Non-uniform cracks in the concrete: the power of free precomputation

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

Tanja Lange
Technische Universiteit Eindhoven

---

eprint.iacr.org/2012/318,
eprint.iacr.org/2012/458

2012.02.19 Koblitz–Menezes "Another look at HMAC":

"... *Third, we describe a fundamental flaw in Bellare's 2006 security proof for HMAC, and show that with the flaw removed the proof gives a security guarantee that is of little value in practice.*"

2012.03.02: "*Bellare contacted us and told us that he strongly objected to our language—especially the word 'flaw'—...*"

form
n the concrete:
er of free precomputation

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

ange
che Universiteit Eindhoven

.iacr.org/2012/318,
.iacr.org/2012/458

2012.02.19 Koblitz–Menezes
"Another look at HMAC":

"*. . . Third, we describe a fundamental flaw in Bellare's 2006 security proof for HMAC, and show that with the flaw removed the proof gives a security guarantee that is of little value in practice.*"

2012.03.02: "*Bellare contacted us and told us that he strongly objected to our language—especially the word 'flaw'—. . .*"

Yehuda
really ou
there is
the proo
uniform
to not b
is NO F

Jonatha
research
concerne
Alfred M
an invite
2012 rel
criticizin
I share t

rete:
precomputation
n
is at Chicago &
siteit Eindhoven

siteit Eindhoven

---

g/2012/318,
g/2012/458

2012.02.19 Koblitz–Menezes
"Another look at HMAC":

"... *Third, we describe a fundamental flaw in Bellare's 2006 security proof for HMAC, and show that with the flaw removed the proof gives a security guarantee that is of little value in practice.*"

2012.03.02: "*Bellare contacted us and told us that he strongly objected to our language—especially the word 'flaw'—...*"

Yehuda Lindell: "
really outdid them
there is actually n
the proof of securi
uniform model, wh
to not be familiar
is NO FLAW here

Jonathan Katz: "
researchers are jus
concerned about t
Alfred Menezes wi
an invited talk at
2012 related to his
criticizing provable
I share this concer

tation

ago &
hoven

hoven

18,
58

2012.02.19 Koblitz–Menezes "Another look at HMAC":

"... *Third, we describe a fundamental flaw in Bellare's 2006 security proof for HMAC, and show that with the flaw removed the proof gives a security guarantee that is of little value in practice.*"

2012.03.02: "*Bellare contacted us and told us that he strongly objected to our language—especially the word 'flaw'—...*"

Yehuda Lindell: "*This time really outdid themselves sinc there is actually no error. R the proof of security is in th uniform model, which they a to not be familiar with. ... is NO FLAW here whatsoev*

Jonathan Katz: "*Many researchers are justifiably concerned about the fact th Alfred Menezes will be givin an invited talk at Eurocrypt 2012 related to his line of pa criticizing provable security. I share this concern.*"

2012.02.19 Koblitz–Menezes "Another look at HMAC":

"... *Third, we describe a fundamental flaw in Bellare's 2006 security proof for HMAC, and show that with the flaw removed the proof gives a security guarantee that is of little value in practice.*"

2012.03.02: "*Bellare contacted us and told us that he strongly objected to our language—especially the word 'flaw'—...*"

Yehuda Lindell: "*This time they really outdid themselves since there is actually no error. Rather the proof of security is in the non-uniform model, which they appear to not be familiar with. ... There is NO FLAW here whatsoever.*"

Jonathan Katz: "*Many researchers are justifiably concerned about the fact that Alfred Menezes will be giving an invited talk at Eurocrypt 2012 related to his line of papers criticizing provable security. I share this concern.*"

.19 Koblitz–Menezes
er look at HMAC":

rd, we describe a
ental flaw in Bellare's
curity proof for HMAC,
w that with the
noved the proof gives
ty guarantee that is of
lue in practice."

.02: "*Bellare contacted*
old us that he strongly
l to our language—
ly the word 'flaw'—. . ."

Yehuda Lindell: "*This time they really outdid themselves since there is actually no error. Rather the proof of security is in the non-uniform model, which they appear to not be familiar with. . . . There is NO FLAW here whatsoever.*"

Jonathan Katz: "*Many researchers are justifiably concerned about the fact that Alfred Menezes will be giving an invited talk at Eurocrypt 2012 related to his line of papers criticizing provable security. I share this concern.*"

Bellare t
to 2012.
never oc
reader w
when co
have no
want . . .
theoretic
would b
our feed
informed
the field
uniform
taught i
computa

z–Menezes
HMAC":

cribe a
in Bellare's
of for HMAC,
h the
proof gives
ee that is of
tice."

are contacted
t he strongly
nguage—
d 'flaw'—..."

Yehuda Lindell: "*This time they really outdid themselves since there is actually no error. Rather the proof of security is in the non-uniform model, which they appear to not be familiar with. ... There is NO FLAW here whatsoever.*"

Jonathan Katz: "*Many researchers are justifiably concerned about the fact that Alfred Menezes will be giving an invited talk at Eurocrypt 2012 related to his line of papers criticizing provable security. I share this concern.*"

Bellare to Koblitz
to 2012.10 Koblitz
never occurred to
reader would not
when complexity i
have non-uniformi
want ... to gain
theoretical cryptog
would benefit fron
our feedback and
informed about th
the field. ... Unif
uniform complexit
taught in a gradua
computational co

s

s
AC,

s
of

cted
gly

. . ."

Yehuda Lindell: "*This time they really outdid themselves since there is actually no error. Rather the proof of security is in the non-uniform model, which they appear to not be familiar with. . . . There is NO FLAW here whatsoever.*"

Jonathan Katz: "*Many researchers are justifiably concerned about the fact that Alfred Menezes will be giving an invited talk at Eurocrypt 2012 related to his line of papers criticizing provable security. I share this concern.*"

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*I never occurred to me that a reader would not understand when complexity is concrete have non-uniformity. . . . If y want . . . to gain respect am theoretical cryptographers, i would benefit from reflecting our feedback and being bett informed about the basics o the field. . . . Uniform and n uniform complexity are typic taught in a graduate course computational complexity th

Yehuda Lindell: "*This time they really outdid themselves since there is actually no error. Rather the proof of security is in the non-uniform model, which they appear to not be familiar with. . . . There is NO FLAW here whatsoever.*"

Jonathan Katz: "*Many researchers are justifiably concerned about the fact that Alfred Menezes will be giving an invited talk at Eurocrypt 2012 related to his line of papers criticizing provable security. I share this concern.*"

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

Lindell: "*This time they utdid themselves since actually no error. Rather of of security is in the non- model, which they appear e familiar with. . . . There LAW here whatsoever.*"

n Katz: "*Many ers are justifiably ed about the fact that Menezes will be giving ed talk at Eurocrypt ated to his line of papers g provable security. this concern.*"

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

2012.03.
"*. . . Thi*
fundame
practice-
Bellare's
HMAC,
defect re
a securit
little val

This time they
nselves since
o error. Rather
ity is in the non-
hich they appear
with. . . . There
whatsoever."

Many
stifiably
he fact that
ill be giving
Eurocrypt
s line of papers
e security.
rn."

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

2012.03.17 Koblitz
"*. . . Third, we des*
fundamental defec
practice-oriented s
Bellare's 2006 sec
HMAC, and show
defect removed his
a security guarant
little value in prac

they
ce
ather
e non-
appear
There
er."

at
g

apers

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

2012.03.17 Koblitz–Menezes
"*. . . Third, we describe a fundamental defect from a practice-oriented standpoint Bellare's 2006 security result HMAC, and show that with defect removed his proof giv a security guarantee that is little value in practice.*"

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

2012.03.17 Koblitz–Menezes: "*. . . Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

2012.03.17 Koblitz–Menezes: "*. . . Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" $\Rightarrow$ $>$20 solid seconds of applause.

