Software-based side-channel attacks

ECC 2012

B. B. Brumley

bbb at qualcomm.com
Qualcomm Technologies, Inc.

28–31 Oct 2012

Acknowledgments

Onur Acıiçmez, Manuel Barbosa, Philipp Grabher, Risto Hakala, Dan Page, Nicola Tuveri, Frederik Vercauteren

Part I

Preliminaries

Agenda

Papers

B. B. Brumley, R. M. Hakala. Cache-Timing Template Attacks. ASIACRYPT 2009.

O. Acıiçmez, B. B. Brumley, P. Grabher. New Results on Instruction Cache Attacks. CHES 2010.

B. B. Brumley, N. Tuveri. Remote Timing Attacks are Still Practical. ESORICS 2011.

B. B. Brumley, M. Barbosa, D. Page, F. Vercauteren. Practical Realisation and Elimination of an ECC-Related Software Bug Attack. CT-RSA 2012.

CVEs

`CVE-2011-1945`

`CVE-2011-4354`

Previous work

P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. CRYPTO 1996.

N. Howgrave-Graham, N. P. Smart. Lattice Attacks on Digital Signature Schemes. Des. Codes Cryptography 2001.

D. Brumley, D. Boneh. Remote Timing Attacks are Practical. USENIX Security 2003.

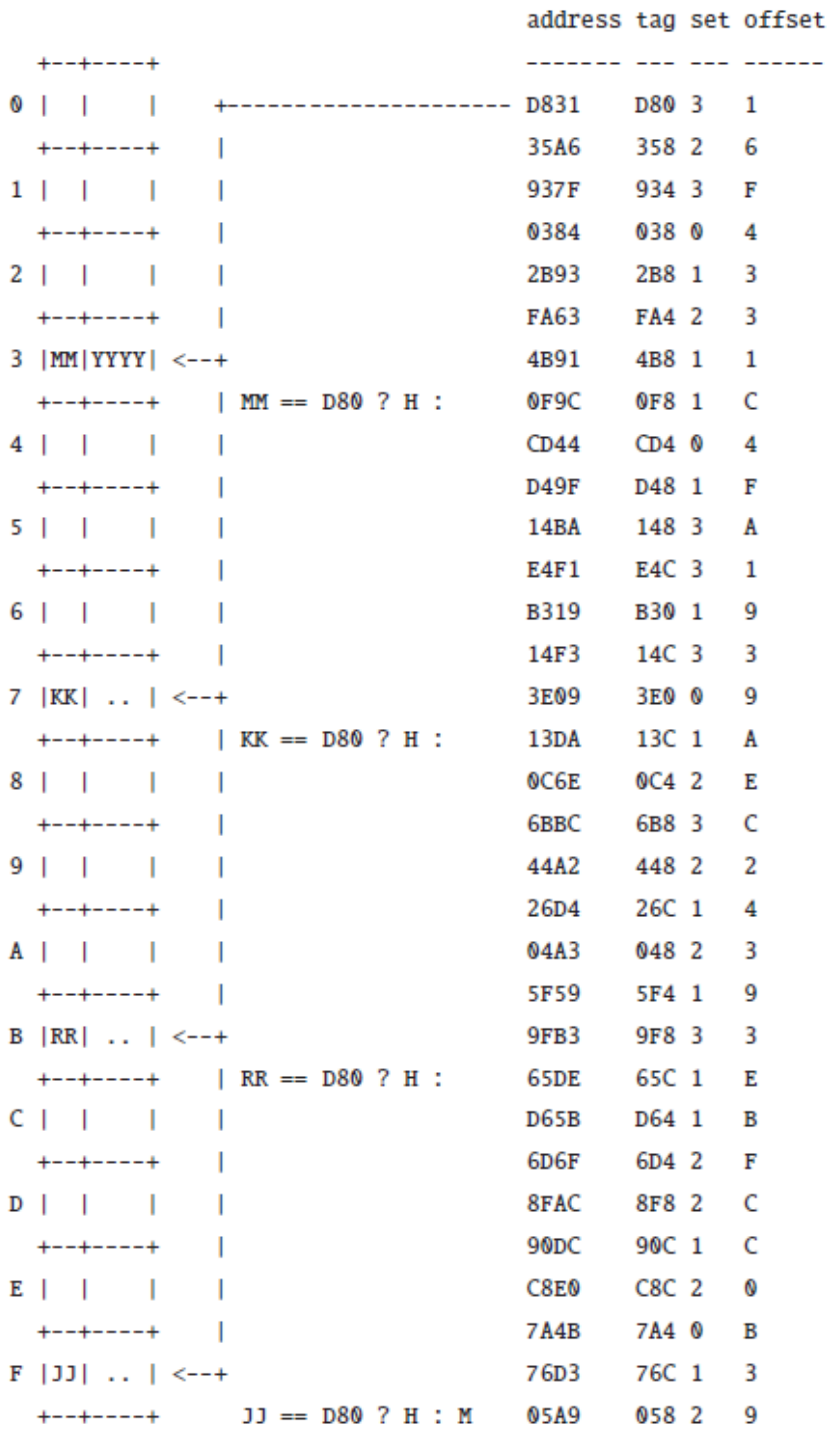
C. Percival. Cache Missing for Fun and Profit. BSDCan 2005.

