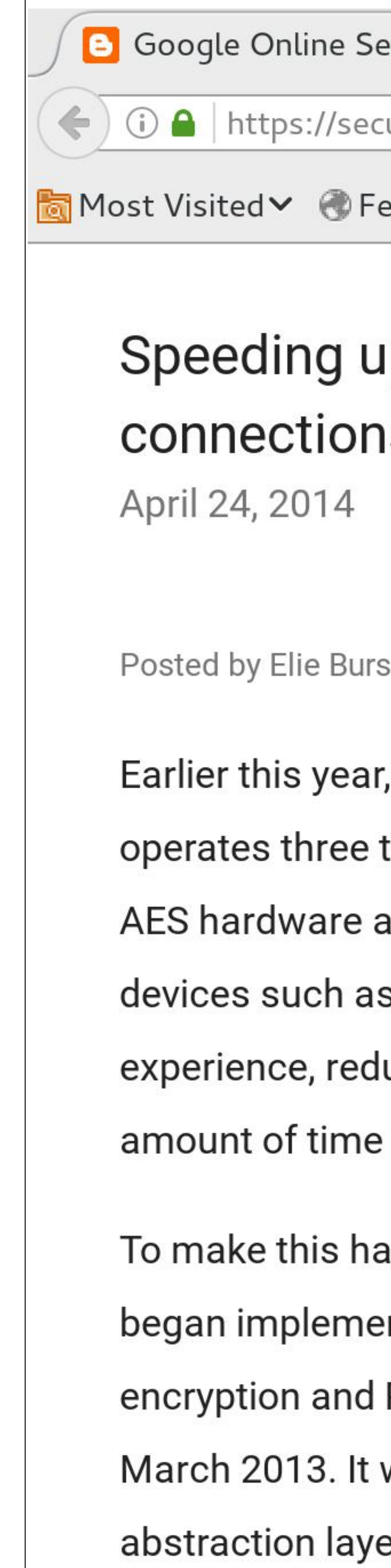


Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

But wait. eBACS reports
0.92 cycles/byte for AES-256-CTR,
1.18 cycles/byte for ChaCha20.

Author non-response: "essential for us to examine standard open implementations". Slow ones?


Google Online Se
https://secu
Most Visited Fe

Speeding u
connection
April 24, 2014

Posted by Elie Burs

Earlier this year,
operates three t
AES hardware a
devices such as
experience, redu
amount of time

To make this ha
began implemen
encryption and
March 2013. It w
abstraction laye

g example:
ng typically
a huge number
an samples.

n–Smyth–Oder–
n–Güneysu–
vestigation of
ness within
sampling":
NGs, samplers.

impacts:
cost of
of signing.

