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

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

Tanja Lange
Technische Universiteit Eindhoven

Full 53-page paper,
including progress towards
formalizing collision resistance:
eprint.iacr.org/2012/318

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

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

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

Tanja Lange
Technische Universiteit Eindhoven

Full 53-page paper,
including progress towards
formalizing collision resistance:
eprint.iacr.org/2012/318

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with $<$100% success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

form
n the concrete:
er of free precomputation

ernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

ange
che Universiteit Eindhoven

page paper,
g progress towards
ing collision resistance:
.iacr.org/2012/318

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with <100% success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

P-256 di
total TL
Should T

rete:
precomputation

is at Chicago &
siteit Eindhoven

siteit Eindhoven

---

r,
towards
on resistance:
g/2012/318

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with $<$100% success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

P-256 discrete-log
total TLS-ECDHE
Should TLS users

tation

ago &
hoven

hoven

ce:
18

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with $<$100% success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

P-256 discrete-log attack $\Rightarrow$
total TLS-ECDHE-P-256 br
Should TLS users worry?

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with $<$100% success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

P-256 discrete-log attack $\Rightarrow$ total TLS-ECDHE-P-256 break! Should TLS users worry?

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with $<100\%$ success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

P-256 discrete-log attack $\Rightarrow$ total TLS-ECDHE-P-256 break! Should TLS users worry?

No. Many researchers have tried and failed to find good P-256 discrete-log attacks.

Concrete security: an example

What is the best NIST P-256 discrete-log attack algorithm?

ECDL input: P-256 points P, Q, where P is a standard generator.

ECDL output: $\log_P Q$.

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

More generally, allow attacks with $<$100% success probability; analyze tradeoffs between "time" and success probability. This talk focuses on high prob.

P-256 discrete-log attack $\Rightarrow$ total TLS-ECDHE-P-256 break! Should TLS users worry?

No. Many researchers have tried and failed to find good P-256 discrete-log attacks.

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

Similar conjectures for AES-128, RSA-3072, etc.: see, e.g., 2005 Bellare–Rogaway.

e security: an example

the best NIST P-256
-log attack algorithm?

nput: P-256 points P, Q,
' is a standard generator.

utput: $\log_P Q$.

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

nerally, allow attacks with
success probability;
tradeoffs between
and success probability.
k focuses on high prob.

P-256 discrete-log attack $\Rightarrow$
total TLS-ECDHE-P-256 break!
Should TLS users worry?

No. Many researchers have
tried and failed to find good
P-256 discrete-log attacks.

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

Similar conjectures for AES-128,
RSA-3072, etc.: see, e.g.,
2005 Bellare–Rogaway.

Concrete

Another
Each TL
with suc
takes "ti

an example

NIST P-256
algorithm?

56 points P, Q,
dard generator.

$_P Q$.

n of "best":

ow attacks with
obability;
between
s probability.
on high prob.

P-256 discrete-log attack $\Rightarrow$
total TLS-ECDHE-P-256 break!
Should TLS users worry?

No. Many researchers have
tried and failed to find good
P-256 discrete-log attacks.

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

Similar conjectures for AES-128,
RSA-3072, etc.: see, e.g.,
2005 Bellare–Rogaway.

Concrete reductio

Another conjectur
Each TLS-ECDHE
with success prob
takes "time" $\geq 2^{12}$

ple

56

n?

P, Q,

ator.

”:

s with

ity.

ob.

P-256 discrete-log attack $\Rightarrow$
total TLS-ECDHE-P-256 break!
Should TLS users worry?

No. Many researchers have tried and failed to find good P-256 discrete-log attacks.

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

Similar conjectures for AES-128, RSA-3072, etc.: see, e.g., 2005 Bellare–Rogaway.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 at
with success probability $\geq p$
takes “time” $\geq 2^{128} p^{1/2}$.

P-256 discrete-log attack $\Rightarrow$ total TLS-ECDHE-P-256 break! Should TLS users worry?