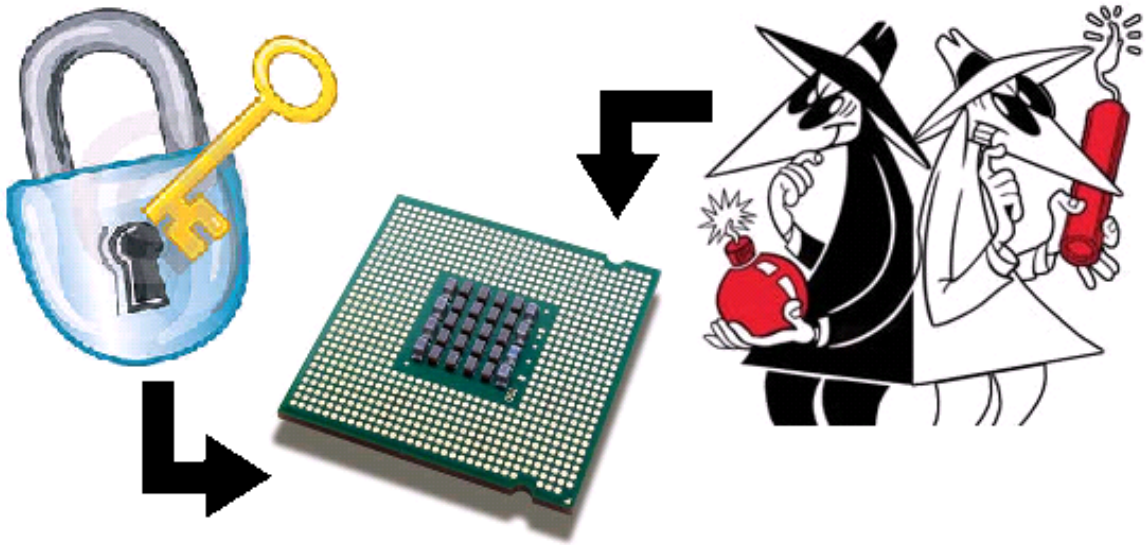
O. Acıiçmez. Yet another MicroArchitectural Attack: exploiting I-Cache. CSAW 2007.

E. Biham, Y. Carmeli, A. Shamir. Bug Attacks. CRYPTO 2008.

Part II

Caching 101


address tag set offset
0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
MM
YYYY
KK
RR
MM == D80 ? H :
KK == D80 ? H :
RR == D80 ? H :
JJ == D80 ? H : M
D831 D80 3 1
35A6 358 2 6
937F 934 3 F
0384 038 0 4
2B93 2B8 1 3
FA63 FA4 2 3
4B91 4B8 1 1
0F9C 0F8 1 C
CD44 CD4 0 4
D49F D48 1 F
14BA 148 3 A
E4F1 E4C 3 1
B319 B30 1 9
14F3 14C 3 3
3E09 3E0 0 9
13DA 13C 1 A
0C6E 0C4 2 E
6BBC 6B8 3 C
44A2 448 2 2
26D4 26C 1 4
04A3 048 2 3
5F59 5F4 1 9
9FB3 9F8 3 3
65DE 65C 1 E
D65B D64 1 B
6D6F 6D4 2 F
8FAC 8F8 2 C
90DC 90C 1 C
C8E0 C8C 2 0
7A4B 7A4 0 B
76D3 76C 1 3
05A9 058 2 9