Bellare to Koblitz (according to 2012.10 Koblitz talk): "*It never occurred to me that a reader would not understand that when complexity is concrete, we have non-uniformity. . . . If you want . . . to gain respect among theoretical cryptographers, it would benefit from reflecting our feedback and being better informed about the basics of the field. . . . Uniform and non-uniform complexity are typically taught in a graduate course in computational complexity theory.*"

2012.03.17 Koblitz–Menezes: "*. . . Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" ⇒ >20 solid seconds of applause. youtube?v=156ORg5xXkk

to Koblitz (according
10 Koblitz talk): "*It*
ccurred to me that a
would not understand that
mplexity is concrete, we
n-uniformity. . . . If you
to gain respect among
cal cryptographers, it
enefit from reflecting
lback and being better
d about the basics of
. . . . Uniform and non-
complexity are typically
n a graduate course in
ational complexity theory."

2012.03.17 Koblitz–Menezes:
"*. . . Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" ⇒ >20 solid seconds of applause.
youtube?v=156ORg5xXkk

<u>Understa</u>

What is
AES-128

Attack i
that con
and com

Attack c

Standard
minimize

(according
z talk): "*It
me that a
understand that
s concrete, we
ty. . . . If you
respect among
graphers, it
n reflecting
being better
e basics of
orm and non-
y are typically
ate course in
mplexity theory.*"

2012.03.17 Koblitz–Menezes:
"*. . . Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" ⇒ >20 solid seconds of applause.
youtube?v=156ORg5xXkk

<u>Understanding the</u>

What is the best c
AES-128 key-recov

Attack input: a bl
that contains a se
and computes $p \mapsto$

Attack output: k.

Standard definitio
minimize "time".

g
t
d that
, we
ou
nong
t
g
ter
f
on-
cally
in
heory."

2012.03.17 Koblitz–Menezes:
"... *Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" ⇒ >20 solid seconds of applause.
youtube?v=156ORg5xXkk

Understanding the dispute

What is the best chosen-pla
AES-128 key-recovery attac

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$

Attack output: k.

Standard definition of "best" minimize "time".

2012.03.17 Koblitz–Menezes:
"... *Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" ⇒ >20 solid seconds of applause.
youtube?v=1560Rg5xXkk

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

2012.03.17 Koblitz–Menezes:
"... *Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice.*"

2012.04: Menezes gives Eurocrypt invited talk "Another look at provable security" ⇒ >20 solid seconds of applause.
youtube?v=1560Rg5xXkk

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with <100% success probability; analyze tradeoffs between "time" and success probability.

.17 Koblitz–Menezes:
rd, we describe a
ental defect from a
-oriented standpoint in
s 2006 security result for
and show that with this
emoved his proof gives
ty guarantee that is of
ue in practice."

: Menezes gives
pt invited talk "Another
provable security" ⇒
d seconds of applause.
e?v=156ORg5xXkk

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with <100% success probability; analyze tradeoffs between "time" and success probability.

Maybe a
could be
AES-CB
Should A
be worri

z–Menezes:
cribe a
ct from a
standpoint in
urity result for
that with this
s proof gives
ee that is of
tice."

gives
talk "Another
ecurity" ⇒
of applause.
Rg5xXkk

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with <100% success probability; analyze tradeoffs between "time" and success probability.

Maybe a key-recov
could be turned in
AES-CBC-MAC fo
Should AES-CBC-
be worried about t

s:

in
t for
this
ves
of

ther
>
se.

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with $<100\%$ success probability; analyze tradeoffs between "time" and success probability.

Maybe a key-recovery attack could be turned into an AES-CBC-MAC forgery atta Should AES-CBC-MAC user be worried about this?

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with $<100\%$ success probability; analyze tradeoffs between "time" and success probability.

Maybe a key-recovery attack could be turned into an AES-CBC-MAC forgery attack! Should AES-CBC-MAC users be worried about this?

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with $<100\%$ success probability; analyze tradeoffs between "time" and success probability.

Maybe a key-recovery attack could be turned into an AES-CBC-MAC forgery attack! Should AES-CBC-MAC users be worried about this?

No. Many researchers have tried and failed to find good AES key-recovery attacks.

Understanding the dispute

What is the best chosen-plaintext AES-128 key-recovery attack?

Attack input: a black box that contains a secret key k and computes $p \mapsto \mathsf{AES}_k(p)$.

Attack output: k.

Standard definition of "best": minimize "time".

More generally, allow attacks with $<100\%$ success probability; analyze tradeoffs between "time" and success probability.

Maybe a key-recovery attack could be turned into an AES-CBC-MAC forgery attack! Should AES-CBC-MAC users be worried about this?

No. Many researchers have tried and failed to find good AES key-recovery attacks.

Standard conjecture: For each $p \in [0, 1]$, each AES key-recovery attack with success probability $\geq p$ takes "time" $\geq 2^{128} p$.

See, e.g., 2005 Bellare–Rogaway.

anding the dispute

the best chosen-plaintext
3 key-recovery attack?

nput: a black box
tains a secret key k
nputes $p \mapsto \mathsf{AES}_k(p)$.

output: k.

d definition of "best":
e "time".

nerally, allow attacks with
success probability;
tradeoffs between
and success probability.

Maybe a key-recovery attack
could be turned into an
AES-CBC-MAC forgery attack!
Should AES-CBC-MAC users
be worried about this?

No. Many researchers
have tried and failed to find good
AES key-recovery attacks.

Standard conjecture:
For each $p \in [0,1]$,
each AES key-recovery attack
with success probability $\geq p$
takes "time" $\geq 2^{128} p$.

See, e.g., 2005 Bellare–Rogaway.

Interlude

How mu
following

```
def pic
  if nC
    if
      i
      if
      r
  if n1
    if
    ret
if n2
retur
```

e dispute

chosen-plaintext
very attack?

ack box
cret key k
$\mapsto \mathrm{AES}_k(p)$.

n of "best":

ow attacks with
obability;
between
ss probability.

Maybe a key-recovery attack
could be turned into an
AES-CBC-MAC forgery attack!
Should AES-CBC-MAC users
be worried about this?

No. Many researchers
have tried and failed to find good
AES key-recovery attacks.

Standard conjecture:
For each $p \in [0,1]$,
each AES key-recovery attack
with success probability $\geq p$
takes "time" $\geq 2^{128} p$.

See, e.g., 2005 Bellare–Rogaway.

Interlude regarding

How much "time"
following algorithm

```
def pidigit(n0,
  if n0 == 0:
    if n1 == 0:
      if n2 ==
      return
    if n2 == 0:
    return
  if n1 == 0:
    if n2 == 0:
    return
  if n2 == 0: r
  return
```

intext
k?

).

":

s with

ity.

Maybe a key-recovery attack
could be turned into an
AES-CBC-MAC forgery attack!
Should AES-CBC-MAC users
be worried about this?

No. Many researchers
have tried and failed to find good
AES key-recovery attacks.

Standard conjecture:
For each $p \in [0,1]$,
each AES key-recovery attack
with success probability $\geq p$
takes "time" $\geq 2^{128} p$.

See, e.g., 2005 Bellare–Rogaway.

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: retu
      return
    if n2 == 0: return
    return
  if n1 == 0:
    if n2 == 0: return
    return
  if n2 == 0: return
  return
```

Maybe a key-recovery attack
could be turned into an
AES-CBC-MAC forgery attack!
Should AES-CBC-MAC users
be worried about this?

No. Many researchers
have tried and failed to find good
AES key-recovery attacks.

Standard conjecture:
For each $p \in [0,1]$,
each AES key-recovery attack
with success probability $\geq p$
takes "time" $\geq 2^{128} p$.

See, e.g., 2005 Bellare–Rogaway.

Interlude regarding "time"

How much "time" does the
following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return             1
    if n2 == 0: return   4
    return               1
  if n1 == 0:
    if n2 == 0: return   5
    return               9
  if n2 == 0: return     2
  return                 6
```

a key-recovery attack
e turned into an
C-MAC forgery attack!
AES-CBC-MAC users
ed about this?

ny researchers
ed and failed to find good
y-recovery attacks.

d conjecture:
$p \in [0, 1]$,
S key-recovery attack
ccess probability $\geq p$
ime" $\geq 2^{128} p$.

., 2005 Bellare–Rogaway.