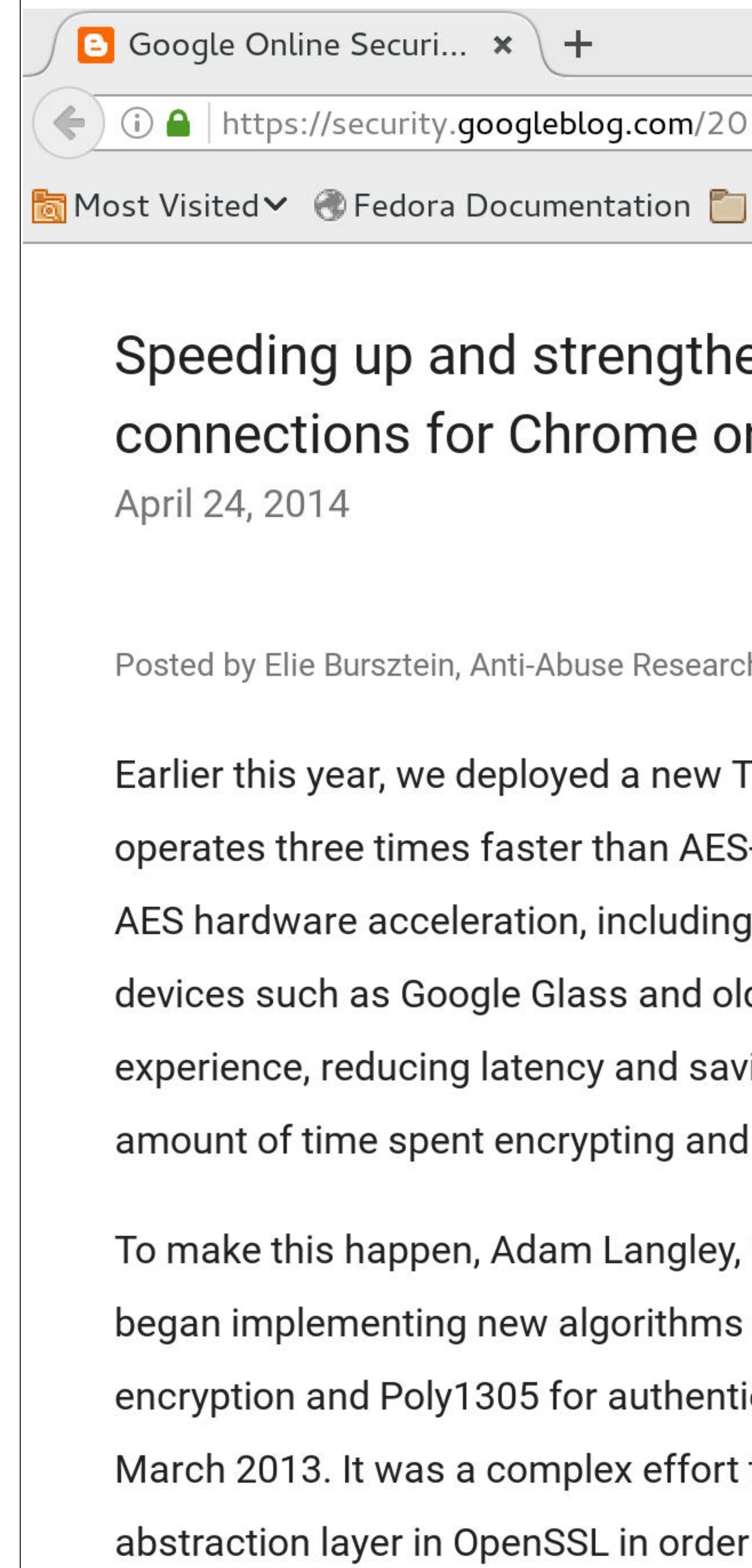


Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

But wait. eBACS reports 0.92 cycles/byte for AES-256-CTR, 1.18 cycles/byte for ChaCha20.

Author non-response: "essential for us to examine standard open implementations". Slow ones?




Speeding up and strengthe
connections for Chrome o
April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Researc

Earlier this year, we deployed a new T
operates three times faster than AES-
AES hardware acceleration, including
devices such as Google Glass and ol
experience, reducing latency and sav
amount of time spent encrypting and

To make this happen, Adam Langley,
began implementing new algorithms
encryption and Poly1305 for authenti
March 2013. It was a complex effort
abstraction layer in OpenSSL in order

:
y
mber
s.
Oder–
u–
of
n
:
lers.

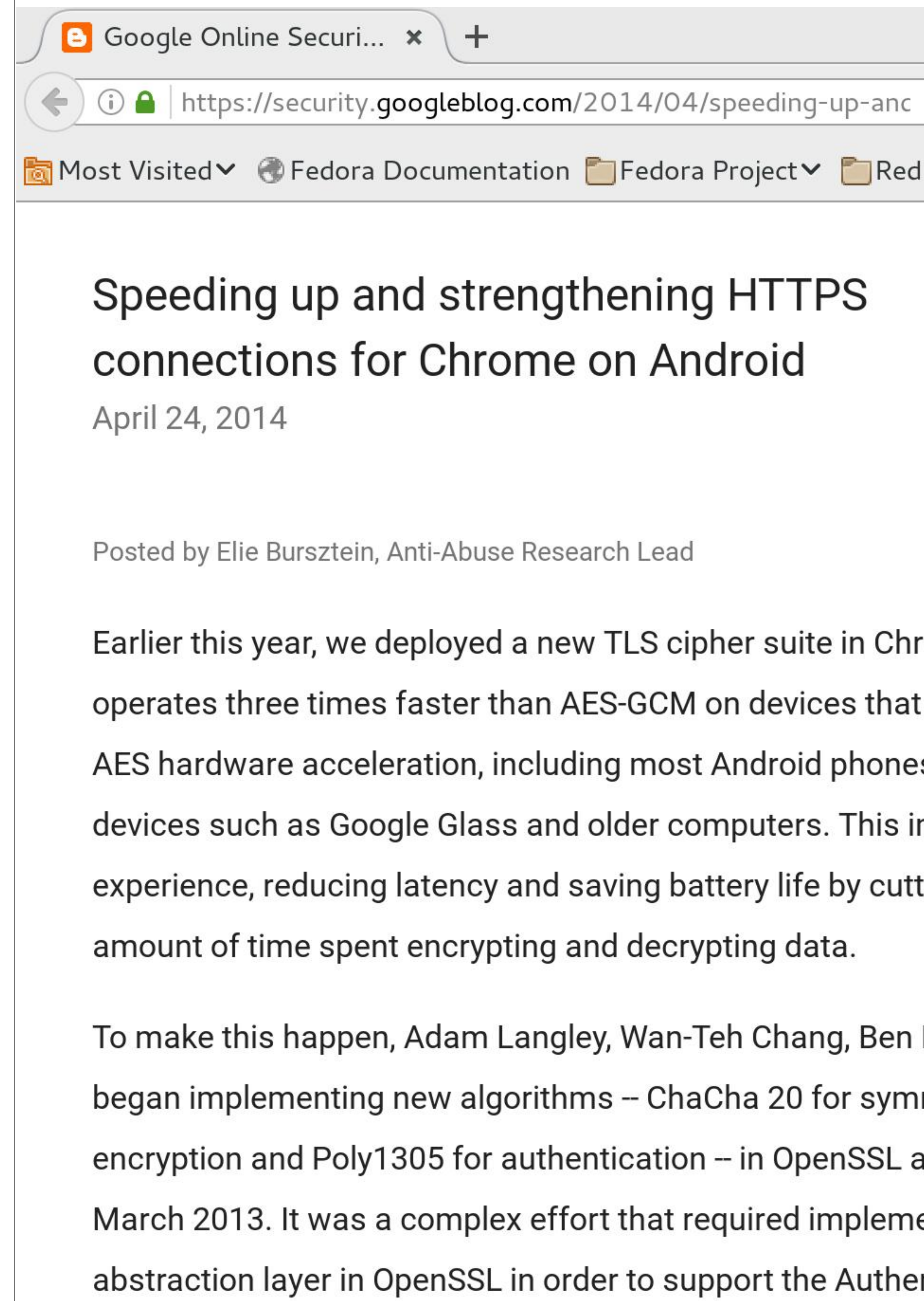


Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

But wait. eBACS reports 0.92 cycles/byte for AES-256-CTR, 1.18 cycles/byte for ChaCha20.