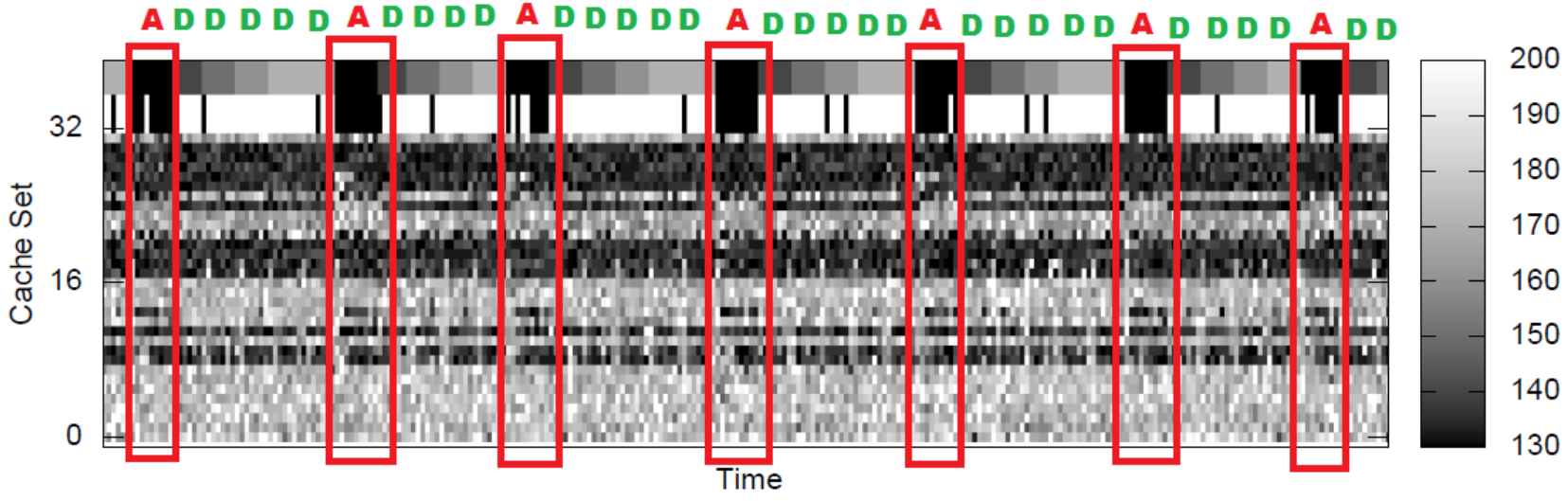


Part III

Data cache-timing attacks

```
655         for (k = max_len - 1; k >= 0; k--)
656             {
657             if (!r_is_at_infinity)
658                 {
659                 if (!EC_POINT_dbl(group, r, r, ctx)) goto err;
660                 }
661
662             for (i = 0; i < totalnum; i++)
663                 {
664                 if (wNAF_len[i] > (size_t)k)
665                     {
666                     int digit = wNAF[i][k];
667                     int is_neg;
668
669                     if (digit)
670                         {
671                         is_neg = digit < 0;
672
673                         if (is_neg)
674                             digit = -digit;
675
676                         if (is_neg != r_is_inverted)
677                             {
678                             if (!r_is_at_infinity)
679                                 {
680                                 if (!EC_POINT_invert(group, r, ctx)) goto err;
681                                 }
682                             r_is_inverted = !r_is_inverted;
683                             }
684
685                         /* digit > 0 */
686
687                         if (r_is_at_infinity)
688                             {
689                             if (!EC_POINT_copy(r, val_sub[i][digit >> 1])) goto err;
690                             r_is_at_infinity = 0;
691                             }
692                         else
693                             {
694                             if (!EC_POINT_add(group, r, r, val_sub[i][digit >> 1], ctx)) goto err;
```

```
xor %edi, %edi
rdtsc
mov %eax, %esi
LOOP:
; cache set 0
imul 0x0000(%ecx), %ecx
imul 0x0800(%ecx), %ecx
imul 0x1000(%ecx), %ecx
imul 0x1800(%ecx), %ecx
rdtsc
sub %esi, %eax
movb %al, 0x00(%ebx, %edi)
add %eax, %esi
; cache set 1
imul 0x0040(%ecx), %ecx
imul 0x0840(%ecx), %ecx
imul 0x1040(%ecx), %ecx
imul 0x1840(%ecx), %ecx
rdtsc
sub %esi, %eax
movb %al, 0x01(%ebx, %edi)
add %eax, %esi
; cache set 2
imul 0x0080(%ecx), %ecx
imul 0x0880(%ecx), %ecx
imul 0x1080(%ecx), %ecx
imul 0x1880(%ecx), %ecx
rdtsc
sub %esi, %eax
movb %al, 0x02(%ebx, %edi)
add %eax, %esi
...
; cache set 31
imul 0x07c0(%ecx), %ecx
imul 0x0fc0(%ecx), %ecx
imul 0x17c0(%ecx), %ecx
imul 0x1fc0(%ecx), %ecx
rdtsc
sub %esi, %eax
movb %al, 0x1f(%ebx, %edi)
add %eax, %esi
add $32, %edi
cmp <buffer len>, %edi
jge END
jmp LOOP
```



Attack effort

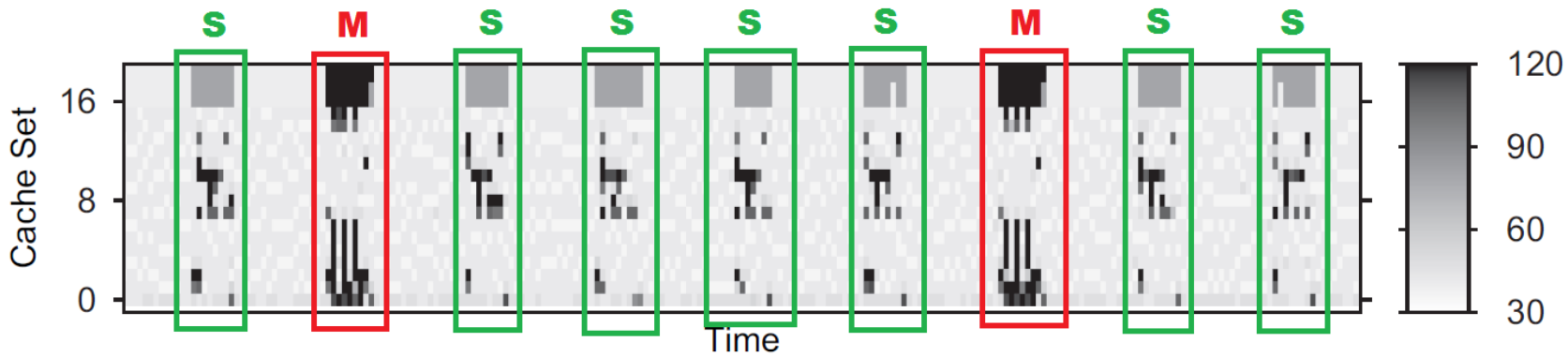
Intel Pentium 4, 8K signatures, 59min on a core2 duo.