No. Many researchers have tried and failed to find good P-256 discrete-log attacks.

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

Similar conjectures for AES-128, RSA-3072, etc.: see, e.g., 2005 Bellare–Rogaway.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack with success probability $\geq p$ takes "time" $\geq 2^{128} p^{1/2}$.

P-256 discrete-log attack $\Rightarrow$
total TLS-ECDHE-P-256 break!
Should TLS users worry?

No. Many researchers have tried and failed to find good P-256 discrete-log attacks.

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

Similar conjectures for AES-128, RSA-3072, etc.: see, e.g., 2005 Bellare–Rogaway.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack with success probability $\geq p$ takes "time" $\geq 2^{128} p^{1/2}$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break ECDHE-P-256? ECDSA-P-256? ECIES-P-256? ECMQV-P-256? Other P-256-based protocols? Far less attention than for ECDL.

iscrete-log attack ⇒
LS-ECDHE-P-256 break!
TLS users worry?

ny researchers have
d failed to find good
iscrete-log attacks.

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

conjectures for AES-128,
72, etc.: see, e.g.,
llare–Rogaway.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

Why should users have any
confidence in this conjecture?

How many researchers
have really tried to break
ECDHE-P-256? ECDSA-P-256?
ECIES-P-256? ECMQV-P-256?
Other P-256-based protocols?
Far less attention than for ECDL.

Provable

Prove: i
a TLS-E
then the
a P-256
with sim
and succ

attack ⇒
-P-256 break!
worry?

hers have
find good
attacks.

re:

’
algorithm
ability $\geq p$
$^{28}p^{1/2}$.

s for AES-128,
ee, e.g.,
away.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack with success probability $\geq p$ takes "time" $\geq 2^{128} p^{1/2}$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break ECDHE-P-256? ECDSA-P-256? ECIES-P-256? ECMQV-P-256? Other P-256-based protocols? Far less attention than for ECDL.

Provable security t

Prove: if there is
a TLS-ECDHE-P-
then there is
a P-256 discrete-lo
with similar "time"
and success proba

eak!

128,

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack
with success probability $\geq p$
takes "time" $\geq 2^{128} p^{1/2}$.

Why should users have any
confidence in this conjecture?

How many researchers
have really tried to break
ECDHE-P-256? ECDSA-P-256?
ECIES-P-256? ECMQV-P-256?
Other P-256-based protocols?
Far less attention than for ECDL.

Provable security to the reso

Prove: if there is
a TLS-ECDHE-P-256 attack
then there is
a P-256 discrete-log attack
with similar "time"
and success probability.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack with success probability $\geq p$ takes "time" $\geq 2^{128} p^{1/2}$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break ECDHE-P-256? ECDSA-P-256? ECIES-P-256? ECMQV-P-256? Other P-256-based protocols? Far less attention than for ECDL.

Provable security to the rescue!

Prove: if there is a TLS-ECDHE-P-256 attack then there is a P-256 discrete-log attack with similar "time" and success probability.

Concrete reductions

Another conjecture:
Each TLS-ECDHE-P-256 attack with success probability $\geq p$ takes "time" $\geq 2^{128} p^{1/2}$.

Why should users have any confidence in this conjecture?

How many researchers have really tried to break ECDHE-P-256? ECDSA-P-256? ECIES-P-256? ECMQV-P-256? Other P-256-based protocols? Far less attention than for ECDL.

Provable security to the rescue!

Prove: if there is a TLS-ECDHE-P-256 attack then there is a P-256 discrete-log attack with similar "time" and success probability.

Oops: This turns out to be hard. But changing DL to DDH + adding more assumptions allows a proof: Crypto 2012 Jager–Kohlar–Schäge–Schwenk "On the security of TLS-DHE in the standard model".

e reductions

conjecture:
LS-ECDHE-P-256 attack
ccess probability $\geq p$
ime" $\geq 2^{128} p^{1/2}$.

ould users have any
ce in this conjecture?

ny researchers
lly tried to break
-P-256? ECDSA-P-256?
P-256? ECMQV-P-256?
-256-based protocols?
attention than for ECDL.

