

Simplicity

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tanja Lange

Technische Universiteit Eindhoven

NIST's ECC standards

= NSA's prime choices

+ NSA's curve choices

+ NSA's coordinate choices

+ NSA's computation choices

+ NSA's protocol choices.

NIST's ECC standards create
unnecessary complexity
in ECC implementations.

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

Simplicity

D. J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Joint work with:

Tanja Lange

Technische Universiteit Eindhoven

NIST's ECC standards

= NSA's prime choices

+ NSA's curve choices

+ NSA's coordinate choices

+ NSA's computation choices

+ NSA's protocol choices.

NIST's ECC standards create
unnecessary complexity
in ECC implementations.

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

ty

ernstein

ty of Illinois at Chicago &

che Universiteit Eindhoven

ork with:

ange

che Universiteit Eindhoven

ECC standards

s prime choices

s curve choices

s coordinate choices

s computation choices

s protocol choices.

1

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

2

Should c

every im

Replace

1

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

2

Should cryptograp
every imaginable s

Replace GCM with

1

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

2

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

NIST's ECC standards create **unnecessary complexity in ECC implementations.**

This unnecessary complexity

- scares away implementors,
- reduces ECC adoption,
- interferes with optimization,
- keeps ECC out of small devices,
- scares away auditors,
- interferes with verification, and
- creates ECC security failures.

1992 Rivest: *"The poor user is given enough rope with which to hang himself—something a standard should not do."*

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

ECC standards create
necessary complexity
implementations.

necessary complexity
away implementors,
ECC adoption,
res with optimization,
ECC out of small devices,
away auditors,
res with verification, and
ECC security failures.

vest: *“The poor user is
ough rope with which
himself—something
ard should not do.”*

2

Should cryptographers apply
every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate
and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index
calculus. Bigger keys; slower;
much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the
user's performance requirements.

Priority #3 is simplicity.

3

Wild over
examples

“Simplic

“Simplic

ards create
plexity
ntations.

complexity
lementors,
option,
optimization,
of small devices,
itors,
erification, and
urity failures.

*e poor user is
e with which
something
not do."*

2

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

3

Wild overgeneraliz
examples of oversi

“Simplicity damage

“Simplicity damage

2

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

3

Wild overgeneralizations from examples of oversimplification

“Simplicity damages security”

“Simplicity damages speed.”

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Should cryptographers apply every imaginable simplification?

Replace GCM with ECB?

No: ECB doesn't authenticate and doesn't securely encrypt.

Replace ECDH with FFDH?

No: FFDH is vulnerable to index calculus. Bigger keys; slower; much harder security analysis.

Priority #1 is security.

Priority #2 is to meet the user's performance requirements.

Priority #3 is simplicity.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Next-generation ECC simplicity **contributes to security** and **contributes to speed**.

cryptographers apply
imaginable simplification?
GCM with ECB?
3 doesn't authenticate
asn't securely encrypt.
ECDH with FFDH?
DH is vulnerable to index
Bigger keys; slower;
order security analysis.
#1 is security.
#2 is to meet the
performance requirements.
#3 is simplicity.

3

Wild overgeneralizations from
examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are
often used to cover up deficient
analyses of speed and security.

In fact, many simplifications
don't hurt security at all
and don't hurt speed at all.

Next-generation ECC simplicity
contributes to security
and **contributes to speed.**

4

Constant

Imitate

Allocate

for each

Always p

on all bi

3

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Next-generation ECC simplicity **contributes to security** and **contributes to speed**.

4

Constant-time Cur

Imitate hardware i

Allocate constant

for each integer.

Always perform ar

on all bits. Don't

3

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Next-generation ECC simplicity **contributes to security** and **contributes to speed**.

4

Constant-time Curve25519

Imitate hardware in software

Allocate constant number of registers for each integer.

Always perform arithmetic on all bits. Don't skip bits.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Next-generation ECC simplicity **contributes to security** and **contributes to speed**.

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Next-generation ECC simplicity **contributes to security** and **contributes to speed**.

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

Wild overgeneralizations from examples of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

These overgeneralizations are often used to cover up deficient analyses of speed and security.

In fact, many simplifications don't hurt security at all and don't hurt speed at all.

Next-generation ECC simplicity **contributes to security** and **contributes to speed**.

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

If you're multiplying a by b , with 256 bits allocated for a and 256 bits allocated for b : allocate 512 bits for ab .