Part IV

Instruction cache-timing attacks

```
129 int BN_mod_mul_montgomery(BIGNUM *r, const BIGNUM *a, const BIGNUM *b,
130                          BN_MONT_CTX *mont, BN_CTX *ctx)
131     {
132     BIGNUM *tmp;
133     int ret=0;
134 #if defined(OPENSSL_BN_ASM_MONT) && defined(MONT_WORD)
135     int num = mont->N.top;
136
137     if (num>1 && a->top==num && b->top==num)
138         {
139         if (bn_wexpand(r,num) == NULL) return(0);
140         if (bn_mul_mont(r->d,a->d,b->d,mont->N.d,mont->n0,num))
141             {
142             r->neg = a->neg^b->neg;
143             r->top = num;
144             bn_correct_top(r);
145             return(1);
146             }
147         }
148 #endif
149
150     BN_CTX_start(ctx);
151     tmp = BN_CTX_get(ctx);
152     if (tmp == NULL) goto err;
153
154     bn_check_top(tmp);
155     if (a == b)
156         {
157         if (!BN_sqr(tmp,a,ctx)) goto err;
158         }
159     else
160         {
161         if (!BN_mul(tmp,a,b,ctx)) goto err;
162         }
163     /* reduce from aRR to aR */
```

```
xor %edi, %edi
rdtsc
mov %eax, %esi
jmp L0
.align 4096
L0: ; cache set 0
  jmp L64
  .rept 59
  nop
  .endr
L1: ; cache set 1
  jmp L65
  .rept 59
  nop
  .endr
...
L64: ; cache set 0
  jmp L128
  .rept 59
  nop
  .endr
...
L448: ; cache set 0
  rdtsc
  sub %esi, %eax
  movb %al, (%ebx, %edi)
  add %eax, %esi
  inc %edi
  jmp L1
  .rept 49
  nop
  .endr
...
L511: ; cache set 63
  rdtsc
  sub %esi, %eax
  movb %al, (%ebx, %edi)
  add %eax, %esi
  inc %edi
  cmp <buffer len>, %edi
  jge END
  jmp L0
```



Attack effort

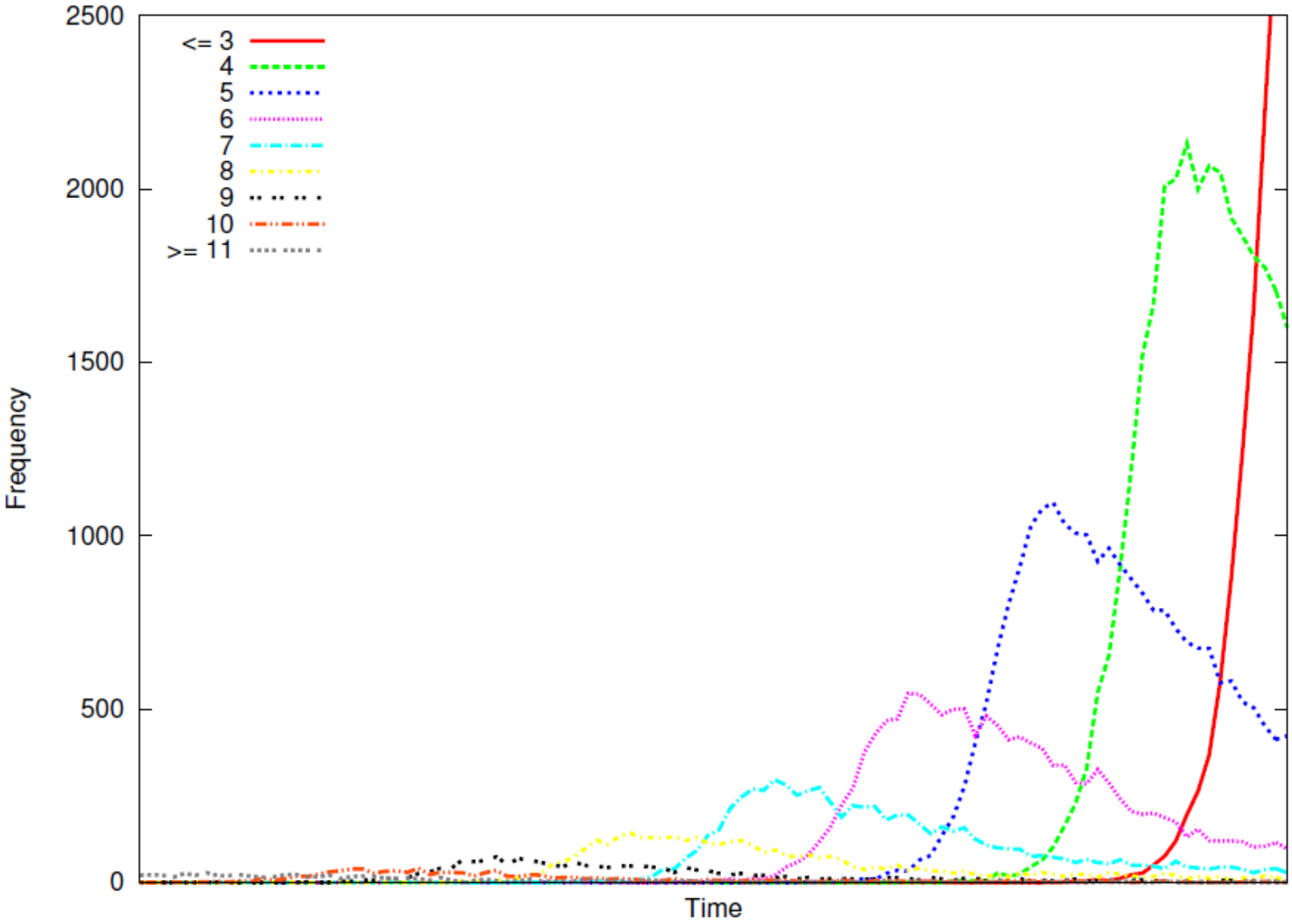
Intel Atom, 17K signatures and traces, 54min on 5 core2 quad cores.