Provable security to the rescue!

Prove: if there is
a TLS-ECDHE-P-256 attack
then there is
a P-256 discrete-log attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing DL to DDH
+ adding more assumptions
allows a proof: Crypto 2012
Jager–Kohlar–Schäge–Schwenk
"On the security of TLS-DHE
in the standard model".

Similar p
"provabl

Protocol
that har
(e.g., P-
security

After ex
maybe g
of P, an

ns

e:

E-P-256 attack

ability $\geq p$

$^{28}p^{1/2}$.

have any

conjecture?

chers

o break

ECDSA-P-256?

EMQV-P-256?

d protocols?

than for ECDL.

Provable security to the rescue!

Prove: if there is
a TLS-ECDHE-P-256 attack
then there is
a P-256 discrete-log attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing DL to DDH
$+$ adding more assumptions
allows a proof: Crypto 2012
Jager–Kohlar–Schäge–Schwenk
"On the security of TLS-DHE
in the standard model".

Similar pattern thr
"provable security'

Protocol designers
that hardness of a
(e.g., P-256 DDH)
security of various

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

tack

e?

256?
56?
s?
CDL.

Provable security to the rescue!

Prove: if there is
a TLS-ECDHE-P-256 attack
then there is
a P-256 discrete-log attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing DL to DDH
+ adding more assumptions
allows a proof: Crypto 2012
Jager–Kohlar–Schäge–Schwenk
"On the security of TLS-DHE
in the standard model".

Similar pattern throughout t
"provable security" literatur

Protocol designers (try to) p
that hardness of a problem
(e.g., P-256 DDH) implies
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
a TLS-ECDHE-P-256 attack
then there is
a P-256 discrete-log attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing DL to DDH
+ adding more assumptions
allows a proof: Crypto 2012
Jager–Kohlar–Schäge–Schwenk
"On the security of TLS-DHE
in the standard model".

Similar pattern throughout the
"provable security" literature.

Protocol designers (try to) prove
that hardness of a problem P
(e.g., P-256 DDH) 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
a TLS-ECDHE-P-256 attack
then there is
a P-256 discrete-log attack
with similar "time"
and success probability.

Oops: This turns out to be hard.
But changing DL to DDH
+ adding more assumptions
allows a proof: Crypto 2012
Jager–Kohlar–Schäge–Schwenk
"On the security of TLS-DHE
in the standard model".

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., P-256 DDH) 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
ECDHE-P-256 attack
ere is
discrete-log attack
nilar "time"
cess probability.

This turns out to be hard.
nging DL to DDH
g more assumptions
proof: Crypto 2012
ohlar–Schäge–Schwenk
security of TLS-DHE
tandard model".

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., P-256 DDH) 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.

Interlude

How mu
following

```
def pic
  if n0
    if
      i
      r
    if
    ret
  if n1
    if
    ret
  if n2
  retur
```

to the rescue!

256 attack

og attack

”

bility.

out to be hard.

to DDH

sumptions

ypto 2012

äge–Schwenk

of TLS-DHE

odel”.

Similar pattern throughout the “provable security” literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., P-256 DDH) 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.

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
```

cue!

k

hard.

enk

IE

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., P-256 DDH) 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.

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: retur
      return
    if n2 == 0: return
    return
  if n1 == 0:
    if n2 == 0: return
    return
  if n2 == 0: return
  return
```

Similar pattern throughout the "provable security" literature.

Protocol designers (try to) prove that hardness of a problem P (e.g., P-256 DDH) 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.

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
```

pattern throughout the
le security" literature.

l designers (try to) prove
dness of a problem P
256 DDH) implies
of various protocols Q.

tensive cryptanalysis of P,
ain 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.

<u>Interlude regarding "time"</u>

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

roughout the
' literature.

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

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

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

<u>Interlude regarding "time"</u>

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

the
e.

prove
P

Q.

s of P,
rdness
of Q.

ze Q?
have
P.
cols Q.

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 258-“step” P-256 discrete-log 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 258-"step" P-256 discrete-log attack (with 100% success probability). If "time" means "steps" then the standard conjectures are wrong.

e regarding "time"

ch "time" does the
g algorithm take?