Overgeneralizations from
of oversimplification:

“Simplicity damages security.”

“Simplicity damages speed.”

Overgeneralizations are
used to cover up deficient
of speed and security.

Many simplifications
hurt security at all
don't hurt speed at all.

Generation ECC simplicity
contributes to security
contributes to speed.

4

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits
for each integer.

Always perform arithmetic
on all bits. Don't skip bits.

If you're adding a to b ,
with 255 bits allocated for a
and 255 bits allocated for b :
allocate 256 bits for $a + b$.

If you're multiplying a by b ,
with 256 bits allocated for a
and 256 bits allocated for b :
allocate 512 bits for ab .

5

If 600 bits

Replace

$r = c m$

same as

Allocate

4

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

If you're multiplying a by b , with 256 bits allocated for a and 256 bits allocated for b : allocate 512 bits for ab .

5

If 600 bits are allo

Replace c with 19

$r = c \bmod 2^{255}$, q

same as c modulo

Allocate 350 bits f

4

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

If you're multiplying a by b , with 256 bits allocated for a and 256 bits allocated for b : allocate 512 bits for ab .

5

If 600 bits are allocated for
 Replace c with $19q + r$ where
 $r = c \bmod 2^{255}$, $q = \lfloor c/2^{255} \rfloor$
 same as c modulo $p = 2^{255}$
 Allocate 350 bits for $19q +$

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

If you're multiplying a by b , with 256 bits allocated for a and 256 bits allocated for b : allocate 512 bits for ab .

If 600 bits are allocated for c :
 Replace c with $19q + r$ where
 $r = c \bmod 2^{255}$, $q = \lfloor c/2^{255} \rfloor$;
 same as c modulo $p = 2^{255} - 19$.
 Allocate 350 bits for $19q + r$.

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

If you're multiplying a by b , with 256 bits allocated for a and 256 bits allocated for b : allocate 512 bits for ab .

If 600 bits are allocated for c :
 Replace c with $19q + r$ where $r = c \bmod 2^{255}$, $q = \lfloor c/2^{255} \rfloor$;
 same as c modulo $p = 2^{255} - 19$.
 Allocate 350 bits for $19q + r$.

Repeat same compression:

350 bits \rightarrow 256 bits.

Small enough for next mult.

Constant-time Curve25519

Imitate hardware in software.

Allocate constant number of bits for each integer.

Always perform arithmetic on all bits. Don't skip bits.

If you're adding a to b , with 255 bits allocated for a and 255 bits allocated for b : allocate 256 bits for $a + b$.

If you're multiplying a by b , with 256 bits allocated for a and 256 bits allocated for b : allocate 512 bits for ab .

If 600 bits are allocated for c :
 Replace c with $19q + r$ where $r = c \bmod 2^{255}$, $q = \lfloor c/2^{255} \rfloor$; same as c modulo $p = 2^{255} - 19$.
 Allocate 350 bits for $19q + r$.

Repeat same compression:

350 bits \rightarrow 256 bits.

Small enough for next mult.

To **completely** reduce 256 bits mod p , do two iterations of constant-time conditional sub.

One conditional sub:

replace c with $c - (1 - s)p$

where s is sign bit in $c - p$.

Constant-time Curve25519

hardware in software.

constant number of bits
integer.

perform arithmetic

bits. Don't skip bits.

adding a to b ,

5 bits allocated for a

bits allocated for b :

256 bits for $a + b$.

multiplying a by b ,

5 bits allocated for a

bits allocated for b :

512 bits for ab .

5

If 600 bits are allocated for c :

Replace c with $19q + r$ where

$$r = c \bmod 2^{255}, \quad q = \lfloor c/2^{255} \rfloor;$$

same as c modulo $p = 2^{255} - 19$.

Allocate 350 bits for $19q + r$.

Repeat same compression:

350 bits \rightarrow 256 bits.

Small enough for next mult.

To **completely** reduce 256 bits
mod p , do two iterations of
constant-time conditional sub.

One conditional sub:

$$\text{replace } c \text{ with } c - (1 - s)p$$

where s is sign bit in $c - p$.

6

Constant

NIST P-

$$2^{256} - 2$$

ECDSA

reductio

an integ

Write A

$$(A_{15}, A_{14}, \dots, A_8, A_7, \dots)$$

$$A_8, A_7, \dots$$

meaning

Define

$$T; S_1; S_2$$

as

5

Curve25519

in software.

number of bits

arithmetic

skip bits.

to b ,

allocated for a

allocated for b :

or $a + b$.

ing a by b ,

allocated for a

allocated for b :

or ab .