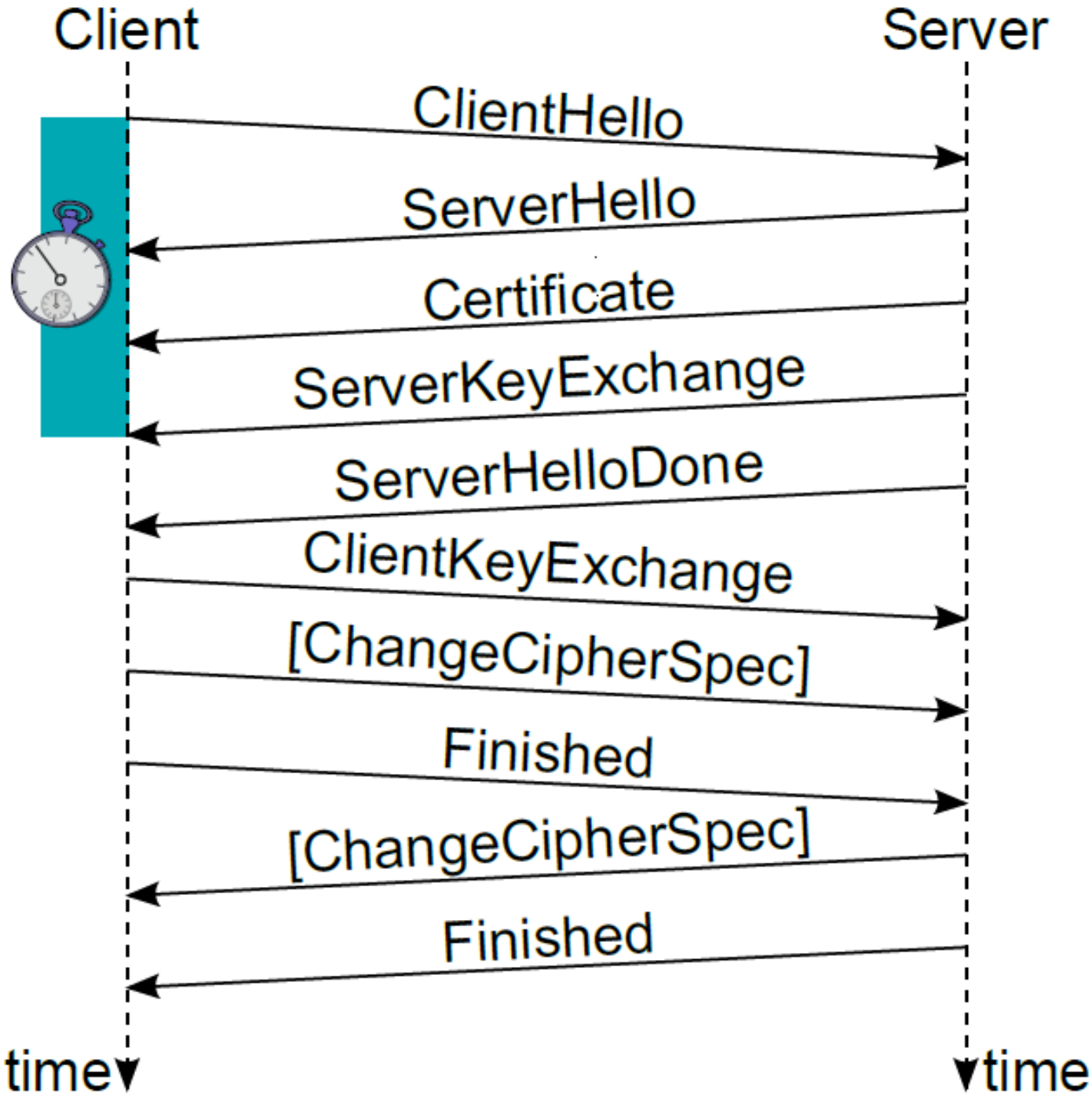
```
257     /* Compute r = (g^k mod p) mod q */
258
259     if ((dsa->flags & DSA_FLAG_NO_EXP_CONSTTIME) == 0)
260         {
261         if (!BN_copy(&kq, &k)) goto err;
262
263         /* We do not want timing information to leak the length of k,
264          * so we compute g^k using an equivalent exponent of fixed length.
265          *
266          * (This is a kludge that we need because the BN_mod_exp_mont()
267          * does not let us specify the desired timing behaviour.) */
268
269         if (!BN_add(&kq, &kq, dsa->q)) goto err;
270         if (BN_num_bits(&kq) <= BN_num_bits(dsa->q))
271             {
272             if (!BN_add(&kq, &kq, dsa->q)) goto err;
273             }
274
275         K = &kq;
276         }
```

