High-speed cryptography

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

with some slides from:
Tanja Lange
Technische Universiteit Eindhoven

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow $\Rightarrow$ fewer users $\Rightarrow$ fewer cryptanalysts $\Rightarrow$ less attractive for everybody.

eed cryptography

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

ne slides from:
ange
che Universiteit Eindhoven

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow $\Rightarrow$ fewer users $\Rightarrow$ fewer cryptanalysts $\Rightarrow$ less attractive for everybody.

Impleme

e.g. Vari
arithmet
consume
Includes
optimize

graphy

n

is at Chicago &

siteit Eindhoven

rom:

siteit Eindhoven

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow $\Rightarrow$ fewer users $\Rightarrow$ fewer cryptanalysts $\Rightarrow$ less attractive for everybody.

Implementors purs

e.g. Variable-lengt
arithmetic library i
consumes 50000 li
Includes 38 asm ir
optimized for vario

ago &
hoven

hoven

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow $\Rightarrow$ fewer users $\Rightarrow$ fewer cryptanalysts $\Rightarrow$ less attractive for everybody.

Implementors pursue speed

e.g. Variable-length-big-integ
arithmetic library inside Ope
consumes 50000 lines of coc
Includes 38 asm implementa
optimized for various CPUs.

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow $\Rightarrow$ fewer users $\Rightarrow$ fewer cryptanalysts $\Rightarrow$ less attractive for everybody.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

Do we care about speed?

Almost all software is much slower than it could be.

Is software applied to much data? Usually not. Usually the wasted CPU time is negligible.

But *crypto* software should be applied to all communication.

Crypto that's too slow ⇒ fewer users ⇒ fewer cryptanalysts ⇒ less attractive for everybody.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA verification computes $(S^{-1}H(M))B + (S^{-1}R)A$. OpenSSL has complicated code for fast computation of S^{-1}. Much simpler code would make verification considerably slower.

are about speed?

all software is
ower than it could be.

are applied to much data?
not. Usually the
CPU time is negligible.

pto software should be
to all communication.

that's too slow ⇒
ers ⇒ fewer cryptanalysts
attractive for everybody.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA verification computes $(S^{-1}H(M))B + (S^{-1}R)A$. OpenSSL has complicated code for fast computation of S^{-1}. Much simpler code would make verification considerably slower.

Applicat

e.g. Late
practices
(2012) s
a regula
it is esti
can safe
at least
Signing
bit key t
1024-bit
(such as
four tim

speed?

e is
it could be.

to much data?
lly the
is negligible.

re should be
munication.

slow ⇒
ver cryptanalysts
for everybody.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA verification computes $(S^{-1}H(M))B + (S^{-1}R)A$. OpenSSL has complicated code for fast computation of S^{-1}. Much simpler code would make verification considerably slower.

Applications pursu

e.g. Latest "DNSS
practices" recomm
(2012) says "No o
a regular 1024-bit
it is estimated tha
can safely use 102
at least the next t
Signing and verifyi
bit key takes longe
1024-bit key . . . p
(such as verificatio
four times slower.'

e.

data?

le.

be

n.

alysts

ody.

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA verification computes $(S^{-1}H(M))B + (S^{-1}R)A$. OpenSSL has complicated code for fast computation of S^{-1}. Much simpler code would make verification considerably slower.

Applications pursue speed

e.g. Latest "DNSSEC operat
practices" recommendation
(2012) says "No one has bro
a regular 1024-bit [RSA] key
it is estimated that most zo
can safely use 1024-bit keys
at least the next ten years .
Signing and verifying with a
bit key takes longer than wit
1024-bit key . . . public oper
(such as verification) are abo
four times slower."

Implementors pursue speed

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA verification computes $(S^{-1}H(M))B + (S^{-1}R)A$. OpenSSL has complicated code for fast computation of S^{-1}. Much simpler code would make verification considerably slower.

Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

...entors pursue speed

...able-length-big-integer
...tic library inside OpenSSL
...es 50000 lines of code.
... 38 asm implementations
...ed for various CPUs.