If 600 bits are allocated for c :

Replace c with $19q + r$ where

$$r = c \bmod 2^{255}, \quad q = \lfloor c/2^{255} \rfloor;$$

same as c modulo $p = 2^{255} - 19$.

Allocate 350 bits for $19q + r$.

Repeat same compression:

350 bits \rightarrow 256 bits.

Small enough for next mult.

To **completely** reduce 256 bits

mod p , do two iterations of

constant-time conditional sub.

One conditional sub:

replace c with $c - (1 - s)p$

where s is sign bit in $c - p$.

6

Constant-time NIST

NIST P-256 prime

$$2^{256} - 2^{224} + 2^{192}$$

ECDSA standard s

reduction procedure

an integer "A less

Write A as

$$(A_{15}, A_{14}, A_{13}, A_{12},$$

$$A_8, A_7, A_6, A_5, A_4,$$

meaning $\sum_i A_i 2^{32i}$

Define

$$T; S_1; S_2; S_3; S_4; D$$

as

5

If 600 bits are allocated for c :
 Replace c with $19q + r$ where
 $r = c \bmod 2^{255}$, $q = \lfloor c/2^{255} \rfloor$;
 same as c modulo $p = 2^{255} - 19$.
 Allocate 350 bits for $19q + r$.

Repeat same compression:
 350 bits \rightarrow 256 bits.

Small enough for next mult.

To **completely** reduce 256 bits
 mod p , do two iterations of
 constant-time conditional sub.

One conditional sub:
 replace c with $c - (1 - s)p$
 where s is sign bit in $c - p$.

6

Constant-time NIST P-256

NIST P-256 prime p is
 $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

ECDSA standard specifies
 reduction procedure given
 an integer “ A less than p^2 ”:

Write A as

$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10},$
 $A_8, A_7, A_6, A_5, A_4, A_3, A_2,$
 meaning $\sum_i A_i 2^{32i}$.

Define

$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3$
 as

If 600 bits are allocated for c :
 Replace c with $19q + r$ where
 $r = c \bmod 2^{255}$, $q = \lfloor c/2^{255} \rfloor$;
 same as c modulo $p = 2^{255} - 19$.

Allocate 350 bits for $19q + r$.

Repeat same compression:
 350 bits \rightarrow 256 bits.

Small enough for next mult.

To **completely** reduce 256 bits
 mod p , do two iterations of
 constant-time conditional sub.

One conditional sub:
 replace c with $c - (1 - s)p$
 where s is sign bit in $c - p$.

Constant-time NIST P-256

NIST P-256 prime p is
 $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies
 reduction procedure given
 an integer “ A less than p^2 ”:

Write A as

$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$
 $A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,
 meaning $\sum_i A_i 2^{32i}$.

Define

$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$
 as

bits are allocated for c :

c with $19q + r$ where

$\text{mod } 2^{255}$, $q = \lfloor c/2^{255} \rfloor$;

$c \text{ modulo } p = 2^{255} - 19$.

350 bits for $19q + r$.

same compression:

→ 256 bits.

enough for next mult.

completely reduce 256 bits

do two iterations of

constant-time conditional sub.

conditional sub:

c with $c - (1 - s)p$

is sign bit in $c - p$.

Constant-time NIST P-256

NIST P-256 prime p is

$$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1.$$

ECDSA standard specifies

reduction procedure given

an integer “ A less than p^2 ”:

Write A as

$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$

$A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,

meaning $\sum_i A_i 2^{32i}$.

Define

$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$

as

$(A_7, A_6,$

$(A_{15}, A_{14},$

$(0, A_{15}, A_{14},$

$(A_{15}, A_{14},$

$(A_8, A_{13},$

$(A_{10}, A_8,$

$(A_{11}, A_9,$

$(A_{12}, 0, A_{11},$

$(A_{13}, 0, A_{12},$

Compute

$S_4 - D_1$

Reduce

subtract

cated for c :
 $q + r$ where
 $q = \lfloor c/2^{255} \rfloor$;
 $p = 2^{255} - 19$.

For $19q + r$.

pression:

ts.

next mult.

duce 256 bits

rations of

ditional sub.

ub:

$(1 - s)p$

in $c - p$.