Part V

Remote timing attacks

```
255     /* find top most bit and go one past it */
256     i = scalar->top - 1;
257     mask = BN_TBIT;
258     word = scalar->d[i];
259     while (!(word & mask)) mask >>= 1;
260     mask >>= 1;
261     /* if top most bit was at word break, go to next word */
262     if (!mask)
263         {
264         i--;
265         mask = BN_TBIT;
266         }
267
268     for (; i >= 0; i--)
269         {
270         word = scalar->d[i];
271         while (mask)
272             {
273             if (word & mask)
274                 {
275                 if (!gf2m_Madd(group, &point->X, x1, z1, x2, z2, ctx)) goto err;
276                 if (!gf2m_Mdouble(group, x2, z2, ctx)) goto err;
277                 }
278             else
279                 {
280                 if (!gf2m_Madd(group, &point->X, x2, z2, x1, z1, ctx)) goto err;
281                 if (!gf2m_Mdouble(group, x1, z1, ctx)) goto err;
282                 }
283             mask >>= 1;
284             }
285         mask = BN_TBIT;
```


2500
2000
1500
1000
500
0
Frequency
Time
<= 3
4
5
6
7
8
9
10
>= 11


Client
Server
ClientHello
ServerHello
Certificate
ServerKeyExchange
ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished
[ChangeCipherSpec]
Finished
time
time

Collected signatures count	4096	8192	16384
Filtered set size	64	64	64
Average “false positives” count	17.06	4.01	0.90

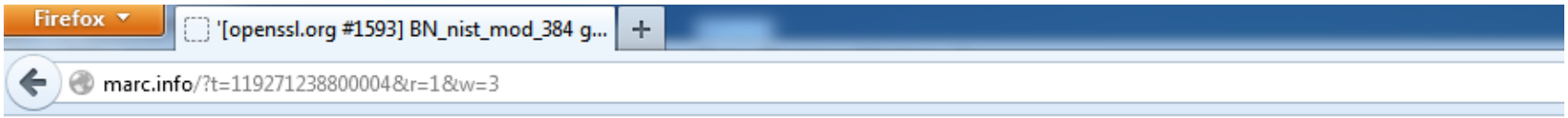
Table: Messages exchanged over localhost loopback interface.

Collected signatures count	4096	8192	16384
Filtered set size	64	64	64
Average “false positives” count	19.40	8.96	11.81

Table: Messages exchanged between two hosts on a switched network segment.

Part VI

Bug attacks



Note: The system is in read-onl

Search: Search Reset

[◉] Subjects [○] Authors [ ] Bodies (must pick a list first)

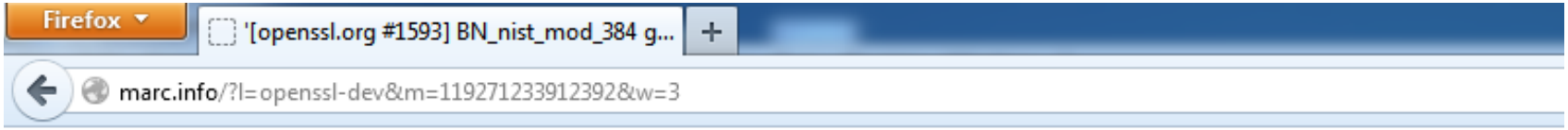
Set Page Width: [80] [90] [100] [120]

Viewing messages in thread '[openssl.org #1593] BN_nist_mod_384 gives wrong answers'

openssl-dev Developers list for the OpenSSL Project

2012-10-01 - 2012-11-01 (118 messages)