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return                1
    if n2 == 0: return      4
    return                  1
  if n1 == 0:
    if n2 == 0: return      5
    return                  9
  if n2 == 0: return        2
  return                    6
```

Students
learn to
Skipped

This alg

very attack
to an
orgery attack!
MAC users
this?

hers
ed to find good
attacks.

re:
,
overy attack
ability $\geq p$
^{28}p.

llare–Rogaway.

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return                1
    if n2 == 0: return      4
    return                  1
  if n1 == 0:
    if n2 == 0: return      5
    return                  9
  if n2 == 0: return        2
  return                    6
```

Students in algorit
learn to count exe
Skipped branches

This algorithm use

k

ck!

s

good

ck

away.

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return             1
    if n2 == 0: return   4
    return             1
  if n1 == 0:
    if n2 == 0: return   5
    return             9
  if n2 == 0: return     2
  return               6
```

Students in algorithm course
learn to count executed "ste
Skipped branches take 0 "st

This algorithm uses 4 "steps

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return             1
    if n2 == 0: return   4
    return             1
if n1 == 0:
  if n2 == 0: return   5
  return             9
if n2 == 0: return     2
return               6
```

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return             1
    if n2 == 0: return   4
    return             1
if n1 == 0:
  if n2 == 0: return   5
  return             9
if n2 == 0: return     2
return               6
```

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k + 1$ "steps".

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return             1
    if n2 == 0: return   4
    return               1
if n1 == 0:
  if n2 == 0: return     5
  return                 9
if n2 == 0: return       2
return                   6
```

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k + 1$ "steps".

Variant: There exists a 256-"step" AES key-recovery attack (with 100% success probability).

Interlude regarding "time"

How much "time" does the following algorithm take?

```
def pidigit(n0,n1,n2):
  if n0 == 0:
    if n1 == 0:
      if n2 == 0: return 3
      return             1
    if n2 == 0: return   4
    return             1
if n1 == 0:
  if n2 == 0: return   5
  return             9
if n2 == 0: return     2
return               6
```

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k+1$ "steps".

Variant: There exists a 256-"step" AES key-recovery attack (with 100% success probability). If "time" means "steps" then the standard conjecture is wrong.

e regarding "time"

ch "time" does the
g algorithm take?

```
digit(n0,n1,n2):
0 == 0:
 n1 == 0:
if n2 == 0: return 3
return                  1
 n2 == 0: return        4
turn                    1
1 == 0:
 n2 == 0: return        5
turn                    9
2 == 0: return          2
rn                      6
```

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k + 1$ "steps".

Variant: There exists a 256-"step" AES key-recovery attack (with 100% success probability). If "time" means "steps" then the standard conjecture is wrong.

2000 Be
"*We fix*
Access M
model o
running
executio
of A's d
conventi
caused [
tables . . .
can thin
fixed bas
NAND g
means t

g "time"

does the
n take?

```
,n1,n2):

:
 0: return 3
           1
: return   4
           1


: return   5
           9
return     2
           6
```

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k + 1$ "steps".

Variant: There exists a 256-"step" AES key-recovery attack (with 100% success probability). If "time" means "steps" then the standard conjecture is wrong.

2000 Bellare–Kilia

"We fix some part
Access Machine (
model of computa
running time [mea
execution time plu
of A's description
convention elimina
caused [by] arbitra
tables . . . Alterna
can think of circui
fixed basis of gate
NAND gates . . .
means the circuit

rn 3
1
1
4
1
5
9
2
6

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k + 1$ "steps".

Variant: There exists a 256-"step" AES key-recovery attack (with 100% success probability). If "time" means "steps" then the standard conjecture is wrong.

2000 Bellare–Kilian–Rogawa

"*We fix some particular Ran Access Machine (RAM) as a model of computation. . . . A running time [means] A's ac execution time plus the leng of A's description . . . This convention eliminates patho caused [by] arbitrarily large tables . . . Alternatively, the can think of circuits over so fixed basis of gates, like 2-in NAND gates . . . now time s means the circuit size.*"

Students in algorithm courses learn to count executed "steps". Skipped branches take 0 "steps".

This algorithm uses 4 "steps".

Generalization: There exists an algorithm that, given $n < 2^k$, prints the nth digit of π using $k+1$ "steps".

Variant: There exists a 256-"step" AES key-recovery attack (with 100% success probability). If "time" means "steps" then the standard conjecture is wrong.

2000 Bellare–Kilian–Rogaway: "*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

s in algorithm courses
count executed "steps".
branches take 0 "steps".

orithm uses 4 "steps".

zation: There exists an
m that, given $n < 2^k$,
he nth digit of π
+ 1 "steps".

There exists a 256-
AES key-recovery attack
0% success probability).
" means "steps" then the
l conjecture is wrong.

2000 Bellare–Kilian–Rogaway:
"*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

Side con

1. Defin
Bellare–
flawed:

Paper co
security
interpret
false, giv

thm courses
cuted "steps".
take 0 "steps".

es 4 "steps".

here exists an
ven $n < 2^k$,
it of π
s".

ists a 256-
ecovery attack
ss probability).
steps" then the
re is wrong.

2000 Bellare–Kilian–Rogaway:
"*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

Side comments:

1. Definition from
Bellare–Kilian–Rog
flawed: failed to a

Paper conjectured
security bounds; a
interpretation of c
false, given paper'

es
eps".
eps".
s".
an
^{k}",
tack
lity).
n the
g.

2000 Bellare–Kilian–Rogaway:
"*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

Side comments:

1. Definition from Crypto 19
Bellare–Kilian–Rogaway was
flawed: failed to add length.

Paper conjectured "useful"
security bounds; any reasona
interpretation of conjecture
false, given paper's definitio

2000 Bellare–Kilian–Rogaway:
"*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2000 Bellare–Kilian–Rogaway:
"*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

2000 Bellare–Kilian–Rogaway:
"*We fix some particular Random Access Machine (RAM) as a model of computation. . . . A's running time [means] A's actual execution time plus the length of A's description . . . This convention eliminates pathologies caused [by] arbitrarily large lookup tables . . . Alternatively, the reader can think of circuits over some fixed basis of gates, like 2-input NAND gates . . . now time simply means the circuit size.*"

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

3. NAND definition is easier but breaks many theorems.

llare–Kilian–Rogaway:
some particular Random Machine (RAM) as a f computation. . . . A's time [means] A's actual n time plus the length escription . . . This ion eliminates pathologies by] arbitrarily large lookup . Alternatively, the reader k of circuits over some sis of gates, like 2-input gates . . . now time simply he circuit size."

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

3. NAND definition is easier but breaks many theorems.

<u>Reductio</u>

Another
Each AE
forgery a
probabili
takes "ti

n–Rogaway:
ticular Random
RAM) as a
tion. . . . A's
ns] A's actual
us the length
. . . This
ates pathologies
arily large lookup
tively, the reader
its over some
s, like 2-input
now time simply
size."

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

3. NAND definition is easier but breaks many theorems.

<u>Reductions</u>

Another standard
Each AES-CBC-M
forgery attack wit
probability $\geq p + q$
takes "time" $> 2^{12}$

y:
ndom
a
A's
ctual
rth

logies
lookup
reader
me
nput
simply

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

3. NAND definition is easier but breaks many theorems.

Reductions

Another standard conjecture
Each AES-CBC-MAC q-bloc
forgery attack with success
probability $\geq p + q(q-1)/2$
takes "time" $> 2^{128} p$.

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

3. NAND definition is easier but breaks many theorems.

Reductions

Another standard conjecture: Each AES-CBC-MAC q-block forgery attack with success probability $\geq p + q(q-1)/2^{129}$ takes "time" $> 2^{128} p$.

Side comments:

1. Definition from Crypto 1994 Bellare–Kilian–Rogaway was flawed: failed to add length.

Paper conjectured "useful" DES security bounds; any reasonable interpretation of conjecture was false, given paper's definition.

2. Many more subtle issues defining RAM "time": see 1990 van Emde Boas survey.

3. NAND definition is easier but breaks many theorems.

Reductions

Another standard conjecture: Each AES-CBC-MAC q-block forgery attack with success probability $\geq p + q(q-1)/2^{129}$ takes "time" $> 2^{128} p$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break AES-CBC-MAC? AES-CTR? AES-GCM? Other AES-based protocols? Far less attention than for key recovery.

mments:

ition from Crypto 1994
Kilian–Rogaway was
failed to add length.

onjectured "useful" DES
bounds; any reasonable
tation of conjecture was
ven paper's definition.

y more subtle issues
RAM "time": see
n Emde Boas survey.