Author non-response: "essential for us to examine standard open implementations". Slow ones?



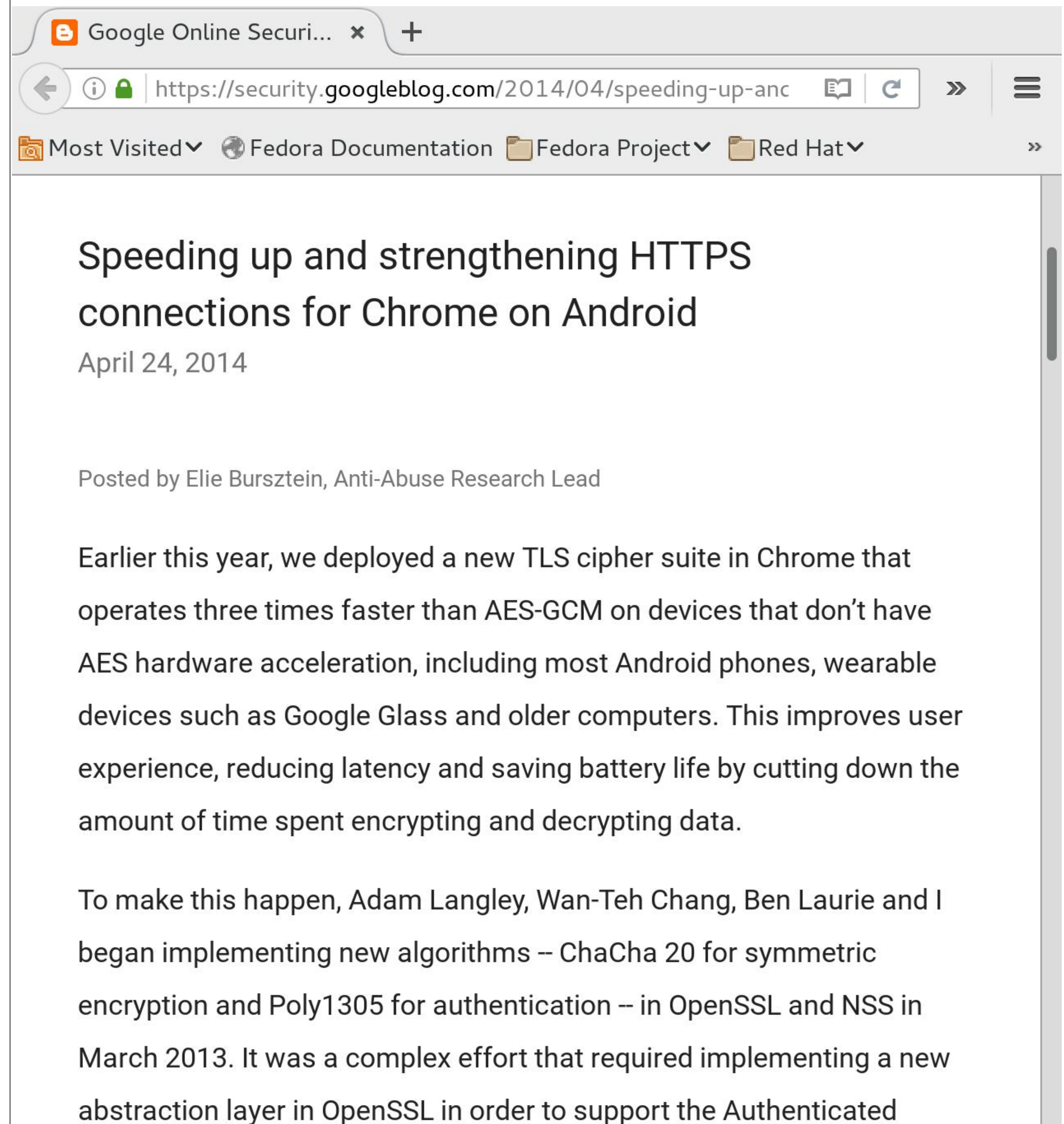
Two examples of speed reported in this 2017 paper for a 3.4GHz Skylake (Intel Core i7-6700):

383.69 MByte/sec (8.86 cycles/byte) for AES CTR-DRBG using AES-NI; 106.07 MByte/sec (32 cycles/byte) for ChaCha20.

But wait. eBACS reports 0.92 cycles/byte for AES-256-CTR, 1.18 cycles/byte for ChaCha20.

Author non-response: "essential for us to examine standard open implementations". Slow ones?






Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated

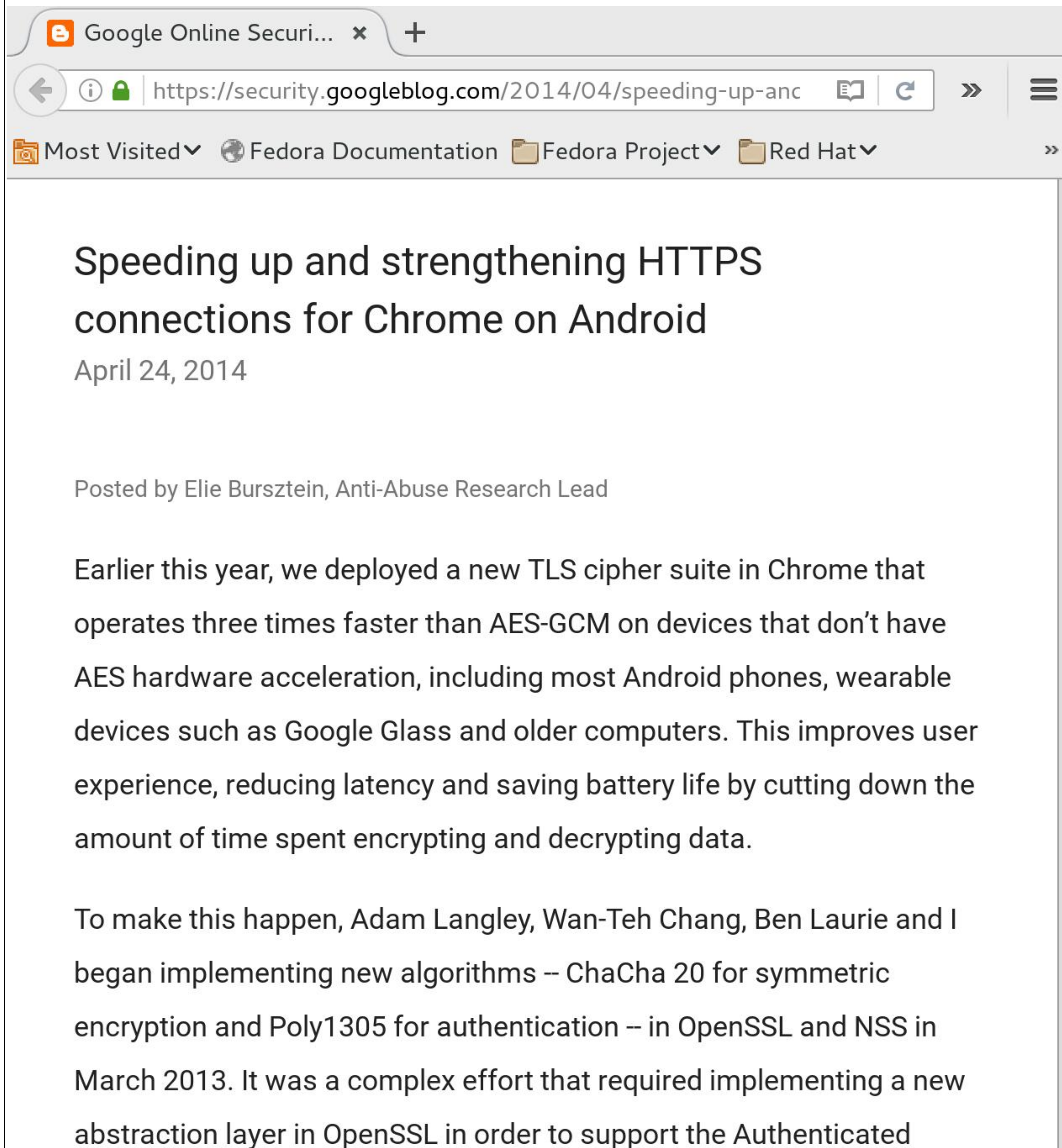
mples of speed reported
017 paper for a 3.4GHz
(Intel Core i7-6700):

MByte/sec (8.86
yte) for AES CTR-DRBG
ES-NI; 106.07 MByte/sec
es/byte) for ChaCha20.

t. eBACS reports
les/byte for AES-256-CTR,
les/byte for ChaCha20.

non-response: “essential
examine standard open
ntations”. Slow ones?




Google Online Securi...
https://security.googleblog.com/2014/04/speeding-up-anc
Most Visited Fedora Documentation Fedora Project Red Hat