```
digit(n0,n1,n2):
) == 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 258-"step" P-256 discrete-log attack (with 100% success probability). If "time" means "steps" then the standard conjectures are wrong.

1994 Be

"We say

A is a (t

A runs i

makes a

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 258-"step" P-256 discrete-log attack (with 100% success probability). If "time" means "steps" then the standard conjectures are wrong.

1994 Bellare–Kilia

"*We say that*

A is a (t, q)*-advers*

A runs in at most

makes at most q c

```
rn 3
   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 258-"step" P-256 discrete-log attack (with 100% success probability). If "time" means "steps" then the standard conjectures are wrong.

1994 Bellare–Kilian–Rogawa

"*We say that*
A is a (t, q)*-adversary if*
A runs in at most t *steps an*
makes at most q *queries to*

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 258-"step" P-256 discrete-log attack (with 100% success probability). If "time" means "steps" then the standard conjectures are wrong.

1994 Bellare–Kilian–Rogaway: "*We say that A is a (t, q)-adversary if A runs in at most t steps and makes at most q queries to $\mathcal{O}$.*"

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 258-"step" P-256 discrete-log attack (with 100% success probability). If "time" means "steps" then the standard conjectures are wrong.

1994 Bellare–Kilian–Rogaway: "*We say that A is a* (t, q)*-adversary if A runs in at most* t *steps and makes at most* q *queries to* $\mathcal{O}$*.*"

Oops: table-lookup attack has very small t.

Paper conjectured "useful" DES security bounds. Any reasonable interpretation of conjecture was false, given paper's definition. Theorems in paper were vacuous.

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 258-
P-256 discrete-log attack
00% success probability).
" means "steps" then the
l conjectures are wrong.

1994 Bellare–Kilian–Rogaway:
"*We say that*
A is a (t, q)*-adversary if*
A runs in at most t *steps and*
makes at most q *queries to* $\mathcal{O}$."

Oops: table-lookup attack
has very small t.

Paper conjectured "useful" DES
security bounds. Any reasonable
interpretation of conjecture was
false, given paper's definition.
Theorems in paper were vacuous.

2000 Be
"*We fix*
Access I
model o
running
executio
of A's d
conventi
caused [
tables . .

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

es 4 "steps".

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

ists a 258-
rete-log attack
ss probability).
steps" then the
res are wrong.

1994 Bellare–Kilian–Rogaway:
"*We say that*
A is a (t, q)*-adversary if*
A runs in at most t *steps and*
makes at most q *queries to* $\mathcal{O}$*.*"

Oops: table-lookup attack
has very small t.

Paper conjectured "useful" DES
security bounds. Any reasonable
interpretation of conjecture was
false, given paper's definition.
Theorems in paper were vacuous.

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 . . ."

es
eps".
eps".
s".
an
k,
.
tack
lity).
en the
ong.

1994 Bellare–Kilian–Rogaway:
"*We say that*
A is a (t, q)*-adversary if*
A runs in at most t *steps and*
makes at most q *queries to* $\mathcal{O}$."

Oops: table-lookup attack
has very small t.

Paper conjectured "useful" DES
security bounds. Any reasonable
interpretation of conjecture was
false, given paper's definition.
Theorems in paper were vacuous.

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 . . ."

1994 Bellare–Kilian–Rogaway:
"We say that
A is a (t,q)*-adversary if*
A runs in at most t *steps and*
makes at most q *queries to* $\mathcal{O}$."

Oops: table-lookup attack
has very small t.

Paper conjectured "useful" DES
security bounds. Any reasonable
interpretation of conjecture was
false, given paper's definition.
Theorems in paper were vacuous.

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 . . ."

1994 Bellare–Kilian–Rogaway:
"We say that A is a (t, q)*-adversary if A runs in at most* t *steps and makes at most* q *queries to* $\mathcal{O}$*."*

Oops: table-lookup attack has very small t.

Paper conjectured "useful" DES security bounds. Any reasonable interpretation of conjecture was false, given paper's definition. Theorems in paper were vacuous.

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 . . ."

Main point of our paper:
There are more pathologies!

Illustrative example: ECDL.

llare–Kilian–Rogaway:
that
$, q$)-adversary if
n at most t steps and
t most q queries to $\mathcal{O}$.”

able-lookup attack
small t.

onjectured “useful” DES
bounds. Any reasonable
tation of conjecture was
ven paper's definition.
ns in paper were vacuous.

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* . . .”

Main point of our paper:
There are more pathologies!

Illustrative example: ECDL.

The rho

Simplifie

Make a
$R_0, R_1,$
where cu
the next

Birthday
Random
elements
after abo
P-256:

The wal
Cycle-fin
(e.g., Fl

n–Rogaway:

sary if
t steps and
queries to $\mathcal{O}$."

p attack

"useful" DES
Any reasonable
onjecture was
s definition.
r were vacuous.

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* . . ."

Main point of our paper:
There are more pathologies!

Illustrative example: ECDL.

The rho method

Simplified, non-pa

Make a pseudo-ra
$R_0, R_1, R_2, \ldots$ in t
where current poi
the next point: R_i

Birthday paradox:
Randomly choosin
elements picks on
after about $\sqrt{\pi \ell /}$
P-256: $\ell \approx 2^{256}$ s

The walk now ent
Cycle-finding algo
(e.g., Floyd) quick

y:

nd

$\mathcal{O}$."

DES

nable

was

n.

uous.

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 . . .*"

Main point of our paper:
There are more pathologies!

Illustrative example: ECDL.

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk
$R_0, R_1, R_2, \ldots$ in the group
where current point determi
the next point: $R_{i+1} = f(R$

Birthday paradox:
Randomly choosing from ℓ
elements picks one element
after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ d

The walk now enters a cycle
Cycle-finding algorithm
(e.g., Floyd) quickly detects

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 . . .*"

Main point of our paper:
There are more pathologies!

Illustrative example: ECDL.

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

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 ."

int of our paper:
re more pathologies!

ve example: ECDL.

<u>The rho method</u>

Simplified, non-parallel rho:

Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Goal: Co

Assume
we know
so that

Then R_i
$y_iP + x$
so $(y_i -$
If $x_i \neq x$
$\log_P Q =$

n–Rogaway:
ticular Random
RAM) as a
tion. . . . A's
ns] A's actual
is the length
. . . This
ates pathologies
arily large lookup

paper:
thologies!

e: ECDL.

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Goal: Compute lo

Assume that for e
we know $x_i, y_i \in$
so that $R_i = y_i P$

Then $R_i = R_j$ me
$y_i P + x_i Q = y_j P$
so $(y_i - y_j)P = ($
If $x_i \neq x_j$ the DL
$\log_P Q = (y_j - y_i$

y:

ndom

a

A's

ctual

rth

logies

lookup

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Goal: Compute $\log_P Q$.

Assume that for each i we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$
If $x_i \neq x_j$ the DLP is solvec
$\log_P Q = (y_j - y_i)/(x_i - x_j$

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Goal: Compute $\log_P Q$.

Assume that for each i we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

The rho method

Simplified, non-parallel rho:

Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
P-256: $\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

Goal: Compute $\log_P Q$.

Assume that for each i we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(P, Q) r-adding walk":
precompute $S_1, S_2, \ldots, S_r$ as random combinations $aP + bQ$;
define $f(R) = R + S_{H(R)}$ where H hashes to $\{1, 2, \ldots, r\}$.

method

ed, non-parallel rho:

pseudo-random walk
$R_2, \ldots$ in the group $\langle P \rangle$,
urrent point determines
point: $R_{i+1} = f(R_i)$.

paradox:
ly choosing from ℓ
s picks one element twice
out $\sqrt{\pi\ell/2}$ draws.
$\ell \approx 2^{256}$ so $\approx 2^{128}$ draws.

k now enters a cycle.
nding algorithm
oyd) quickly detects this.

Goal: Compute $\log_P Q$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(P, Q) r-adding walk":
precompute $S_1, S_2, \ldots, S_r$
as random combinations $aP + bQ$;
define $f(R) = R + S_{H(R)}$
where H hashes to $\{1, 2, \ldots, r\}$.

Parallel

1994 var

Declare
the set c
e.g., all
bits of r

Perform,
different
but sam

Termina
once it h
Report p
Server re
all distin

rallel rho:

ndom walk
the group $\langle P\rangle$,
nt determines
$_{i+1} = f(R_i)$.