Constant-time NIST P-256

NIST P-256 prime p is

$$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1.$$

ECDSA standard specifies

reduction procedure given

an integer “ A less than p^2 ”:

Write A as

$$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9, \\ A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0),$$

meaning $\sum_i A_i 2^{32i}$.

Define

$$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$$

as

$(A_7, A_6, A_5, A_4, A_3,$

$A_{15}, A_{14}, A_{13}, A_{12},$

$(0, A_{15}, A_{14}, A_{13}, A_{12},$

$(A_{15}, A_{14}, 0, 0, 0, A_{15},$

$(A_8, A_{13}, A_{15}, A_{14},$

$(A_{10}, A_8, 0, 0, 0, A_{15},$

$(A_{11}, A_9, 0, 0, A_{15},$

$(A_{12}, 0, A_{10}, A_9, A_{15},$

$(A_{13}, 0, A_{11}, A_{10}, A_{15},$

Compute $T + 2S_1$

$S_4 - D_1 - D_2 - D_3 - D_4$

Reduce modulo p

subtracting a few

Constant-time NIST P-256

NIST P-256 prime p is

$$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1.$$

ECDSA standard specifies reduction procedure given an integer “ A less than p^2 ”:

Write A as

$$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9, A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0),$$

meaning $\sum_i A_i 2^{32i}$.

Define

$$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$$

as

$$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$$

$$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0)$$

$$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$$

$$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8)$$

$$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11},$$

$$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11},$$

$$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13},$$

$$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14},$$

$$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15},$$

$$\text{Compute } T + 2S_1 + 2S_2 + S_4 - D_1 - D_2 - D_3 - D_4.$$

Reduce modulo p “by adding or subtracting a few copies” of

Constant-time NIST P-256

NIST P-256 prime p is
 $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies
 reduction procedure given
 an integer “ A less than p^2 ”:

Write A as

$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$
 $A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,
 meaning $\sum_i A_i 2^{32i}$.

Define

$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$
 as

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

Compute $T + 2S_1 + 2S_2 + S_3 +$
 $S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or
 subtracting a few copies” of p .

Long-time NIST P-256

P-256 prime p is

$$2^{224} + 2^{192} + 2^{96} - 1.$$

standard specifies

an procedure given

for “ A less than p^2 ”:

as

$A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$

$A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0$),

$$\sum_i A_i 2^{32i}.$$

$S_2; S_3; S_4; D_1; D_2; D_3; D_4$

7

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$

$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$

$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$

$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$

$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$

$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$

$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$

$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$

$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or subtracting a few copies” of p .

8

What is

A loop?

presuma

p is
 $+ 2^{96} - 1$.

specifies
 re given
 than p^2 ”:

$A_{11}, A_{10}, A_9,$
 $A_4, A_3, A_2, A_1, A_0),$
 i .

$D_1; D_2; D_3; D_4$

7

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 +$
 $S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or
 subtracting a few copies” of p .

8

What is “a few co
 A loop? **Variable**
 presumably a secu

7

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4.$

Reduce modulo p “by adding or subtracting a few copies” of $p.$

8

What is “a few copies”?
 A loop? **Variable time**,
 presumably a security problem

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4.$

Reduce modulo p “by adding or subtracting a few copies” of $p.$

What is “a few copies”?
 A loop? **Variable time**,
 presumably a security problem.

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or subtracting a few copies” of p .

What is “a few copies”?
 A loop? **Variable time**,
 presumably a security problem.

Correct but quite slow:
 conditionally add $4p$,
 conditionally add $2p$,
 conditionally add p ,
 conditionally sub $4p$,
 conditionally sub $2p$,
 conditionally sub p .

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or subtracting a few copies” of p .

What is “a few copies”?
 A loop? **Variable time**,
 presumably a security problem.

Correct but quite slow:
 conditionally add $4p$,
 conditionally add $2p$,
 conditionally add p ,
 conditionally sub $4p$,
 conditionally sub $2p$,
 conditionally sub p .

Delay until end of computation?
 Trouble: “A less than p^2 ”.

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or subtracting a few copies” of p .

What is “a few copies”?
 A loop? **Variable time**,
 presumably a security problem.

Correct but quite slow:
 conditionally add $4p$,
 conditionally add $2p$,
 conditionally add p ,
 conditionally sub $4p$,
 conditionally sub $2p$,
 conditionally sub p .