D definition is easier
ks many theorems.

Reductions

Another standard conjecture: Each AES-CBC-MAC q-block forgery attack with success probability $\geq p + q(q-1)/2^{129}$ takes "time" $> 2^{128} p$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break AES-CBC-MAC? AES-CTR? AES-GCM? Other AES-based protocols? Far less attention than for key recovery.

Provable

Prove: i
an AES-
then the
an AES
with sim
and succ

Crypto 1994
gaway was
dd length.

"useful" DES
ny reasonable
onjecture was
s definition.

otle issues
me": see
oas survey.

on is easier
theorems.

Reductions

Another standard conjecture:
Each AES-CBC-MAC q-block
forgery attack with success
probability $\geq p + q(q-1)/2^{129}$
takes "time" $> 2^{128} p$.

Why should users have any
confidence in this conjecture?

How many researchers have really
tried to break AES-CBC-MAC?
AES-CTR? AES-GCM? Other
AES-based protocols? Far less
attention than for key recovery.

Provable security t

Prove: if there is
an AES-CBC-MAC
then there is
an AES key-recove
with similar "time"
and success proba

994

DES
able
was
n.

Reductions

Another standard conjecture: Each AES-CBC-MAC q-block forgery attack with success probability $\geq p + q(q-1)/2^{129}$ takes "time" $> 2^{128} p$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break AES-CBC-MAC? AES-CTR? AES-GCM? Other AES-based protocols? Far less attention than for key recovery.

Provable security to the reso

Prove: if there is an AES-CBC-MAC attack then there is an AES key-recovery attack with similar "time" and success probability.

Reductions

Another standard conjecture:
Each AES-CBC-MAC q-block
forgery attack with success
probability $\geq p + q(q-1)/2^{129}$
takes "time" $> 2^{128} p$.

Why should users have any
confidence in this conjecture?

How many researchers have really
tried to break AES-CBC-MAC?
AES-CTR? AES-GCM? Other
AES-based protocols? Far less
attention than for key recovery.

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Reductions

Another standard conjecture:
Each AES-CBC-MAC q-block
forgery attack with success
probability $\geq p + q(q-1)/2^{129}$
takes "time" $> 2^{128} p$.

Why should users have any
confidence in this conjecture?

How many researchers have really
tried to break AES-CBC-MAC?
AES-CTR? AES-GCM? Other
AES-based protocols? Far less
attention than for key recovery.

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing from key-recovery
attack to PRF distinguishing
attack allows a proof:
1994 Bellare–Kilian–Rogaway.

ons

standard conjecture:
ES-CBC-MAC q-block
attack with success
ity $\geq p + q(q-1)/2^{129}$
ime" $> 2^{128}p$.

ould users have any
ce in this conjecture?

ny researchers have really
break AES-CBC-MAC?
R? AES-GCM? Other
sed protocols? Far less
n than for key recovery.

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing from key-recovery
attack to PRF distinguishing
attack allows a proof:
1994 Bellare–Kilian–Rogaway.

Similar p
"provabl

Protocol
that har
(e.g., AE
security

After ex
maybe g
of P, an

conjecture:
AC q-block
h success
$(q-1)/2^{129}$
^{28}p.

have any
conjecture?

chers have really
S-CBC-MAC?
GCM? Other
ols? Far less
key recovery.

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing from key-recovery
attack to PRF distinguishing
attack allows a proof:
1994 Bellare–Kilian–Rogaway.

Similar pattern th
"provable security"

Protocol designers
that hardness of a
(e.g., AES PRF at
security of various

After extensive cry
maybe gain confid
of P, and hence in

e:
ck
129
e?
really
AC?
er
ess
ery.

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing from key-recovery
attack to PRF distinguishing
attack allows a proof:
1994 Bellare–Kilian–Rogaway.

Similar pattern throughout t
"provable security" literatur

Protocol designers (try to) p
that hardness of a problem
(e.g., AES PRF attacks) imp
security of various protocols

After extensive cryptanalysis
maybe gain confidence in ha
of P, and hence in security

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing from key-recovery
attack to PRF distinguishing
attack allows a proof:
1994 Bellare–Kilian–Rogaway.

Similar pattern throughout the
"provable security" literature.

Protocol designers (try to) prove
that hardness of a problem P
(e.g., AES PRF attacks) implies
security of various protocols Q.

After extensive cryptanalysis of P,
maybe gain confidence in hardness
of P, and hence in security of Q.

Provable security to the rescue!

Prove: if there is
an AES-CBC-MAC attack
then there is
an AES key-recovery attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing from key-recovery
attack to PRF distinguishing
attack allows a proof:
1994 Bellare–Kilian–Rogaway.

Similar pattern throughout the
"provable security" literature.

Protocol designers (try to) prove
that hardness of a problem P
(e.g., AES PRF attacks) implies
security of various protocols Q.

After extensive cryptanalysis of P,
maybe gain confidence in hardness
of P, and hence in security of Q.

Why not directly cryptanalyze Q?
Cryptanalysis is hard work: have
to focus on *a few* problems P.
Proofs scale to *many* protocols Q.

e security to the rescue!

f there is
CBC-MAC attack
ere is
key-recovery attack
ilar "time"
cess probability.

This turns out to be hard.
nging from key-recovery
o PRF distinguishing
llows a proof:
llare–Kilian–Rogaway.

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., AES PRF attacks) implies security of various protocols Q.

After extensive cryptanalysis of P, maybe gain confidence in hardness of P, and hence in security of Q.

Why not directly cryptanalyze Q? Cryptanalysis is hard work: have to focus on *a few* problems P. Proofs scale to *many* protocols Q.

The big

These c

Example
a fast A
with suc

to the rescue!

C attack

ery attack
"

bility.

out to be hard.
n key-recovery
tinguishing
oof:
n–Rogaway.

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., AES PRF attacks) implies security of various protocols Q.

After extensive cryptanalysis of P, maybe gain confidence in hardness of P, and hence in security of Q.

Why not directly cryptanalyze Q? Cryptanalysis is hard work: have to focus on *a few* problems P. Proofs scale to *many* protocols Q.

The big oops

These conjecture

Example: There e
a fast AES PRF a
with success proba

cue!

hard.
very
g

y.

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., AES PRF attacks) implies security of various protocols Q.

After extensive cryptanalysis of P, maybe gain confidence in hardness of P, and hence in security of Q.

Why not directly cryptanalyze Q? Cryptanalysis is hard work: have to focus on *a few* problems P. Proofs scale to *many* protocols Q.

The big oops

These conjectures are wro

Example: There exists a fast AES PRF attack with success probability $\geq 2^{-}$

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., AES PRF attacks) implies security of various protocols Q.

After extensive cryptanalysis of P, maybe gain confidence in hardness of P, and hence in security of Q.

Why not directly cryptanalyze Q? Cryptanalysis is hard work: have to focus on *a few* problems P. Proofs scale to *many* protocols Q.

The big oops

These conjectures are wrong.

Example: There exists a fast AES PRF attack with success probability $\geq 2^{-64}$.

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., AES PRF attacks) implies security of various protocols Q.

After extensive cryptanalysis of P, maybe gain confidence in hardness of P, and hence in security of Q.

Why not directly cryptanalyze Q? Cryptanalysis is hard work: have to focus on *a few* problems P. Proofs scale to *many* protocols Q.

The big oops

These conjectures are wrong.

Example: There exists a fast AES PRF attack with success probability $\geq 2^{-64}$.

Good candidate for attack: $\mathrm{MD5}_0(7, \mathrm{AES}_k(0), \mathrm{AES}_k(1)) = 1$ with probability $\geq 1/2 + 2^{-64}$; $\mathrm{MD5}_0(7, F(0), F(1)) = 1$ with probability $\leq 1/2$.

Here $\mathrm{MD5}_0(x) = \mathrm{bit}_0(\mathrm{MD5}(x))$.

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., AES PRF attacks) implies security of various protocols Q.

After extensive cryptanalysis of P, maybe gain confidence in hardness of P, and hence in security of Q.