...DSA verification computes
$...M))B + (S^{-1}R)A$.
...L has complicated code
...computation of S^{-1}.
...mpler code would make
...ion considerably slower.

Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

DNSSE...

2048-bi...
co...

sue speed

h-big-integer
nside OpenSSL
nes of code.
mplementations
ous CPUs.

cation computes
$S^{-1}R)A$.
plicated code
on of S^{-1}.
e would make
erably slower.

Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

DNSSEC key sizes

2048-bit DNSSEC controlled b

ger
enSSL
de.
tions

nputes

ode
.
ake
ver.

Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

DNSSEC key sizes, 2016.11.

2048-bit DNSSEC master k
controlled by U.S.

Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

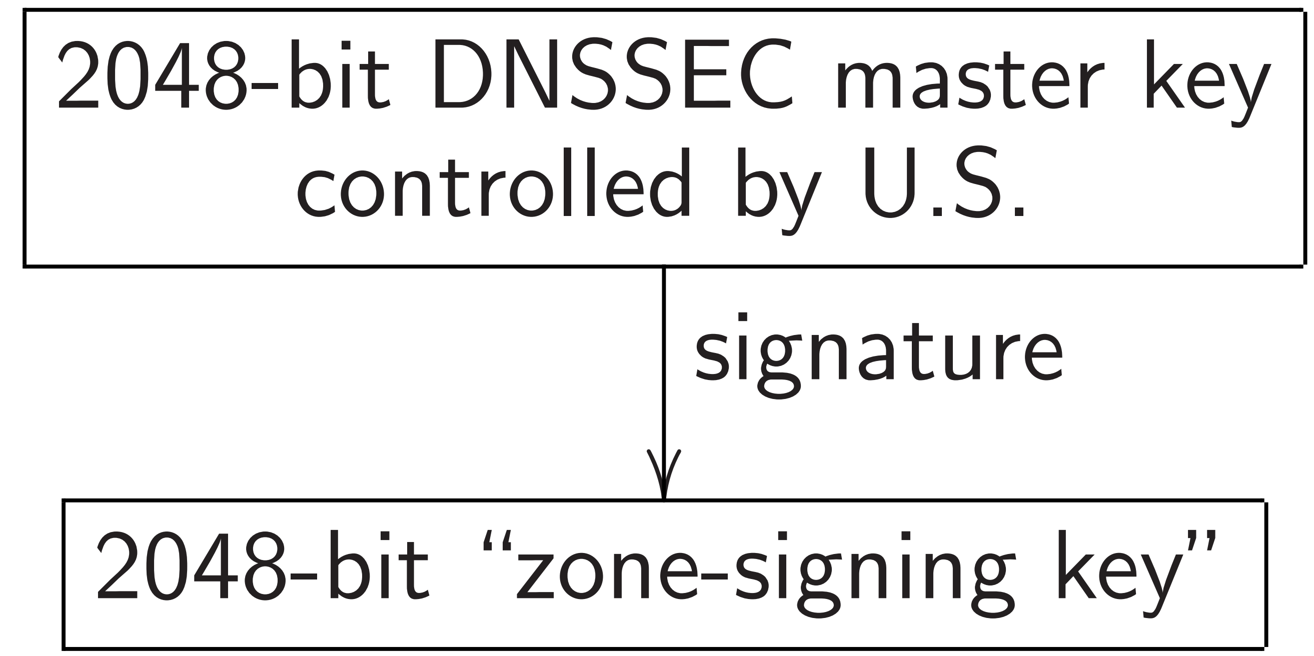
DNSSEC key sizes, 2016.11.28:

2048-bit DNSSEC master key controlled by U.S.

Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

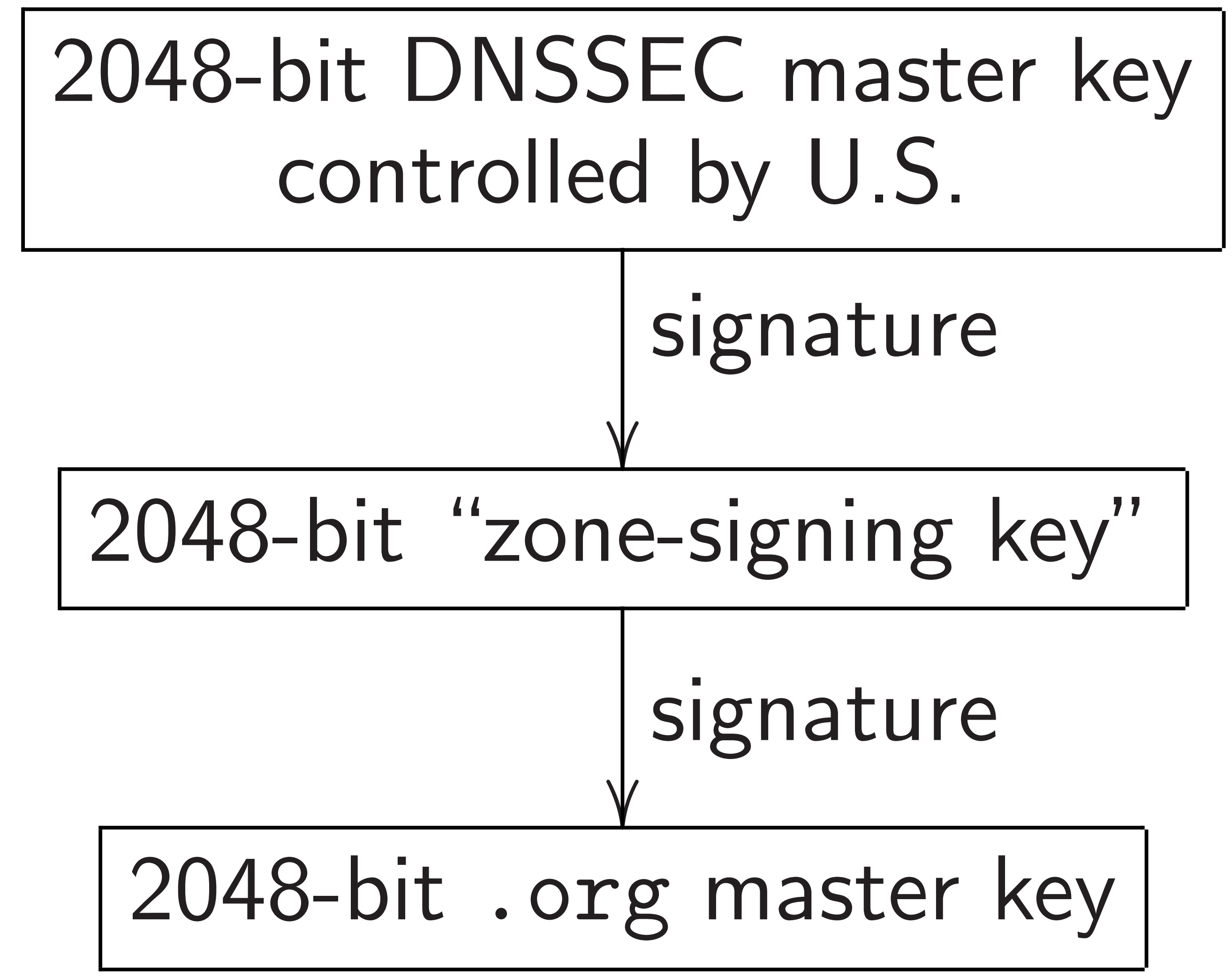
DNSSEC key sizes, 2016.11.28:



Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

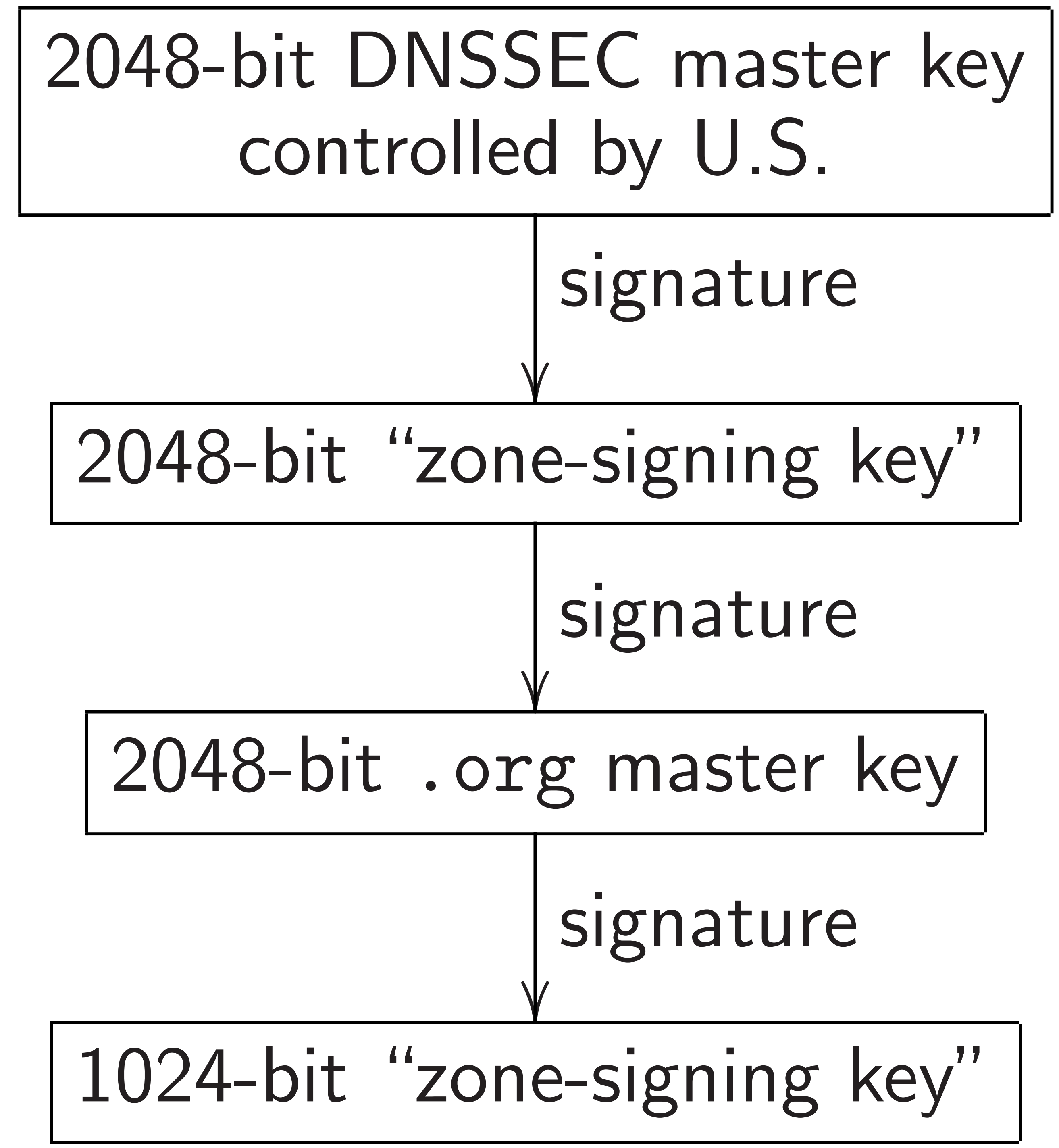
DNSSEC key sizes, 2016.11.28:



Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

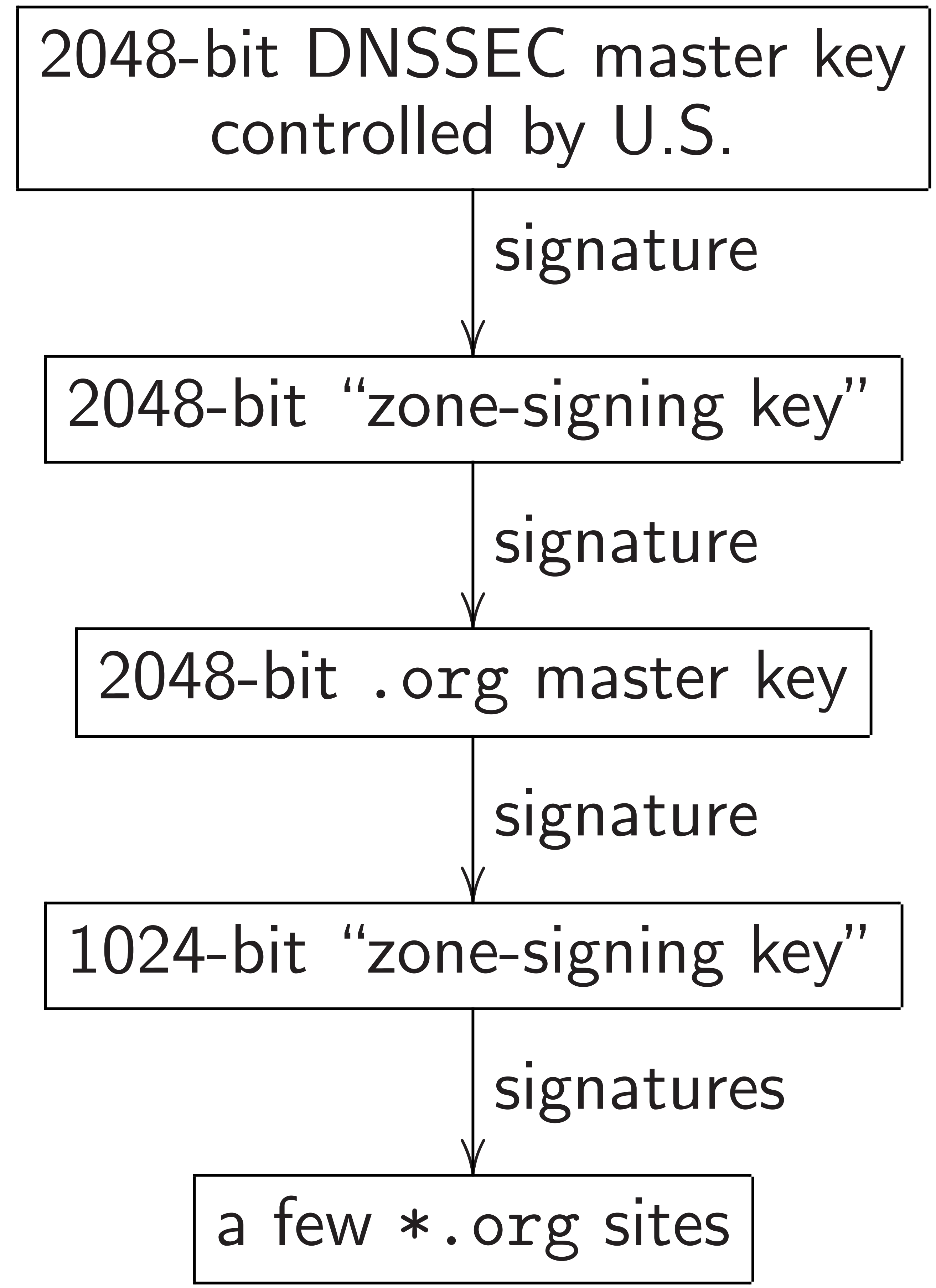
DNSSEC key sizes, 2016.11.28:



Applications pursue speed

e.g. Latest "DNSSEC operational practices" recommendation (2012) says "No one has broken a regular 1024-bit [RSA] key . . . it is estimated that most zones can safely use 1024-bit keys for at least the next ten years . . . Signing and verifying with a 2048-bit key takes longer than with a 1024-bit key . . . public operations (such as verification) are about four times slower."