Delay until end of computation?
 Trouble: “ A less than p^2 ”.

Even worse: what about platforms
 where 2^{32} isn't best radix?

$(A_5, A_4, A_3, A_2, A_1, A_0);$
 $(A_4, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
 $(A_{14}, A_{13}, A_{12}, 0, 0, 0);$
 $(A_4, 0, 0, 0, A_{10}, A_9, A_8);$
 $(A_4, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
 $(A_4, 0, 0, 0, A_{13}, A_{12}, A_{11});$
 $(A_4, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
 $(A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
 $(A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

$e T + 2S_1 + 2S_2 + S_3 +$
 $- D_2 - D_3 - D_4.$

modulo p “by adding or
 subtracting a few copies” of p .

What is “a few copies”?
 A loop? **Variable time**,
 presumably a security problem.

Correct but quite slow:
 conditionally add $4p$,
 conditionally add $2p$,
 conditionally add p ,
 conditionally sub $4p$,
 conditionally sub $2p$,
 conditionally sub p .

Delay until end of computation?
 Trouble: “ A less than p^2 ”.

Even worse: what about platforms
 where 2^{32} isn't best radix?

The Mo

x_2, z_2, x_3

for i in

 bit =

x_2, x_3

z_2, z_3

x_3, z_3

x_2, z_2

$4 * x_3$

x_2, x_3

z_2, z_3

return :

```

3, A2, A1, A0);
2, A11, 0, 0, 0);
A12, 0, 0, 0);
A10, A9, A8);
A13, A11, A10, A9);
13, A12, A11);
A14, A13, A12);
8, A15, A14, A13);
A9, 0, A15, A14).
+ 2S2 + S3 +
D3 - D4.

```

“by adding or
copies” of p .

What is “a few copies”?

A loop? **Variable time**,
presumably a security problem.

Correct but quite slow:

conditionally add $4p$,
conditionally add $2p$,
conditionally add p ,
conditionally sub $4p$,
conditionally sub $2p$,
conditionally sub p .

Delay until end of computation?

Trouble: “A less than p^2 ”.

Even worse: what about platforms
where 2^{32} isn't best radix?

The Montgomery

```

x2, z2, x3, z3 = 1,
for i in reverse
    bit = 1 & (n >
x2, x3 = cswap(
z2, z3 = cswap(
x3, z3 = ((x2*x
                x1*(x2*z
x2, z2 = ((x2^2
                4*x2*z2*(x2^
x2, x3 = cswap(
z2, z3 = cswap(
return x2*z2^(p-

```

A_0);
 $, 0)$);
 $)$);
 $)$);
 $A_{10}, A_9)$);
 $1)$);
 $A_{12})$);
 $, A_{13})$);
 $A_{14})$.
 $S_3 +$
 g or
 p .

What is “a few copies”?

A loop? **Variable time**,
presumably a security problem.

Correct but quite slow:

conditionally add $4p$,

conditionally add $2p$,

conditionally add p ,

conditionally sub $4p$,

conditionally sub $2p$,

conditionally sub p .

Delay until end of computation?

Trouble: “ A less than p^2 ”.

Even worse: what about platforms
where 2^{32} isn't best radix?

The Montgomery ladder

```
x2, z2, x3, z3 = 1, 0, x1, 1
```

```
for i in reversed(range(2
```

```
    bit = 1 & (n >> i)
```

```
    x2, x3 = cswap(x2, x3, bit
```

```
    z2, z3 = cswap(z2, z3, bit
```

```
    x3, z3 = ((x2*x3-z2*z3) ^
```

```
              x1*(x2*z3-z2*x3) ^
```

```
    x2, z2 = ((x2^2-z2^2) ^2,
```

```
              4*x2*z2*(x2^2+A*x2*z2
```

```
    x2, x3 = cswap(x2, x3, bit
```

```
    z2, z3 = cswap(z2, z3, bit
```

```
return x2*z2^(p-2)
```

What is “a few copies”?

A loop? **Variable time**,
presumably a security problem.

Correct but quite slow:

conditionally add $4p$,

conditionally add $2p$,

conditionally add p ,

conditionally sub $4p$,

conditionally sub $2p$,

conditionally sub p .

Delay until end of computation?

Trouble: “A less than p^2 ”.

Even worse: what about platforms
where 2^{32} isn't best radix?