Why not directly cryptanalyze Q? Cryptanalysis is hard work: have to focus on *a few* problems P. Proofs scale to *many* protocols Q.

The big oops

These conjectures are wrong.

Example: There exists a fast AES PRF attack with success probability $\geq 2^{-64}$.

Good candidate for attack: $\mathsf{MD5}_0(7, \mathsf{AES}_k(0), \mathsf{AES}_k(1)) = 1$ with probability $\geq 1/2 + 2^{-64}$; $\mathsf{MD5}_0(7, F(0), F(1)) = 1$ with probability $\leq 1/2$. Here $\mathsf{MD5}_0(x) = \mathsf{bit}_0(\mathsf{MD5}(x))$.

If this candidate doesn't work, replace 7 with 8 or 9 or ....

pattern throughout the
le security" literature.

l designers (try to) prove
dness of a problem P
ES PRF attacks) implies
of various protocols Q.

tensive cryptanalysis of P,
gain confidence in hardness
d hence in security of Q.

t directly cryptanalyze Q?
alysis is hard work: have
on *a few* problems P.
cale to *many* protocols Q.

The big oops

These conjectures are wrong.

Example: There exists
a fast AES PRF attack
with success probability $\geq 2^{-64}$.

Good candidate for attack:
$\mathsf{MD5}_0(7, \mathsf{AES}_k(0), \mathsf{AES}_k(1)) = 1$
with probability $\geq 1/2 + 2^{-64}$;
$\mathsf{MD5}_0(7, F(0), F(1)) = 1$
with probability $\leq 1/2$.
Here $\mathsf{MD5}_0(x) = \mathsf{bit}_0(\mathsf{MD5}(x))$.

If this candidate doesn't work,
replace 7 with 8 or 9 or ....

"*We onl*
for $p \geq$

roughout the
" literature.

s (try to) prove
problem P
ttacks) implies
protocols Q.

yptanalysis of P,
ence in hardness
n security of Q.

cryptanalyze Q?
ard work: have
problems P.
any protocols Q.

The big oops

These conjectures are wrong.

Example: There exists
a fast AES PRF attack
with success probability $\geq 2^{-64}$.

Good candidate for attack:
$\mathrm{MD5}_0(7, \mathrm{AES}_k(0), \mathrm{AES}_k(1)) = 1$
with probability $\geq 1/2 + 2^{-64}$;
$\mathrm{MD5}_0(7, F(0), F(1)) = 1$
with probability $\leq 1/2$.
Here $\mathrm{MD5}_0(x) = \mathrm{bit}_0(\mathrm{MD5}(x))$.

If this candidate doesn't work,
replace 7 with 8 or 9 or ....

"*We only meant t*
for $p \geq 2^{-40}$, *you*

the
e.

prove
P
plies
Q.

s of P,
ardness
of Q.

ze Q?
have
P.
cols Q.

The big oops

These conjectures are wrong.

Example: There exists
a fast AES PRF attack
with success probability $\geq 2^{-64}$.

Good candidate for attack:
$\mathrm{MD5}_0(7, \mathrm{AES}_k(0), \mathrm{AES}_k(1)) = 1$
with probability $\geq 1/2 + 2^{-64}$;
$\mathrm{MD5}_0(7, F(0), F(1)) = 1$
with probability $\leq 1/2$.
Here $\mathrm{MD5}_0(x) = \mathrm{bit}_0(\mathrm{MD5}(x))$.

If this candidate doesn't work,
replace 7 with 8 or 9 or ....

"*We only meant the conject*
for $p \geq 2^{-40}$, *you nitpicker.*"

The big oops

These conjectures are wrong.

Example: There exists
a fast AES PRF attack
with success probability $\geq 2^{-64}$.

Good candidate for attack:
$\mathsf{MD5}_0(7, \mathsf{AES}_k(0), \mathsf{AES}_k(1)) = 1$
with probability $\geq 1/2 + 2^{-64}$;
$\mathsf{MD5}_0(7, F(0), F(1)) = 1$
with probability $\leq 1/2$.
Here $\mathsf{MD5}_0(x) = \mathsf{bit}_0(\mathsf{MD5}(x))$.

If this candidate doesn't work,
replace 7 with 8 or 9 or ....

"*We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker.*"

The big oops

These conjectures are wrong.

Example: There exists
a fast AES PRF attack
with success probability $\geq 2^{-64}$.

Good candidate for attack:
$\mathrm{MD5}_0(7, \mathrm{AES}_k(0), \mathrm{AES}_k(1)) = 1$
with probability $\geq 1/2 + 2^{-64}$;
$\mathrm{MD5}_0(7, F(0), F(1)) = 1$
with probability $\leq 1/2$.
Here $\mathrm{MD5}_0(x) = \mathrm{bit}_0(\mathrm{MD5}(x))$.

If this candidate doesn't work,
replace 7 with 8 or 9 or ....

"*We only meant the conjectures for* $p \geq 2^{-40}$*, you nitpicker.*"

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The big oops

These conjectures are wrong.

Example: There exists
a fast AES PRF attack
with success probability $\geq 2^{-64}$.

Good candidate for attack:
$\mathsf{MD5}_0(7, \mathsf{AES}_k(0), \mathsf{AES}_k(1)) = 1$
with probability $\geq 1/2 + 2^{-64}$;
$\mathsf{MD5}_0(7, F(0), F(1)) = 1$
with probability $\leq 1/2$.
Here $\mathsf{MD5}_0(x) = \mathsf{bit}_0(\mathsf{MD5}(x))$.

If this candidate doesn't work,
replace 7 with 8 or 9 or ....

"*We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker.*"

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The attack algorithm:
iterate $k \mapsto \mathsf{AES}_k(0) \oplus 7$
2^{43} times, look up in
a size-2^{43} Hellman table;
iterate $k \mapsto \mathsf{AES}_k(0) \oplus 8$
2^{43} times, look up in
a size-2^{43} Hellman table; etc.

oops

conjectures are wrong.

e: There exists
ES PRF attack
ccess probability $\geq 2^{-64}$.

ndidate for attack:
$, \mathsf{AES}_k(0), \mathsf{AES}_k(1)) = 1$
bability $\geq 1/2 + 2^{-64}$;
$, F(0), F(1)) = 1$
bability $\leq 1/2$.
$\mathsf{D5}_0(x) = \mathsf{bit}_0(\mathsf{MD5}(x))$.

andidate doesn't work,
7 with 8 or 9 or ....

"*We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker.*"

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The attack algorithm:
iterate $k \mapsto \mathsf{AES}_k(0) \oplus 7$
2^{43} times, look up in
a size-2^{43} Hellman table;
iterate $k \mapsto \mathsf{AES}_k(0) \oplus 8$
2^{43} times, look up in
a size-2^{43} Hellman table; etc.

How abo

ECDL in
where P
ECDL o

es are wrong.

xists
ttack
ability $\geq 2^{-64}$.

or attack:
$, \mathsf{AES}_k(1)) = 1$
$1/2 + 2^{-64}$;
$1)) = 1$
$1/2$.
$\mathsf{bit}_0(\mathsf{MD5}(x))$.

oesn't work,
r 9 or ....

"We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker."

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The attack algorithm:
iterate $k \mapsto \mathsf{AES}_k(0) \oplus 7$
2^{43} times, look up in
a size-2^{43} Hellman table;
iterate $k \mapsto \mathsf{AES}_k(0) \oplus 8$
2^{43} times, look up in
a size-2^{43} Hellman table; etc.

How about NIST

ECDL input: poin
where P is a stand
ECDL output: log

ong.

$^{-64}$.

$) = 1$
64.
;

$(x))$.

rk,

"*We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker.*"

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The attack algorithm:
iterate $k \mapsto \mathrm{AES}_k(0) \oplus 7$
2^{43} times, look up in
a size-2^{43} Hellman table;
iterate $k \mapsto \mathrm{AES}_k(0) \oplus 8$
2^{43} times, look up in
a size-2^{43} Hellman table; etc.

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard gener
ECDL output: $\log_P Q$.

"*We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker.*"

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The attack algorithm:
iterate $k \mapsto \mathrm{AES}_k(0) \oplus 7$
2^{43} times, look up in
a size-2^{43} Hellman table;
iterate $k \mapsto \mathrm{AES}_k(0) \oplus 8$
2^{43} times, look up in
a size-2^{43} Hellman table; etc.