g from ℓ
e element twice
2 draws.
o $\approx 2^{128}$ draws.

ers a cycle.
rithm
kly detects this.

Goal: Compute $\log_P Q$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(P, Q) r-adding walk":
precompute $S_1, S_2, \ldots, S_r$
as random combinations $aP + bQ$;
define $f(R) = R + S_{H(R)}$
where H hashes to $\{1, 2, \ldots, r\}$.

Parallel rho

1994 van Oorscho

Declare some subs
the set of *distingu*
e.g., all $R \in \langle P\rangle$ w
bits of representat

Perform, in paralle
different starting p
but same update f

Terminate each wa
once it hits a disti
Report point to ce
Server receives, sto
all distinguished p

k

$\langle P \rangle$,

nes

$_i)$.

twice

raws.

e.

this.

Goal: Compute $\log_P Q$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(P, Q) r-adding walk":
precompute $S_1, S_2, \ldots, S_r$
as random combinations $aP + bQ$;
define $f(R) = R + S_{H(R)}$
where H hashes to $\{1, 2, \ldots, r\}$.

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P \rangle$
the set of *distinguished poin*
e.g., all $R \in \langle P \rangle$ where last
bits of representation of R a

Perform, in parallel, walks fo
different starting points $Q+$
but same update function f

Terminate each walk
once it hits a distinguished p
Report point to central serve
Server receives, stores, and s
all distinguished points.

Goal: Compute $\log_P Q$.

Assume that for each i we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that $y_i P + x_i Q = y_j P + x_j Q$ so $(y_i - y_j)P = (x_j - x_i)Q$. If $x_i \neq x_j$ the DLP is solved: $\log_P Q = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(P, Q) r-adding walk": precompute $S_1, S_2, \ldots, S_r$ as random combinations $aP + bQ$; define $f(R) = R + S_{H(R)}$ where H hashes to $\{1, 2, \ldots, r\}$.

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P \rangle$ to be the set of *distinguished points*: e.g., all $R \in \langle P \rangle$ where last 20 bits of representation of R are 0.

Perform, in parallel, walks for different starting points $Q + yP$ but same update function f.

Terminate each walk once it hits a distinguished point. Report point to central server. Server receives, stores, and sorts all distinguished points.

ompute $\log_P Q$.

that for each i
$x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
$R_i = y_i P + x_i Q$.

$= R_j$ means that
$_i Q = y_j P + x_j Q$
$y_j)P = (x_j - x_i)Q$.
x_j the DLP is solved:
$= (y_j - y_i)/(x_i - x_j)$.

se-(P, Q) r-adding walk":
pute $S_1, S_2, \ldots, S_r$
om combinations $aP + bQ$;
$(R) = R + S_{H(R)}$
hashes to $\{1, 2, \ldots, r\}$.

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P \rangle$ to be the set of *distinguished points*: e.g., all $R \in \langle P \rangle$ where last 20 bits of representation of R are 0.

Perform, in parallel, walks for different starting points $Q + yP$ but same update function f.

Terminate each walk
once it hits a distinguished point.
Report point to central server.
Server receives, stores, and sorts all distinguished points.

State of

Can brea
ℓ in $\sqrt{\pi}$

Use nega
factor $\sqrt{}$

Solving
takes ≈ 2

This is t
cryptana

g$_P\, Q$.

ach i
$\mathbf{Z}/\ell\mathbf{Z}$
$+ x_i Q$.

ans that
$+ x_j Q$
$x_j - x_i)Q$.
P is solved:
$)/(x_i - x_j)$.

r-adding walk":
$, \ldots, S_r$
ations $aP + bQ$;
$+ S_{H(R)}$
o $\{1, 2, \ldots, r\}$.

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P \rangle$ to be the set of *distinguished points*: e.g., all $R \in \langle P \rangle$ where last 20 bits of representation of R are 0.

Perform, in parallel, walks for different starting points $Q + yP$ but same update function f.

Terminate each walk
once it hits a distinguished point.
Report point to central server.
Server receives, stores, and sorts all distinguished points.

State of the art

Can break DLP in
ℓ in $\sqrt{\pi\ell/2}$ group

Use negation map
factor $\sqrt{2}$ for ellip

Solving DLP on N
takes $\approx 2^{128}$ group

This is the best al
cryptanalysts have

.
l:
$_i$).
walk":
$+ bQ$;
$, r\}$.

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P \rangle$ to be the set of *distinguished points*: e.g., all $R \in \langle P \rangle$ where last 20 bits of representation of R are 0.

Perform, in parallel, walks for different starting points $Q + yP$ but same update function f.