The Montgomery ladder

```
x2,z2,x3,z3 = 1,0,x1,1
```

```
for i in reversed(range(255)):
```

```
    bit = 1 & (n >> i)
```

```
    x2,x3 = cswap(x2,x3,bit)
```

```
    z2,z3 = cswap(z2,z3,bit)
```

```
    x3,z3 = ((x2*x3-z2*z3)^2,
```

```
            x1*(x2*z3-z2*x3)^2)
```

```
    x2,z2 = ((x2^2-z2^2)^2,
```

```
            4*x2*z2*(x2^2+A*x2*z2+z2^2))
```

```
    x2,x3 = cswap(x2,x3,bit)
```

```
    z2,z3 = cswap(z2,z3,bit)
```

```
return x2*z2^(p-2)
```

“a few copies”?

Variable time,

ably a security problem.

but quite slow:

nally add $4p$,

nally add $2p$,

nally add p ,

nally sub $4p$,

nally sub $2p$,

nally sub p .

until end of computation?

“ A less than p^2 ”.

orse: what about platforms

32 isn't best radix?

The Montgomery ladder

```
x2,z2,x3,z3 = 1,0,x1,1
```

```
for i in reversed(range(255)):
```

```
    bit = 1 & (n >> i)
```

```
    x2,x3 = cswap(x2,x3,bit)
```

```
    z2,z3 = cswap(z2,z3,bit)
```

```
    x3,z3 = ((x2*x3-z2*z3)^2,
```

```
            x1*(x2*z3-z2*x3)^2)
```

```
    x2,z2 = ((x2^2-z2^2)^2,
```

```
            4*x2*z2*(x2^2+A*x2*z2+z2^2))
```

```
    x2,x3 = cswap(x2,x3,bit)
```

```
    z2,z3 = cswap(z2,z3,bit)
```

```
return x2*z2^(p-2)
```

Simple;

compute

on $y^2 =$

when A^2

pies" ?
time,
 rity problem.

slow:

$4p$,

$2p$,

p ,

$4p$,

$2p$,

p .

computation?

han p^2 ".

about platforms

st radix?

The Montgomery ladder

```
x2,z2,x3,z3 = 1,0,x1,1
```

```
for i in reversed(range(255)):
```

```
    bit = 1 & (n >> i)
```

```
    x2,x3 = cswap(x2,x3,bit)
```

```
    z2,z3 = cswap(z2,z3,bit)
```

```
    x3,z3 = ((x2*x3-z2*z3)^2,
```

```
             x1*(x2*z3-z2*x3)^2)
```

```
    x2,z2 = ((x2^2-z2^2)^2,
```

```
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
```

```
    x2,x3 = cswap(x2,x3,bit)
```

```
    z2,z3 = cswap(z2,z3,bit)
```

```
return x2*z2^(p-2)
```

Simple; fast; **alwa**
 computes scalar m
 on $y^2 = x^3 + Ax^2$
 when $A^2 - 4$ is no

The Montgomery ladder

```
x2, z2, x3, z3 = 1, 0, x1, 1
```

```
for i in reversed(range(255)):
```

```
    bit = 1 & (n >> i)
```

```
    x2, x3 = cswap(x2, x3, bit)
```

```
    z2, z3 = cswap(z2, z3, bit)
```

```
    x3, z3 = ((x2*x3-z2*z3)^2,
```

```
              x1*(x2*z3-z2*x3)^2)
```

```
    x2, z2 = ((x2^2-z2^2)^2,
```

```
              4*x2*z2*(x2^2+A*x2*z2+z2^2))
```

```
    x2, x3 = cswap(x2, x3, bit)
```

```
    z2, z3 = cswap(z2, z3, bit)
```

```
return x2*z2^(p-2)
```

Simple; fast; **always**

computes scalar multiplication

on $y^2 = x^3 + Ax^2 + x$

when $A^2 - 4$ is non-square.

The Montgomery ladder

```

x2,z2,x3,z3 = 1,0,x1,1
for i in reversed(range(255)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
return x2*z2^(p-2)

```

Simple; fast; **always**

computes scalar multiplication

on $y^2 = x^3 + Ax^2 + x$

when $A^2 - 4$ is non-square.

The Montgomery ladder

```

x2,z2,x3,z3 = 1,0,x1,1
for i in reversed(range(255)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
return x2*z2^(p-2)

```

Simple; fast; **always**

computes scalar multiplication
on $y^2 = x^3 + Ax^2 + x$
when $A^2 - 4$ is non-square.