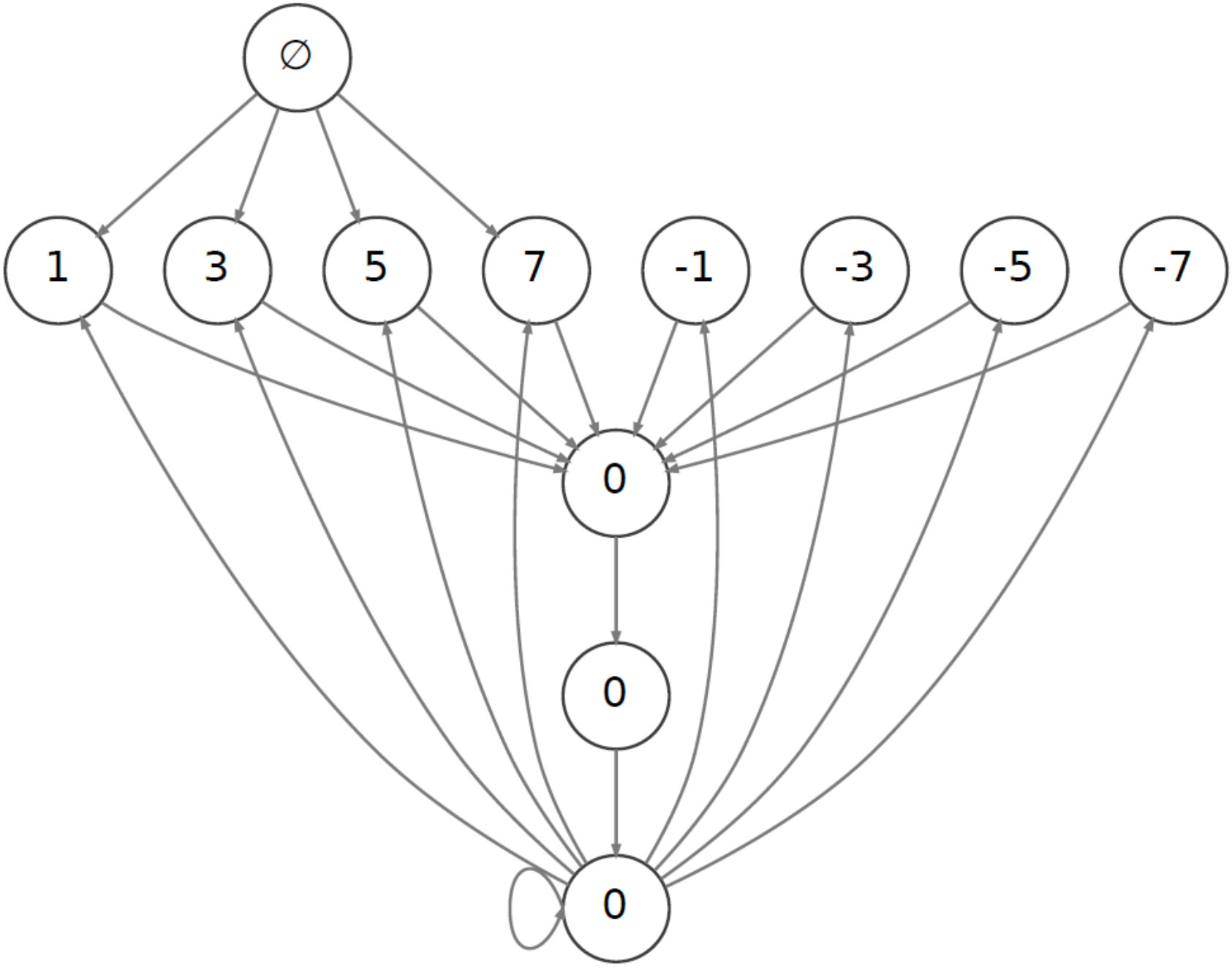
1.	2008-04-23	Re: [openssl.org #1593] BN nist mod 384 gives wrong answer	openssl-dev	Harry Reimann via RT
2.	2008-04-23	Re: [openssl.org #1593] BN nist mod 384 gives wrong answer	openssl-dev	Andy Polyakov via RT
3.	2008-04-09	Re: [openssl.org #1593] BN nist mod 384 gives wrong answer	openssl-dev	Andy Polyakov
4.	2008-04-08	Re: [openssl.org #1593] BN nist mod 384 gives wrong answer	openssl-dev	Harry Reimann via RT
5.	2008-04-01	Re: [openssl.org #1593] BN nist mod 384 gives wrong answer	openssl-dev	Andy Polyakov via RT
6.	2007-10-18	[openssl.org #1593] BN nist mod 384 gives wrong answers	openssl-dev	Harry Reimann via RT