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard generator.
ECDL output: $\log_P Q$.

"*We only meant the conjectures for $p \geq 2^{-40}$, you nitpicker.*"

The conjectures are still wrong!
Example: There exists
an AES key-recovery attack
with success probability ≈ 1
taking "time" $\approx 2^{86}$.

The attack algorithm:
iterate $k \mapsto \mathrm{AES}_k(0) \oplus 7$
2^{43} times, look up in
a size-2^{43} Hellman table;
iterate $k \mapsto \mathrm{AES}_k(0) \oplus 8$
2^{43} times, look up in
a size-2^{43} Hellman table; etc.

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard generator.
ECDL output: $\log_P Q$.

Standard conjecture:
For each $p \in [0, 1]$,
each P-256 ECDL algorithm
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

ly meant the conjectures
2^{-40}*, you nitpicker.*"

jectures are still wrong!
e: There exists
key-recovery attack
ccess probability ≈1
time" $\approx 2^{86}$.

ack algorithm:
$c \mapsto \mathrm{AES}_k(0) \oplus 7$
es, look up in
43 Hellman table;
$c \mapsto \mathrm{AES}_k(0) \oplus 8$
es, look up in
43 Hellman table; etc.

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard generator.
ECDL output: $\log_P Q$.

Standard conjecture:
For each $p \in [0, 1]$,
each P-256 ECDL algorithm
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

Cube-ro

Assumin
overwhe
compute

There ex
algorithm
and has

"Time"

Inescapa
standar
P-256 E
ECDSA

he conjectures

nitpicker."

re still wrong!

xists

ery attack

ability ≈ 1

^{36}.

hm:

$(0) \oplus 7$

in

table;

$(0) \oplus 8$

in

table; etc.

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard generator.
ECDL output: $\log_P Q$.

Standard conjecture:
For each $p \in [0, 1]$,
each P-256 ECDL algorithm
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

Cube-root ECDL a

Assuming plausible
overwhelmingly ve
computer experim

There exists a P-2
algorithm that tak
and has success pr

"Time" includes a

Inescapable conclu
standard conject
P-256 ECDL hard
ECDSA security, e

tures
,

ong!

c.

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard generator.
ECDL output: $\log_P Q$.

Standard conjecture:
For each $p \in [0, 1]$,
each P-256 ECDL algorithm
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

Cube-root ECDL algorithms

Assuming plausible heuristic
overwhelmingly verified by
computer experiment:

There exists a P-256 ECDL
algorithm that takes "time"
and has success probability $\approx$

"Time" includes algorithm l

Inescapable conclusion: **The**
standard conjectures (rega
P-256 ECDL hardness, P-25
ECDSA security, etc.) **are fa**

How about NIST P-256?

ECDL input: points P, Q,
where P is a standard generator.
ECDL output: $\log_P Q$.

Standard conjecture:
For each $p \in [0, 1]$,
each P-256 ECDL algorithm
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

Cube-root ECDL algorithms

Assuming plausible heuristics,
overwhelmingly verified by
computer experiment:

There exists a P-256 ECDL
algorithm that takes "time" $\approx 2^{85}$
and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

out NIST P-256?

nput: points P, Q,
' is a standard generator.
utput: $\log_P Q$.

d conjecture:
n $p \in [0, 1]$,
256 ECDL algorithm
ccess probability $\geq p$
ime" $\geq 2^{128} p^{1/2}$.

Cube-root ECDL algorithms

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" $\approx 2^{85}$ and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

Should P
be worri
P-256 E

No!

We have
that prin
but B ta

We conj
nobody

P-256?

ts P, Q,
dard generator.
$_P\, Q$.

re:
,
algorithm
ability $\geq p$
$^{28} p^{1/2}$.

Cube-root ECDL algorithms

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" $\approx 2^{85}$ and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

Should P-256 ECD
be worried about t
P-256 ECDL algor

No!

We have a progra
that prints out A,
but B takes "time

We conjecture tha
nobody will ever p

ator.

Cube-root ECDL algorithms

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" $\approx 2^{85}$ and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

Should P-256 ECDSA users be worried about this P-256 ECDL algorithm A?

No!

We have a program B that prints out A, but B takes "time" $\approx 2^{170}$.

We conjecture that nobody will ever print out A

Cube-root ECDL algorithms

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" $\approx 2^{85}$ and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

Should P-256 ECDSA users be worried about this P-256 ECDL algorithm A?

No!

We have a program B that prints out A, but B takes "time" $\approx 2^{170}$.

We conjecture that nobody will ever print out A.

Cube-root ECDL algorithms

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" $\approx 2^{85}$ and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

Should P-256 ECDSA users be worried about this P-256 ECDL algorithm A?

No!

We have a program B that prints out A, but B takes "time" $\approx 2^{170}$.

We conjecture that nobody will ever print out A.

But A *exists*, and the standard conjecture doesn't see the 2^{170}.

ot ECDL algorithms

g plausible heuristics,
lmingly verified by
er experiment:

kists a P-256 ECDL
m that takes "time" $\approx 2^{85}$
success probability ≈ 1.

includes algorithm length.

able conclusion: **The**
d conjectures (regarding
CDL hardness, P-256
security, etc.) **are false.**

Should P-256 ECDSA users
be worried about this
P-256 ECDL algorithm A?

No!

We have a program B
that prints out A,
but B takes "time" $\approx 2^{170}$.

We conjecture that
nobody will ever print out A.

But A *exists*, and the standard
conjecture doesn't see the 2^{170}.

Cryptana

Common
a 2^{170} "
(indepen
a 2^{85} "n

For cryp
2^{170}, mu

For the s
definitio
The mai
much be

algorithms

e heuristics,
rified by
ent:

256 ECDL
kes "time" $\approx 2^{85}$
robability ≈ 1.

lgorithm length.

usion: **The**
ures (regarding
ness, P-256
etc.) **are false.**

Should P-256 ECDSA users
be worried about this
P-256 ECDL algorithm A?

No!

We have a program B
that prints out A,
but B takes "time" $\approx 2^{170}$.

We conjecture that
nobody will ever print out A.

But A *exists*, and the standard
conjecture doesn't see the 2^{170}.

Cryptanalysts *do* s

Common parlance
a 2^{170} "precomput
(independent of Q
a 2^{85} "main comp

For cryptanalysts:
2^{170}, much worse

For the standard s
definitions and co
The main computa
much better than

s,

$\approx 2^{85}$
≈ 1.

ength.

e
rding
6
alse.

Should P-256 ECDSA users
be worried about this
P-256 ECDL algorithm A?

No!

We have a program B
that prints out A,
but B takes "time" $\approx 2^{170}$.

We conjecture that
nobody will ever print out A.

But A *exists*, and the standard
conjecture doesn't see the 2^{170}.

Cryptanalysts *do* see the 2^{17}

Common parlance: We have
a 2^{170} "precomputation"
(independent of Q) followed
a 2^{85} "main computation".

For cryptanalysts: This cost
2^{170}, much worse than 2^{128}.

For the standard security
definitions and conjectures:
The main computation costs
much better than 2^{128}.

Should P-256 ECDSA users
be worried about this
P-256 ECDL algorithm A?

No!

We have a program B
that prints out A,
but B takes "time" $\approx 2^{170}$.

We conjecture that
nobody will ever print out A.

But A *exists*, and the standard
conjecture doesn't see the 2^{170}.

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have
a 2^{170} "precomputation"
(independent of Q) followed by
a 2^{85} "main computation".

For cryptanalysts: This costs
2^{170}, much worse than 2^{128}.

For the standard security
definitions and conjectures:
The main computation costs 2^{85},
much better than 2^{128}.

P-256 ECDSA users
ed about this
CDL algorithm A?

e a program B
nts out A,
akes "time" $\approx 2^{170}$.

ecture that
will ever print out A.

xists, and the standard
re doesn't see the 2^{170}.

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have
a 2^{170} "precomputation"
(independent of Q) followed by
a 2^{85} "main computation".

For cryptanalysts: This costs
2^{170}, much worse than 2^{128}.