With some extra lines
can compute (x, y) output
given (x, y) input.

But simpler to use just x ,
as proposed by 1985 Miller.

The Montgomery ladder

```

x2,z2,x3,z3 = 1,0,x1,1
for i in reversed(range(255)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
return x2*z2^(p-2)

```

Simple; fast; **always**

computes scalar multiplication
on $y^2 = x^3 + Ax^2 + x$
when $A^2 - 4$ is non-square.

With some extra lines
can compute (x, y) output
given (x, y) input.

But simpler to use just x ,
as proposed by 1985 Miller.

Adaptations to NIST curves
are much slower; not as simple;
not proven to always work.

Other scalar-mult methods:
proven but much more complex.

Montgomery ladder

```
z3, z3 = 1, 0, x1, 1
```

```
for i in reversed(range(255)):
```

```
    bit = (x1 & (n >> i))
```

```
    x2, x3 = cswap(x2, x3, bit)
```

```
    z2, z3 = cswap(z2, z3, bit)
```

```
    x2 = ((x2*x3-z2*z3)^2,
```

```
    x1*(x2*z3-z2*x3)^2)
```

```
    z2 = ((x2^2-z2^2)^2,
```

```
    2*z2*(x2^2+A*x2*z2+z2^2))
```

```
    x2, x3 = cswap(x2, x3, bit)
```

```
    z2, z3 = cswap(z2, z3, bit)
```

```
return x2*z2^(p-2)
```

Simple; fast; **always**

computes scalar multiplication

on $y^2 = x^3 + Ax^2 + x$

when $A^2 - 4$ is non-square.

With some extra lines

can compute (x, y) output

given (x, y) input.

But simpler to use just x ,

as proposed by 1985 Miller.

Adaptations to NIST curves

are much slower; not as simple;

not proven to always work.

Other scalar-mult methods:

proven but much more complex.

“Hey, yo

that x_1 i

ladder

```

0, x1, 1
d(range(255)):
> i)
x2, x3, bit)
z2, z3, bit)
3-z2*z3)^2,
3-z2*x3)^2)
-z2^2)^2,
2+A*x2*z2+z2^2))
x2, x3, bit)
z2, z3, bit)
2)

```

Simple; fast; **always**
 computes scalar multiplication
 on $y^2 = x^3 + Ax^2 + x$
 when $A^2 - 4$ is non-square.

With some extra lines
 can compute (x, y) output
 given (x, y) input.

But simpler to use just x ,
 as proposed by 1985 Miller.

Adaptations to NIST curves
 are much slower; not as simple;
 not proven to always work.

Other scalar-mult methods:
 proven but much more complex.

“Hey, you forgot to
 that x_1 is on the c

Simple; fast; **always**
 computes scalar multiplication
 on $y^2 = x^3 + Ax^2 + x$
 when $A^2 - 4$ is non-square.

With some extra lines
 can compute (x, y) output
 given (x, y) input.

But simpler to use just x ,
 as proposed by 1985 Miller.

Adaptations to NIST curves
 are much slower; not as simple;
 not proven to always work.
 Other scalar-mult methods:
 proven but much more complex.

“Hey, you forgot to check
 that x_1 is on the curve!”

Simple; fast; **always**
computes scalar multiplication
on $y^2 = x^3 + Ax^2 + x$
when $A^2 - 4$ is non-square.

With some extra lines
can compute (x, y) output
given (x, y) input.

But simpler to use just x ,
as proposed by 1985 Miller.

Adaptations to NIST curves
are much slower; not as simple;
not proven to always work.

Other scalar-mult methods:
proven but much more complex.

“Hey, you forgot to check
that x_1 is on the curve!”

Simple; fast; **always**
computes scalar multiplication
on $y^2 = x^3 + Ax^2 + x$
when $A^2 - 4$ is non-square.

With some extra lines
can compute (x, y) output
given (x, y) input.

But simpler to use just x ,
as proposed by 1985 Miller.

Adaptations to NIST curves
are much slower; not as simple;
not proven to always work.

Other scalar-mult methods:
proven but much more complex.

“Hey, you forgot to check
that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

Simple; fast; **always**
 computes scalar multiplication
 on $y^2 = x^3 + Ax^2 + x$
 when $A^2 - 4$ is non-square.