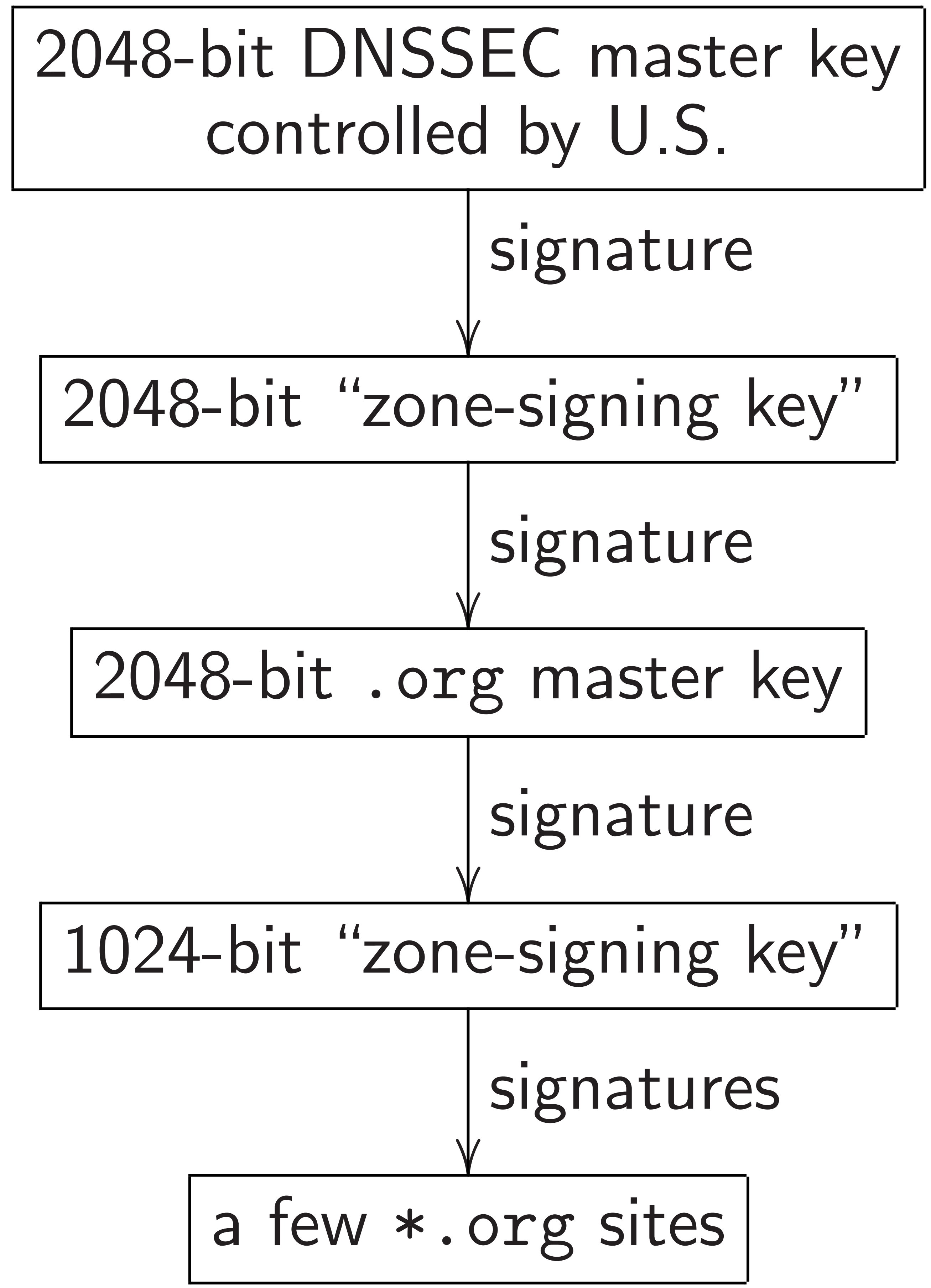
DNSSEC key sizes, 2016.11.28:



ions pursue speed

est “DNSSEC operational
s” recommendation
says “No one has broken
r 1024-bit [RSA] key . . .
mated that most zones
ly use 1024-bit keys for
the next ten years . . .
and verifying with a 2048-
takes longer than with a
key . . . public operations
verification) are about
es slower.”