For the standard security
definitions and conjectures:
The main computation costs 2^{85},
much better than 2^{128}.

What th

DSA users
this
rithm A?

m B

" $\approx 2^{170}$.

t
rint out A.

the standard
see the 2^{170}.

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have
a 2^{170} "precomputation"
(independent of Q) followed by
a 2^{85} "main computation".

For cryptanalysts: This costs
2^{170}, much worse than 2^{128}.

For the standard security
definitions and conjectures:
The main computation costs 2^{85},
much better than 2^{128}.

What the algorith

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have
a 2^{170} "precomputation"
(independent of Q) followed by
a 2^{85} "main computation".

For cryptanalysts: This costs
2^{170}, much worse than 2^{128}.

For the standard security
definitions and conjectures:
The main computation costs 2^{85},
much better than 2^{128}.

What the algorithm does

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have
a 2^{170} "precomputation"
(independent of Q) followed by
a 2^{85} "main computation".

For cryptanalysts: This costs
2^{170}, much worse than 2^{128}.

For the standard security
definitions and conjectures:
The main computation costs 2^{85},
much better than 2^{128}.

What the algorithm does

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have a 2^{170} "precomputation" (independent of Q) followed by a 2^{85} "main computation".

For cryptanalysts: This costs 2^{170}, much worse than 2^{128}.

For the standard security definitions and conjectures:
The main computation costs 2^{85}, much better than 2^{128}.

What the algorithm does

1999 Escott–Sager–Selkirk–Tsapakidis, also crediting Silverman–Stapleton:

Computing (e.g.) $\log_P Q_1$, $\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and $\log_P Q_5$ costs only 2.49× more than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho, *reusing* distinguished points produced by Q_1; etc.

alysts *do* see the 2^{170}.

n parlance: We have
precomputation"
ndent of Q) followed by
main computation".

tanalysts: This costs
uch worse than 2^{128}.

standard security
ns and conjectures:
n computation costs 2^{85},
etter than 2^{128}.

What the algorithm does

1999 Escott–Sager–Selkirk–
Tsapakidis, also crediting
Silverman–Stapleton:

Computing (e.g.) $\log_P Q_1$,
$\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and
$\log_P Q_5$ costs only 2.49× more
than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho,
reusing distinguished points
produced by Q_1; etc.

2001 Ku
cost $\Theta($
for n dis
in group
if $n \ll \ell$

see the 2^{170}.

: We have
tation"
') followed by
utation".

This costs
than 2^{128}.

ecurity
njectures:
ation costs 2^{85},
2^{128}.

What the algorithm does

1999 Escott–Sager–Selkirk–
Tsapakidis, also crediting
Silverman–Stapleton:

Computing (e.g.) $\log_P Q_1$,
$\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and
$\log_P Q_5$ costs only 2.49× more
than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho,
reusing distinguished points
produced by Q_1; etc.

2001 Kuhn–Struik
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete loga
in group of order ℓ
if $n \ll \ell^{1/4}$.

^{70}.

e

by

s

^{85},

What the algorithm does

1999 Escott–Sager–Selkirk–Tsapakidis, also crediting Silverman–Stapleton:

Computing (e.g.) $\log_P Q_1$, $\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and $\log_P Q_5$ costs only 2.49× more than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho,
reusing distinguished points
produced by Q_1; etc.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

What the algorithm does

1999 Escott–Sager–Selkirk–Tsapakidis, also crediting Silverman–Stapleton:

Computing (e.g.) $\log_P Q_1$, $\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and $\log_P Q_5$ costs only $2.49\times$ more than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho,
reusing distinguished points
produced by Q_1; etc.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

What the algorithm does

1999 Escott–Sager–Selkirk–Tsapakidis, also crediting Silverman–Stapleton:

Computing (e.g.) $\log_P Q_1$, $\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and $\log_P Q_5$ costs only 2.49× more than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho,
reusing distinguished points
produced by Q_1; etc.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2004 Hitchcock–Montague–Carter–Dawson:
View computations of
$\log_P Q_1, \ldots, \log_P Q_{n-1}$ as
precomputatation for main
computation of $\log_P Q_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

e algorithm does

cott–Sager–Selkirk–
dis, also crediting
nn–Stapleton:

ing (e.g.) $\log_P Q_1$,
, $\log_P Q_3$, $\log_P Q_4$, and
costs only 2.49× more
mputing $\log_P Q$.

ic idea:
e $\log_P Q_1$ with rho;
e $\log_P Q_2$ with rho,
distinguished points
d by Q_1; etc.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2004 Hitchcock–
Montague–Carter–Dawson:
View computations of
$\log_P Q_1, \ldots, \log_P Q_{n-1}$ as
precomputatation for main
computation of $\log_P Q_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

2012 Be
(1) Ada
insid
(2) Anal
main
prec
(3) Cho
more
main
(4) Also
func
(5) Redu
for e
(6) Brea

m does

r–Selkirk–
rediting
on:

$\log_P Q_1$,
$\log_P Q_4$, and
2.49× more
$\log_P Q$.

with rho;
with rho,
ed points
tc.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2004 Hitchcock–
Montague–Carter–Dawson:
View computations of
$\log_P Q_1, \ldots, \log_P Q_{n-1}$ as
precomputatation for main
computation of $\log_P Q_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

2012 Bernstein–La
(1) Adapt to inter
inside much la
(2) Analyze tradec
main-computa
precomputed t
(3) Choose table e
more carefully
main-computa
(4) Also choose it
function more
(5) Reduce space
for each table
(6) Break $\ell^{1/4}$ bar

and
ore

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2004 Hitchcock–
Montague–Carter–Dawson:
View computations of
$\log_P Q_1, \ldots, \log_P Q_{n-1}$ as
precomputatation for main
computation of $\log_P Q_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

2012 Bernstein–Lange:
(1) Adapt to interval of leng
inside much larger group
(2) Analyze tradeoffs betwee
main-computation time
precomputed table size.
(3) Choose table entries
more carefully to reduce
main-computation time.
(4) Also choose iteration
function more carefully.
(5) Reduce space required
for each table entry.
(6) Break $\ell^{1/4}$ barrier.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2004 Hitchcock–
Montague–Carter–Dawson:
View computations of
$\log_P Q_1, \ldots, \log_P Q_{n-1}$ as
precomputatation for main
computation of $\log_P Q_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

2012 Bernstein–Lange:

(1) Adapt to interval of length ℓ
inside much larger group.

(2) Analyze tradeoffs between
main-computation time and
precomputed table size.

(3) Choose table entries
more carefully to reduce
main-computation time.

(4) Also choose iteration
function more carefully.

(5) Reduce space required
for each table entry.

(6) Break $\ell^{1/4}$ barrier.

uhn–Struik analysis:
$n^{1/2}\ell^{1/2}$)
screte logarithms
of order ℓ
$^{1/4}$.

tchcock–
ue–Carter–Dawson:
mputations of
$\ldots, \log_P Q_{n-1}$ as
putatation for main
ation of $\log_P Q_n$.
tradeoffs between
mputation time and
putation time.

2012 Bernstein–Lange:

(1) Adapt to interval of length ℓ inside much larger group.

(2) Analyze tradeoffs between main-computation time and precomputed table size.

(3) Choose table entries more carefully to reduce main-computation time.

(4) Also choose iteration function more carefully.

(5) Reduce space required for each table entry.

(6) Break $\ell^{1/4}$ barrier.

Applicat

(7) Disp
P-25

(8) Acce

(9) Acce
this

Bonus:

(10) Dis
AE
sec

Credit to
paper fo

analysis:

rithms
ℓ

-Dawson:
s of
Q_{n-1} as
for main
$_P\, Q_n$.
between
time and
me.

2012 Bernstein–Lange:

(1) Adapt to interval of length ℓ inside much larger group.

(2) Analyze tradeoffs between main-computation time and precomputed table size.

(3) Choose table entries more carefully to reduce main-computation time.

(4) Also choose iteration function more carefully.

(5) Reduce space required for each table entry.

(6) Break $\ell^{1/4}$ barrier.

Applications:

(7) Disprove the s
P-256 security

(8) Accelerate trap

(9) Accelerate BG
this needs (1).