With some extra lines
 can compute (x, y) output
 given (x, y) input.
 But simpler to use just x ,
 as proposed by 1985 Miller.

Adaptations to NIST curves
 are much slower; not as simple;
 not proven to always work.

Other scalar-mult methods:
 proven but much more complex.

“Hey, you forgot to check
 that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

“This textbook tells me
 to start the Montgomery ladder
 from the top bit *set* in n !”
 (Exploited in, e.g., 2011
 Brumley–Tuveri “Remote timing
 attacks are still practical” .)

Simple; fast; **always**
 computes scalar multiplication
 on $y^2 = x^3 + Ax^2 + x$
 when $A^2 - 4$ is non-square.

With some extra lines
 can compute (x, y) output
 given (x, y) input.
 But simpler to use just x ,
 as proposed by 1985 Miller.

Adaptations to NIST curves
 are much slower; not as simple;
 not proven to always work.
 Other scalar-mult methods:
 proven but much more complex.

“Hey, you forgot to check
 that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

“This textbook tells me
 to start the Montgomery ladder
 from the top bit *set* in n !”
 (Exploited in, e.g., 2011
 Brumley–Tuveri “Remote timing
 attacks are still practical”.)

The Curve25519 DH function
 takes $2^{254} \leq n < 2^{255}$,
 so this is still constant-time.

fast; **always**

es scalar multiplication

$$x^3 + Ax^2 + x$$

-4 is non-square.

me extra lines

pute (x, y) output

, y) input.

pler to use just x ,

osed by 1985 Miller.

ions to NIST curves

h slower; not as simple;

ven to always work.

calar-mult methods:

out much more complex.

“Hey, you forgot to check
that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

“This textbook tells me
to start the Montgomery ladder
from the top bit *set* in n !”

(Exploited in, e.g., 2011

Brumley–Tuveri “Remote timing
attacks are still practical”.)

The Curve25519 DH function

takes $2^{254} \leq n < 2^{255}$,

so this is still constant-time.

Many m

[blog.cr](#)

[/201403](#)

analyzes

designing

Unneces

ECDSA:

Weierstr

variable-

Next-gen

much sim

much sim

much sim

ys
multiplication
+ x
n-square.
ines
) output
e just x,
85 Miller.
ST curves
not as simple;
ays work.
methods:
more complex.

“Hey, you forgot to check that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

“This textbook tells me to start the Montgomery ladder from the top bit *set* in $n!$ ”

(Exploited in, e.g., 2011 Brumley–Tuveri “Remote timing attacks are still practical” .)

The Curve25519 DH function takes $2^{254} \leq n < 2^{255}$, so this is still constant-time.

Many more issues
blog.cr.yp.to/20140323-ecdsa
analyzes choices m
designing ECC sig
Unnecessary comp
ECDSA: scalar inv
Weierstrass incom
variable-time NAF
Next-generation E
much simpler for i
much simpler for c
much simpler for a

“Hey, you forgot to check that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

“This textbook tells me to start the Montgomery ladder from the top bit *set* in n !”
(Exploited in, e.g., 2011 Brumley–Tuveri “Remote timing attacks are still practical” .)

The Curve25519 DH function takes $2^{254} \leq n < 2^{255}$, so this is still constant-time.

Many more issues

blog.cr.yp.to

[/20140323-ecdsa.html](http://blog.cr.yp.to/20140323-ecdsa.html)

analyzes choices made in designing ECC signatures.

Unnecessary complexity in ECDSA: scalar inversion; Weierstrass incompleteness; variable-time NAF; et al.

Next-generation ECC is much simpler for implementers, much simpler for designers, much simpler for auditors, e

“Hey, you forgot to check that x_1 is on the curve!”

No need to check.

Curve25519 is **twist-secure**.

“This textbook tells me to start the Montgomery ladder from the top bit *set* in n !”
(Exploited in, e.g., 2011 Brumley–Tuveri “Remote timing attacks are still practical” .)

The Curve25519 DH function takes $2^{254} \leq n < 2^{255}$, so this is still constant-time.

Many more issues

blog.cr.yp.to

[/20140323-ecdsa.html](http://blog.cr.yp.to/20140323-ecdsa.html)

analyzes choices made in designing ECC signatures.

Unnecessary complexity in ECDSA: scalar inversion; Weierstrass incompleteness; variable-time NAF; et al.

Next-generation ECC is much simpler for implementors, much simpler for designers, much simpler for auditors, etc.