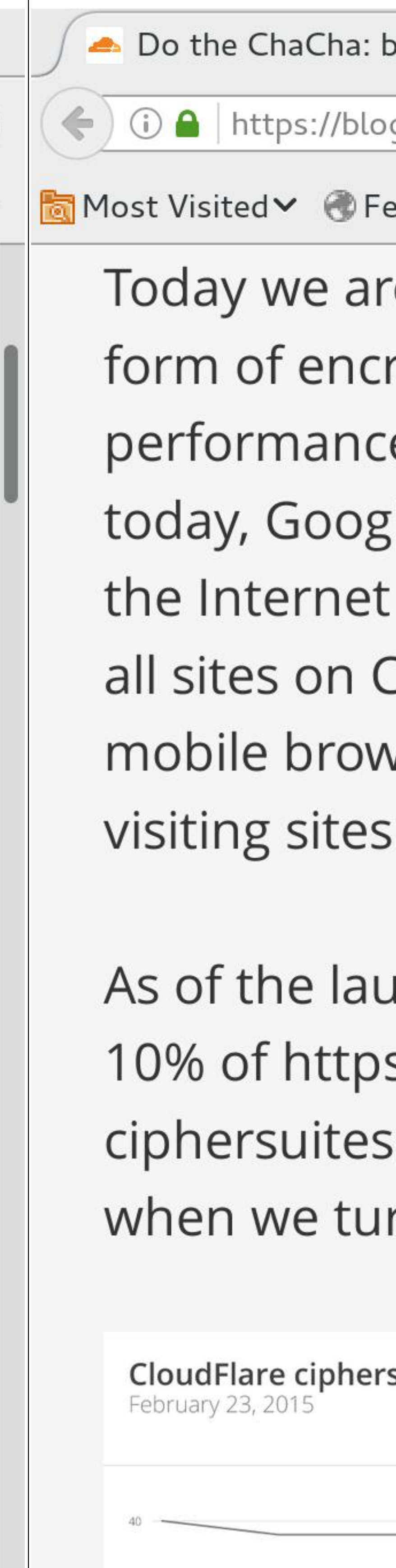
Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated


Do the ChaCha: b
https://blog
Most Visited Fe

Today we ar
form of encr
performanc
today, Goog
the Internet
all sites on C
mobile brow
visiting sites

As of the lau
10% of https
ciphersuites
when we tur

CloudFlare ciphers
February 23, 2015

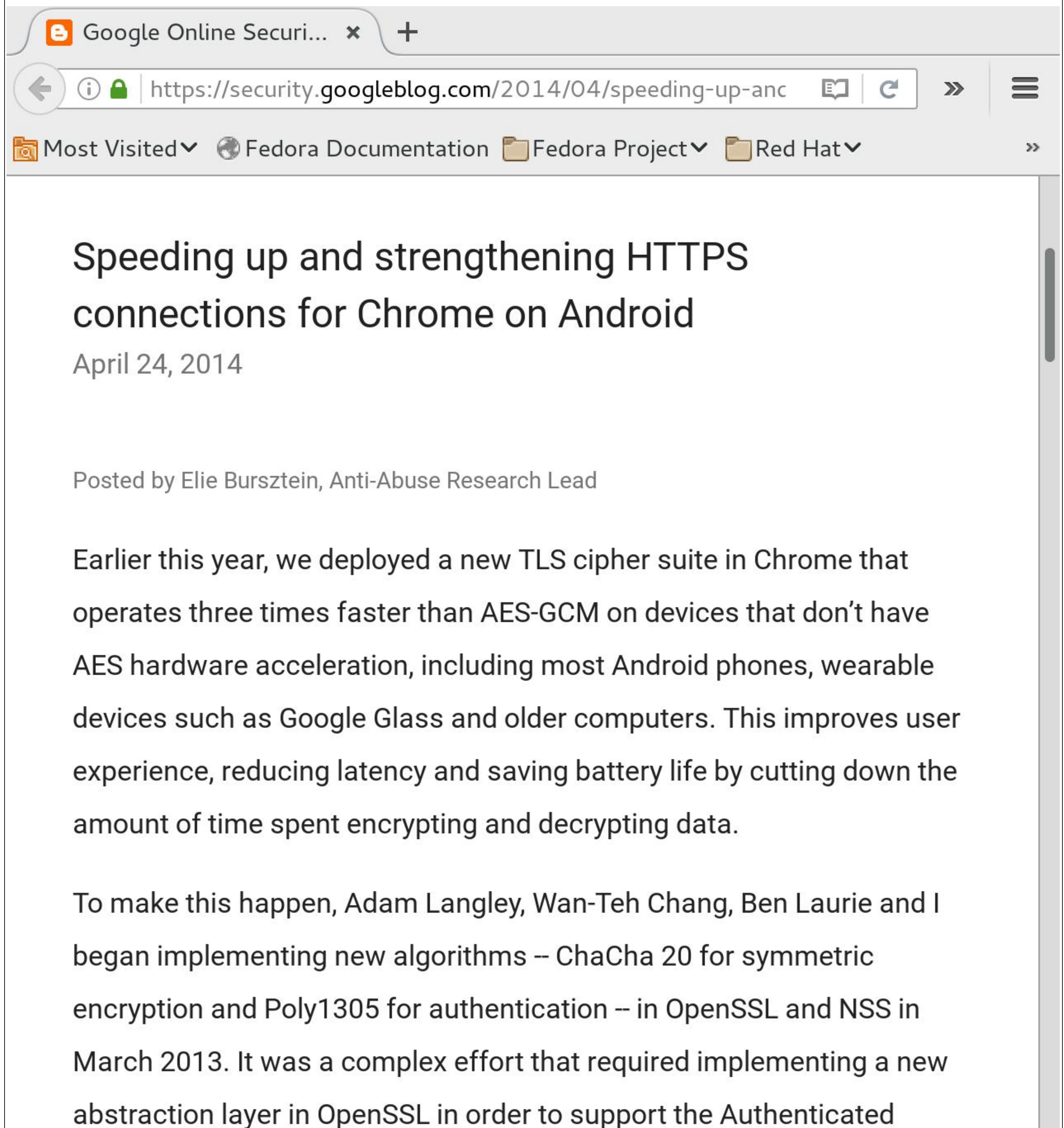
speed reported
for a 3.4GHz
e i7-6700):