Terminate each walk once it hits a distinguished point. Report point to central server. Server receives, stores, and sorts all distinguished points.

State of the art

Can break DLP in group of ℓ in $\sqrt{\pi \ell / 2}$ group operation

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operatio

This is the best algorithm th *cryptanalysts have publishec*

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P\rangle$ to be the set of *distinguished points*: e.g., all $R \in \langle P\rangle$ where last 20 bits of representation of R are 0.

Perform, in parallel, walks for different starting points $Q+yP$ but same update function f.

Terminate each walk once it hits a distinguished point. Report point to central server. Server receives, stores, and sorts all distinguished points.

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

This is the best algorithm that *cryptanalysts have published*.

Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle P \rangle$ to be the set of *distinguished points*: e.g., all $R \in \langle P \rangle$ where last 20 bits of representation of R are 0.

Perform, in parallel, walks for different starting points $Q+yP$ but same update function f.

Terminate each walk once it hits a distinguished point. Report point to central server. Server receives, stores, and sorts all distinguished points.

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

This is the best algorithm that *cryptanalysts have published*.

But is it the best algorithm that *exists*?

rho

n Oorschot–Wiener:

some subset of $\langle P \rangle$ to be
of *distinguished points*:
$R \in \langle P \rangle$ where last 20
epresentation of R are 0.

, in parallel, walks for
starting points $Q+yP$
e update function f.

te each walk
hits a distinguished point.
point to central server.
eceives, stores, and sorts
nguished points.

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

This is the best algorithm that *cryptanalysts have published*.

But is it the best algorithm that *exists*?

This pap

Assumin
overwhe
compute

There ex
algorithr
and has

"Time"

Inescapa
standard
P-256 E
ECDHE

t–Wiener:

set of $\langle P \rangle$ to be
uished points:
where last 20
ion of R are 0.

el, walks for
points $Q+yP$
function f.

alk
nguished point.
entral server.
ores, and sorts
oints.

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

This is the best algorithm that *cryptanalysts have published*.

But is it the best algorithm that *exists*?

This paper's ECDI

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
ECDHE security, e

to be
ts:
20
re 0.

or
yP
.

point.
er.
sorts

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

This is the best algorithm that *cryptanalysts have published*.

But is it the best algorithm that *exists*?

This paper's ECDL algorith

Assuming plausible heuristic
overwhelmingly verified by
computer experiment:

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

"Time" includes algorithm l

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

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

Use negation map to gain factor $\sqrt{2}$ for elliptic curves.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

This is the best algorithm that *cryptanalysts have published*.

But is it the best algorithm that *exists*?

This paper's 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 ECDHE security, etc.) **are false.**

the art

ak DLP in group of order
$\ell/2$ group operations.

ation map to gain
2 for elliptic curves.

DLP on NIST P-256
2^{128} group operations.

he best algorithm that
alysts have published.

the best algorithm
sts?

This paper's 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 ECDHE security, etc.) **are false.**

Should P
be worri
P-256 E

No!

We have
that prin
but B ta

We conj
nobody

group of order
operations.

to gain
tic curves.

IST P-256
operations.

gorithm that
published.

algorithm

This paper's 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 ECDHE security, etc.) **are false.**

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

No!

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

We conjecture tha
nobody will ever p

order
ns.

6
ns.

hat
d.

This paper's 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 ECDHE security, etc.) **are false.**

Should P-256 ECDHE 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