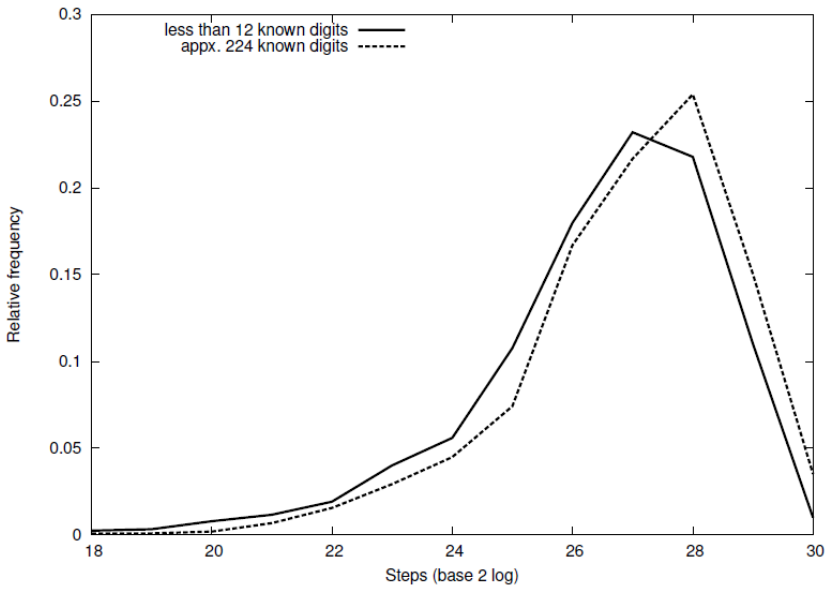


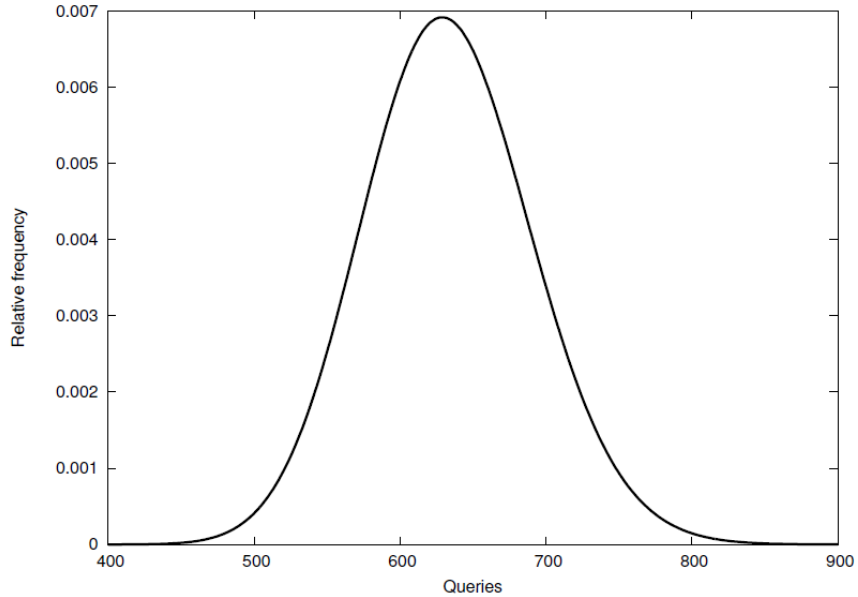
List: openssl-dev
Subject: [openssl.org #1593] BN_nist_mod_384 gives wrong answers
From: "Harry Reimann via RT" <rt () openssl ! org>
Date: 2007-10-18 12:58:48
Message-ID: rt-3.4.5-97698-1192712328-18.1593-21-0 () openssl ! org
[Download message RAW]

```
The function BN_nist_mod_384 (in crypto/bn/bn_nist.c) gives wrong results
for some inputs. For example, on input:
0xffffffff00000000ffffffffffffffffffffffff000000000000000000000000ffffffffffffffffffffffffffffffff \
fe00000002ffffffffffffffffffffffff00000000ffffffffffffffffffffffff it yields
0x100000000ffffffffffffffff00000000
but the correct result is
0x200000001ffffffffffffffe00000001.

As a consequence the function EC_POINT_add gives sometimes wrong results
for the NIST standard curve P-384. For example, if it computes the sum of
the points
(0x56fce068ab4eacbcbdca2a8cf5608e74d89ad30925bedac917aee82799eab18cd22e09a64bdcc2e03fd6f51a7c23b \
ce6,
0x259b376c88fc35a48e25dc20da307cf2ca30cda69f14584020a75061b0a300f04c6acd9c8890a9653625bdaed2d2e4 \
ce) and
(0xa9031f9754b153444235d5730a9f718b27652cf6da412536e85117d866154e722dd1f658b4233d1fc0290ae683dc4 \
31a,
0x9ac68504a9cb1565ea7b9ccf45d53bbbcc06c39cb102a3ef5d3b7f02111489e7dde3bb954e8a10bf4b9c6f852dd2c4 \
6a) which lie on that curve the result is a point which does not lie on the
curve. To see this compile the following code:

---------------- BEGIN: demo.c ----------------------------------------

#include <stdio.h>
```

```
530     /*D4*/
531     nist_set_256(t_d, buf, 13, 0, 11, 10, 9, 0, 15, 14);
532     if (bn_sub_words(r_d, r_d, t_d, BN_NIST_256_TOP))
533         --carry;
534
535     if (carry)
536         {
537         if (carry > 0)
538             bn_sub_words(r_d, r_d, _256_data + BN_NIST_256_TOP *
539                 --carry, BN_NIST_256_TOP);
540         else
541             {
542             carry = -carry;
543             bn_add_words(r_d, r_d, _256_data + BN_NIST_256_TOP *
544                 --carry, BN_NIST_256_TOP);
545             }
546         }
547
548     r->top = BN_NIST_256_TOP;
549     bn_correct_top(r);
550     if (BN_ucmp(r, field) >= 0)
551         {
552         bn_sub_words(r_d, r_d, _nist_p_256, BN_NIST_256_TOP);
553         bn_correct_top(r);
554         }
555     bn_check_top(r);
```


∅
1
3
5
7
-1
-3
-5
-7
0
0
0


less than 12 known digits
appx. 224 known digits
Relative frequency
Steps (base 2 log)
0.3
0.25
0.2
0.15
0.1
0.05
0
18
20
22
24
26
28
30


Relative frequency
Queries
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
400
500
600
700
800
900

Trivia

ECDHE-ECDSA-AES128-SHA

vs

ECDH-ECDSA-AES128-SHA

Part VII

Conclusion


Graduate Texts in Mathematics
Neal Koblitz
A Course in Number Theory and Cryptography
Second Edition
Springer