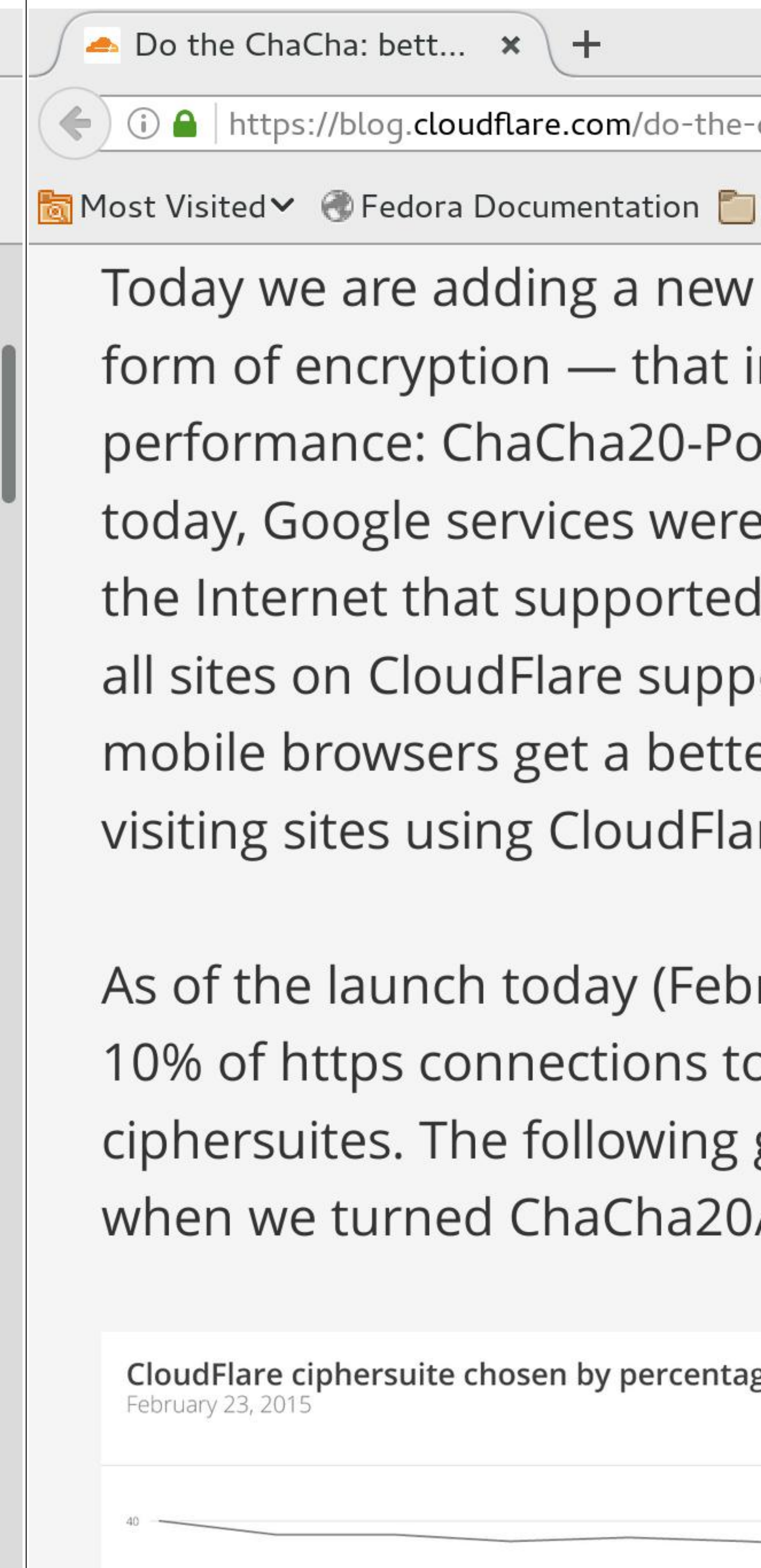
c (8.86
ES CTR-DRBG
5.07 MByte/sec
or ChaCha20.

reports
or AES-256-CTR,
or ChaCha20.

nse: “essential
standard open
Slow ones?





rted
GHz
:

RBG
e/sec
20.

6-CTR,
20.

ntial
pen
s?



Google Online Securi...

https://security.googleblog.com/2014/04/speeding-up-anc

Most Visited Fedora Documentation Fedora Project Red Hat

## Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated

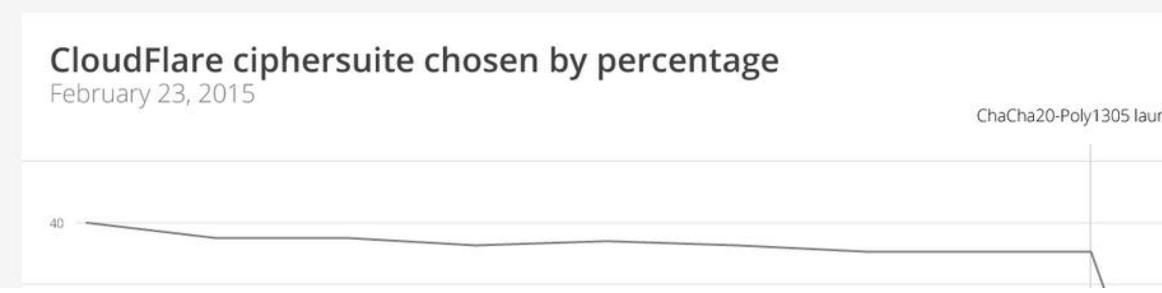
Do the ChaCha: bett...