Bonus:

(10) Disprove the
AES, DSA-30
security conje

Credit to earlier L
paper for (2), (6),

2012 Bernstein–Lange:

(1) Adapt to interval of length ℓ inside much larger group.

(2) Analyze tradeoffs between main-computation time and precomputed table size.

(3) Choose table entries more carefully to reduce main-computation time.

(4) Also choose iteration function more carefully.

(5) Reduce space required for each table entry.

(6) Break $\ell^{1/4}$ barrier.

Applications:

(7) Disprove the standard 2[illegible] P-256 security conjectur

(8) Accelerate trapdoor DL

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard [illegible] AES, DSA-3072, RSA-[illegible] security conjectures.

Credit to earlier Lee–Cheon– paper for (2), (6), (8).

2012 Bernstein–Lange:

(1) Adapt to interval of length ℓ inside much larger group.

(2) Analyze tradeoffs between main-computation time and precomputed table size.

(3) Choose table entries more carefully to reduce main-computation time.

(4) Also choose iteration function more carefully.

(5) Reduce space required for each table entry.

(6) Break $\ell^{1/4}$ barrier.

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

rnstein–Lange:
pt to interval of length ℓ
e much larger group.
yze tradeoffs between
n-computation time and
omputed table size.
ose table entries
e carefully to reduce
n-computation time.
choose iteration
tion more carefully.
uce space required
each table entry.
ak $\ell^{1/4}$ barrier.

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard
choose u
$c_1, \ldots, c$
walk fro

ange:
val of length ℓ
rger group.
offs between
tion time and
table size.
entries
to reduce
tion time.
eration
carefully.
required
entry.
rrier.

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard walk fun
choose uniform ra
$c_1, \ldots, c_r \in \{1, 2,$
walk from R to R

gth ℓ
o.
en
and

Applications:

(7) Disprove the standard 2^{128}
P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.;
this needs (1).

Bonus:

(10) Disprove the standard 2^{128}
AES, DSA-3072, RSA-3072
security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$
walk from R to $R + c_{H(R)}P$

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.

Applications:

(7) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

(9) Accelerate BGN etc.; this needs (1).

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

ions:
rove the standard 2^{128}
56 security conjectures.
elerate trapdoor DL etc.
elerate BGN etc.;
needs (1).

prove the standard 2^{128}
S, DSA-3072, RSA-3072
urity conjectures.

o earlier Lee–Cheon–Hong
r (2), (6), (8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-30

Assume
is extend
to count

tandard 2^{128}
conjectures.
pdoor DL etc.
N etc.;

standard 2^{128}
072, RSA-3072
ectures.

ee–Cheon–Hong
(8).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-3072

Assume that DLP
is extended to 384
to counter previou

128

es.

etc.

2^{128}

3072

-Hong

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

Standard walk function:
choose uniform random
$c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$;
walk from R to $R + c_{H(R)}P$.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

d walk function:
uniform random
$c_r \in \{1, 2, \ldots, \ell - 1\}$;
m R to $R + c_{H(R)}P$.

putation:
me walks at yP
om choices of y.
ble of distinct
shed points D
th $\log_P D$.

mputation:
from Q, walk to
shed point $Q + yP$.
or $Q + yP$ in table.
ails, rerandomize Q.)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomp
Take y =
compute
for every

ction:
ndom
$\ldots, \ell - 1\}$;
$+ c_{H(R)} P$.

at yP
s of y.
inct
ts D
.
:
walk to
t $Q + yP$.
in table.
domize Q.)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-$
for every prime nu

};
?.

)

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq$

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \le y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\le y$.

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack
(and assume field $\mathbf{F}_p$ to avoid
Antoine coming after you).

The following sketch
is not the state of the art—
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \le y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\le y$.

If this fails, try again
with hg, hg^2, etc.

72

that DLP subgroup
ded to 384 bits
ter previous attack
sume field $\mathbf{F}_p$ to avoid
coming after you).

owing sketch
e state of the art—
d enough to break
assumption.

$\mathbf{F}_p^*$ have order q, $h = g^k$.
nd k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\leq y$.

If this fails, try again
with hg, hg^2, etc.

Analysis

About y
for a tot
to store

Can writ
probabili

h_i is y-s
very clos
where u

Overall t
between
iteration
detectio

subgroup
bits
s attack
$\mathbf{F}_p$ to avoid
fter you).

ch
the art—
to break
on.

rder q, $h = g^k$.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\leq y$.

If this fails, try again
with hg, hg^2, etc.

Analysis

About $y/\log y \approx 2^{1}$
for a total of $2^{109.}$
to store all small D

Can write h as h_1
probability $\approx (6/\pi^{2}$

h_i is y-smooth wi
very close to u^{-u}
where $u = 1535/1$

Overall the attack
between $2^{107.85}$ a
iterations; batch s
detection is fast.

id

$= g^k$.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\leq y$.

If this fails, try again
with hg, hg^2, etc.

Analysis

About $y/\log y \approx 2^{103.75}$ prim
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probabi
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\leq y$.

If this fails, try again
with hg, hg^2, etc.

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

putation:
$= 2^{110}$,
e $\log_g x^{(p-1)/q}$
y prime number $x \le y$.

mputation:
write h as
h_1/h_2 in $\mathbf{F}_p^*$
$\in \{1, 2, 3, \ldots, 2^{1535}\}$,
$^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
$\{h_1, h_2\} = 1$;

n try to factor h_1, h_2
nes $\le y$.

ails, try again
hg^2, etc.

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\le y$
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

Possible

$-1)/q$

mber $x \leq y$.

:

$\mathbf{F}_p^*$

$\ldots, 2^{1535}\}$,

$0, 1, \ldots, 2^{1535}\}$,

$= 1$;

ctor h_1, h_2

ain

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$ for a total of $2^{109.33}$ bytes to store all small DLs.

Can write h as h_1/h_2 with probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability very close to $u^{-u} \approx 2^{-53.06}$ where $u = 1535/110$.

Overall the attack requires between $2^{107.85}$ and $2^{108.85}$ iterations; batch smoothness detection is fast.

Possible responses

y.

}, 1535}},

2

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$ for a total of $2^{109.33}$ bytes to store all small DLs.

Can write h as h_1/h_2 with probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability very close to $u^{-u} \approx 2^{-53.06}$ where $u = 1535/110$.

Overall the attack requires between $2^{107.85}$ and $2^{108.85}$ iterations; batch smoothness detection is fast.

Possible responses

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

Possible responses

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$ for a total of $2^{109.33}$ bytes to store all small DLs.

Can write h as h_1/h_2 with probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability very close to $u^{-u} \approx 2^{-53.06}$ where $u = 1535/110$.

Overall the attack requires between $2^{107.85}$ and $2^{108.85}$ iterations; batch smoothness detection is fast.

Possible responses

(1) Accept 2^{85} etc. as security; live with it. Protect the proofs!

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

Possible responses

(1) Accept 2^{85} etc. as security;
live with it. Protect the proofs!

(2) Switch to NAND metric; or
(3) switch to AT metric.
Breaks most theorems;
still bogus results in NAND.

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

Possible responses

(1) Accept 2^{85} etc. as security;
live with it. Protect the proofs!

(2) Switch to NAND metric; or
(3) switch to AT metric.
Breaks most theorems;
still bogus results in NAND.

(4) Add effectivity. Include
cost for finding the algorithm.

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\leq y$
for a total of $2^{109.33}$ bytes
to store all small DLs.

Can write h as h_1/h_2 with
probability $\approx (6/\pi^2)2^{3071}/p$.

h_i is y-smooth with probability
very close to $u^{-u} \approx 2^{-53.06}$
where $u = 1535/110$.

Overall the attack requires
between $2^{107.85}$ and $2^{108.85}$
iterations; batch smoothness
detection is fast.

Possible responses

(1) Accept 2^{85} etc. as security;
live with it. Protect the proofs!

(2) Switch to NAND metric; or
(3) switch to AT metric.
Breaks most theorems;
still bogus results in NAND.

(4) Add effectivity. Include
cost for finding the algorithm.

(5) Add uniformity.
Clearly stops attacks
but breaks most theorems.
Abandons goal of defining concrete security of AES etc.