This paper's 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 ECDHE security, etc.) **are false.**

Should P-256 ECDHE 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.

This paper's 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 ECDHE security, etc.) **are false.**

Should P-256 ECDHE 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}.

per's ECDL algorithms

g plausible heuristics,
lmingly verified by
er experiment:

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

includes algorithm length.

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

Should P-256 ECDHE 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

Commor
a 2^{170} "
(indeper
a 2^{85} "n

For cryp
2^{170}, mu

For the s
definitio
The mai
much be

L 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 ECDHE 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

ms

s,

$\approx 2^{85}$

$\approx 1.$

ength.

e

rding

6

alse.

Should P-256 ECDHE 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 cost much better than 2^{128}.

Should P-256 ECDHE 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 ECDHE 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}.

Almost s
redefine
on P on
c_i chose

DHE users
this
rithm A?

m B

" $\approx 2^{170}$.

t
print 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}.

Almost standard w
redefine steps S_i t
on P only; i.e., S_i
c_i chosen uniforml

.

ard

^{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}.

Almost standard walk functi

redefine steps S_i to depend

on P only; i.e., $S_i = c_i P$ wi

c_i chosen uniformly at rando

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}.

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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}.

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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

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}.

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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.

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}.

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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.)

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

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

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

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

Almost standard walk function:
redefine steps S_i to depend
on P only; i.e., $S_i = c_i P$ with
c_i chosen uniformly at random.

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.)

What yo

P-256 is
There ex
AES-128
at cost b
e.g., tim
(Assumi

$\Rightarrow$ Very
between
and actu

Also: Ar
for fixing

eprint.

see the 2^{170}.

: We have
tation"
) followed by
utation".

This costs
than 2^{128}.

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

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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.)

What you find in t

P-256 isn't the on
There *exist* algorit
AES-128, RSA-30
at cost below 2^{128}
e.g., time 2^{85} to b
(Assuming standar

$\Rightarrow$ Very large sepa
between standard
and actual security

Also: Analysis of
for fixing the defin

eprint.iacr.org

^{70}.

e

by

s

s 2^{85},

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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.)

What you find in the full pa

P-256 isn't the only problem
There *exist* algorithms break
AES-128, RSA-3072, DSA-3
at cost below 2^{128};
e.g., time 2^{85} to break AES.
(Assuming standard heuristi

$\Rightarrow$ Very large separation between standard definition and actual security.

Also: Analysis of various ide for fixing the definitions.

eprint.iacr.org/2012/3

Almost standard walk function: redefine steps S_i to depend on P only; i.e., $S_i = c_i P$ with c_i chosen uniformly at random.

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.)

What you find in the full paper:

P-256 isn't the only problem!
There *exist* algorithms breaking AES-128, RSA-3072, DSA-3072 at cost below 2^{128};
e.g., time 2^{85} to break AES.
(Assuming standard heuristics.)

⇒ Very large separation between standard definition and actual security.

Also: Analysis of various ideas for fixing the definitions.

`eprint.iacr.org/2012/318`