https://blog.cloudflare.com/do-the-chacha-better-mobile-p

Most Visited Fedora Documentation Fedora Project Red

Today we are adding a new feature — actu
form of encryption — that improves mobil
performance: ChaCha20-Poly1305 cipher s
today, Google services were the only major
the Internet that supported this new algori
all sites on CloudFlare support it, too. This
mobile browsers get a better experience w
visiting sites using CloudFlare.

As of the launch today (February 23, 2015),
10% of https connections to CloudFlare use
ciphersuites. The following graph shows th
when we turned ChaCha20/Poly1305 on gl



## Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated





Today we are adding a new feature — actually a new form of encryption — that improves mobile performance: ChaCha20-Poly1305 cipher suites. Until today, Google services were the only major sites on the Internet that supported this new algorithm. Now all sites on CloudFlare support it, too. This means mobile browsers get a better experience when visiting sites using CloudFlare.

As of the launch today (February 23, 2015), nearly 10% of https connections to CloudFlare use the new ciphersuites. The following graph shows the uptick when we turned ChaCha20/Poly1305 on globally:

p and strengthening HTTPS
s for Chrome on Android

ztein, Anti-Abuse Research Lead

we deployed a new TLS cipher suite in Chrome that
imes faster than AES-GCM on devices that don't have
cceleration, including most Android phones, wearable
Google Glass and older computers. This improves user
ucing latency and saving battery life by cutting down the
spent encrypting and decrypting data.

ppen, Adam Langley, Wan-Teh Chang, Ben Laurie and I
nting new algorithms -- ChaCha 20 for symmetric
Poly1305 for authentication -- in OpenSSL and NSS in
was a complex effort that required implementing a new
r in OpenSSL in order to support the Authenticated







Today we are adding a new feature — actually a new form of encryption — that improves mobile performance: ChaCha20-Poly1305 cipher suites. Until today, Google services were the only major sites on the Internet that supported this new algorithm. Now all sites on CloudFlare support it, too. This means mobile browsers get a better experience when visiting sites using CloudFlare.

As of the launch today (February 23, 2015), nearly 10% of https connections to CloudFlare use the new ciphersuites. The following graph shows the uptick when we turned ChaCha20/Poly1305 on globally:






Maybe Skip SH

In 2005 and 2006,
[1][2][3]. These rep
as cryptographers
at all. After all, mar

In the wake of this
order to hedge t
"ket-chak", I believ
proved that we *do*
2005 didn't extend
hash functions, all
it existed, it was no
tendency to assum
ber is bigger.

As I've mentioned
It contributes to t
tested and harder
platforms typically
code-size, which is
even slower than
crypto primitives

14/04/speeding-up-anc

Fedora Project Red Hat

ening HTTPS
n Android

n Lead

TLS cipher suite in Chrome that
-GCM on devices that don't have
most Android phones, wearable
der computers. This improves user
ing battery life by cutting down the
decrypting data.

Wan-Teh Chang, Ben Laurie and I
-- ChaCha 20 for symmetric
cation -- in OpenSSL and NSS in
that required implementing a new
to support the Authenticated



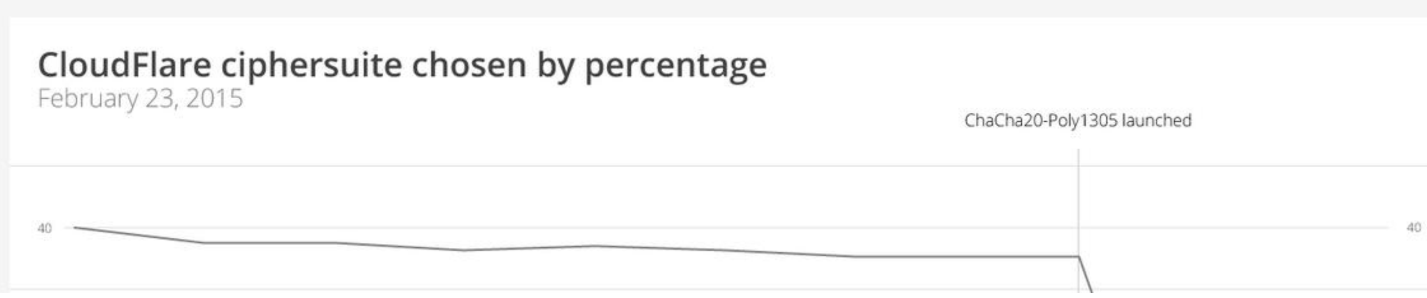
Do the ChaCha: bett...

https://blog.cloudflare.com/do-the-chacha-better-mobile-p

Most Visited Fedora Documentation Fedora Project Red Hat

Today we are adding a new feature — actually a new form of encryption — that improves mobile performance: ChaCha20-Poly1305 cipher suites. Until today, Google services were the only major sites on the Internet that supported this new algorithm. Now all sites on CloudFlare support it, too. This means mobile browsers get a better experience when visiting sites using CloudFlare.

As of the launch today (February 23, 2015), nearly 10% of https connections to CloudFlare use the new ciphersuites. The following graph shows the uptick when we turned ChaCha20/Poly1305 on globally:




ImperialViolet - Maybe S...

https://www.imperialviolet.org/201

Most Visited Fedora Documentation

Maybe Skip SHA-3 (31 May 2017

In 2005 and 2006, a series of significant
[1][2][3]. These repeated break-through
as cryptographers questioned whether
at all. After all, many hash functions fro

In the wake of this, NIST announced (P
order to hedge the risk of SHA-2 fa
"ket-chak", I believe) won (PDF) and bec
proved that we *do* know how to build h
2005 didn't extend to SHA-2 and the S
hash functions, all of which are secure a
it existed, it was no longer clear that SH
tendency to assume that SHA-3 must be
ber is bigger.

As I've mentioned before, diversity of c
It contributes to the exponential num
tested and hardened; it draws on lim
platforms typically need separate, op
code-size, which is a worry again in the
even slower than SHA-2 which is alre
crypto primitives

26

Hat

ome that
don't have
s, wearable
mproves user
ing down the

Laurie and I
metric
nd NSS in
enting a new
nticated

27

Do the ChaCha: bett...

https://blog.cloudflare.com/do-the-chacha-better-mobile-p

Most Visited Fedora Documentation Fedora Project Red Hat

Today we are adding a new feature — actually a new form of encryption — that improves mobile performance: ChaCha20-Poly1305 cipher suites. Until today, Google services were the only major sites on the Internet that supported this new algorithm. Now all sites on CloudFlare support it, too. This means mobile browsers get a better experience when visiting sites using CloudFlare.

As of the launch today (February 23, 2015), nearly 10% of https connections to CloudFlare use the new ciphersuites. The following graph shows the uptick when we turned ChaCha20/Poly1305 on globally:


ImperialViolet - Maybe S...

https://www.imperialviolet.org/2017/05/31/skipsha3.html

Most Visited Fedora Documentation Fedora Project Red

## Maybe Skip SHA-3 (31 May 2017)

In 2005 and 2006, a series of significant results were publish
[1][2][3]. These repeated break-throughs caused something
as cryptographers questioned whether we knew how to buil
at all. After all, many hash functions from the 1990's had not

In the wake of this, NIST announced (PDF) a competition to
order to hedge the risk of SHA-2 falling. In 2012, Kecc
"ket-chak", I believe) won (PDF) and became SHA-3. But the
proved that we *do* know how to build hash functions: the se
2005 didn't extend to SHA-2 and the SHA-3 process produ
hash functions, all of which are secure as far as we can tell.
it existed, it was no longer clear that SHA-3 was needed. Yet
tendency to assume that SHA-3 must be better than SHA-2 b
ber is bigger.

As I've mentioned before, diversity of cryptographic primiti
It contributes to the exponential number of combinations
tested and hardened; it draws on limited developer resou
platforms typically need separate, optimised code; and i
code-size, which is a worry again in the mobile age. SHA-3 is
even slower than SHA-2 which is already a comparative
crypto primitives

Do the ChaCha: bett... × +

https://blog.cloudflare.com/do-the-chacha-better-mobile-p

Most Visited Fedora Documentation Fedora Project Red Hat

Today we are adding a new feature — actually a new form of encryption — that improves mobile performance: ChaCha20-Poly1305 cipher suites. Until today, Google services were the only major sites on the Internet that supported this new algorithm. Now all sites on CloudFlare support it, too. This means mobile browsers get a better experience when visiting sites using CloudFlare.

As of the launch today (February 23, 2015), nearly 10% of https connections to CloudFlare use the new ciphersuites. The following graph shows the uptick when we turned ChaCha20/Poly1305 on globally:




ImperialViolet - Maybe S... × +

https://www.imperialviolet.org/2017/05/31/skipsha3.html

Most Visited Fedora Documentation Fedora Project Red Hat

## Maybe Skip SHA-3 (31 May 2017)

In 2005 and 2006, a series of significant results were published against SHA-1 [1][2][3]. These repeated break-throughs caused something of a crisis of faith as cryptographers questioned whether we knew how to build hash functions at all. After all, many hash functions from the 1990's had not aged well [1][2].

In the wake of this, NIST announced (PDF) a competition to develop SHA-3 in order to hedge the risk of SHA-2 falling. In 2012, Keccak (pronounced "ket-chak", I believe) won (PDF) and became SHA-3. But the competition itself proved that we *do* know how to build hash functions: the series of results in 2005 didn't extend to SHA-2 and the SHA-3 process produced a number of hash functions, all of which are secure as far as we can tell. Thus, by the time it existed, it was no longer clear that SHA-3 was needed. Yet there is a natural tendency to assume that SHA-3 must be better than SHA-2 because the number is bigger.

As I've mentioned before, diversity of cryptographic primitives is expensive. It contributes to the exponential number of combinations that need to be tested and hardened; it draws on limited developer resources as multiple platforms typically need separate, optimised code; and it contributes to code-size, which is a worry again in the mobile age. SHA-3 is also slow, and is even slower than SHA-2 which is already a comparative laggard amongst crypto primitives.