DNSSEC key sizes, 2016.11.28:



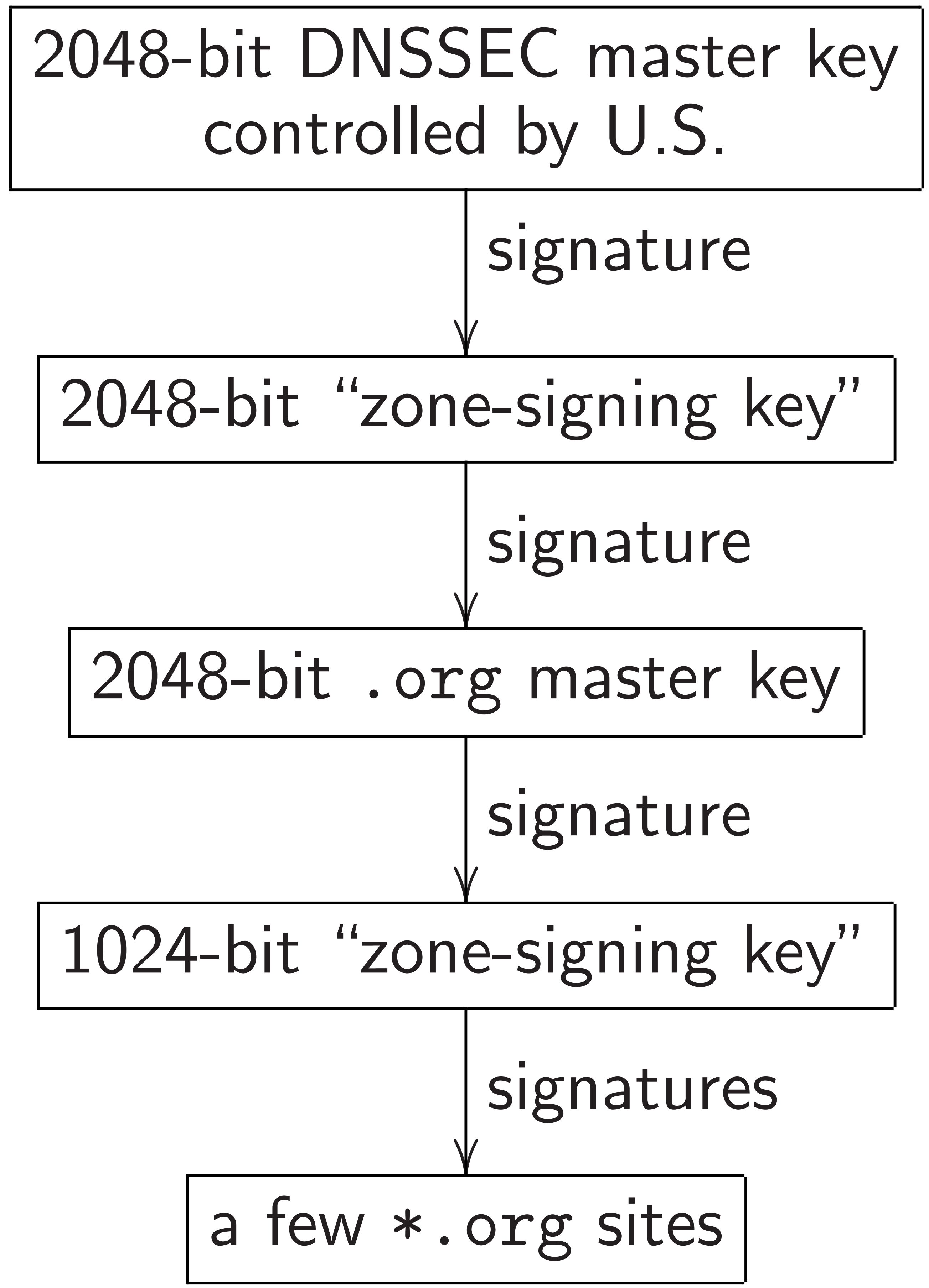
2011 We
security

“V2V sa
broadcas
second,
1,000 or
second.
available
amount
message
an actua
are proc
security
message

ue speed

SEC operational
nendation
one has broken
[RSA] key . . .
t most zones
4-bit keys for
en years . . .
ing with a 2048-
er than with a
ublic operations
on) are about
’

DNSSEC key sizes, 2016.11.28:



2011 Weimerskirch
security for car co
“V2V safety applic
broadcast 10 mess
second, and a vehi
1,000 or more mes
second. There are
available to proces
amount of messag
messages that mig
an actual impact t
are processed, or (
security hardware
messages is applie

tional

oken

y . . .

nes

for

. .

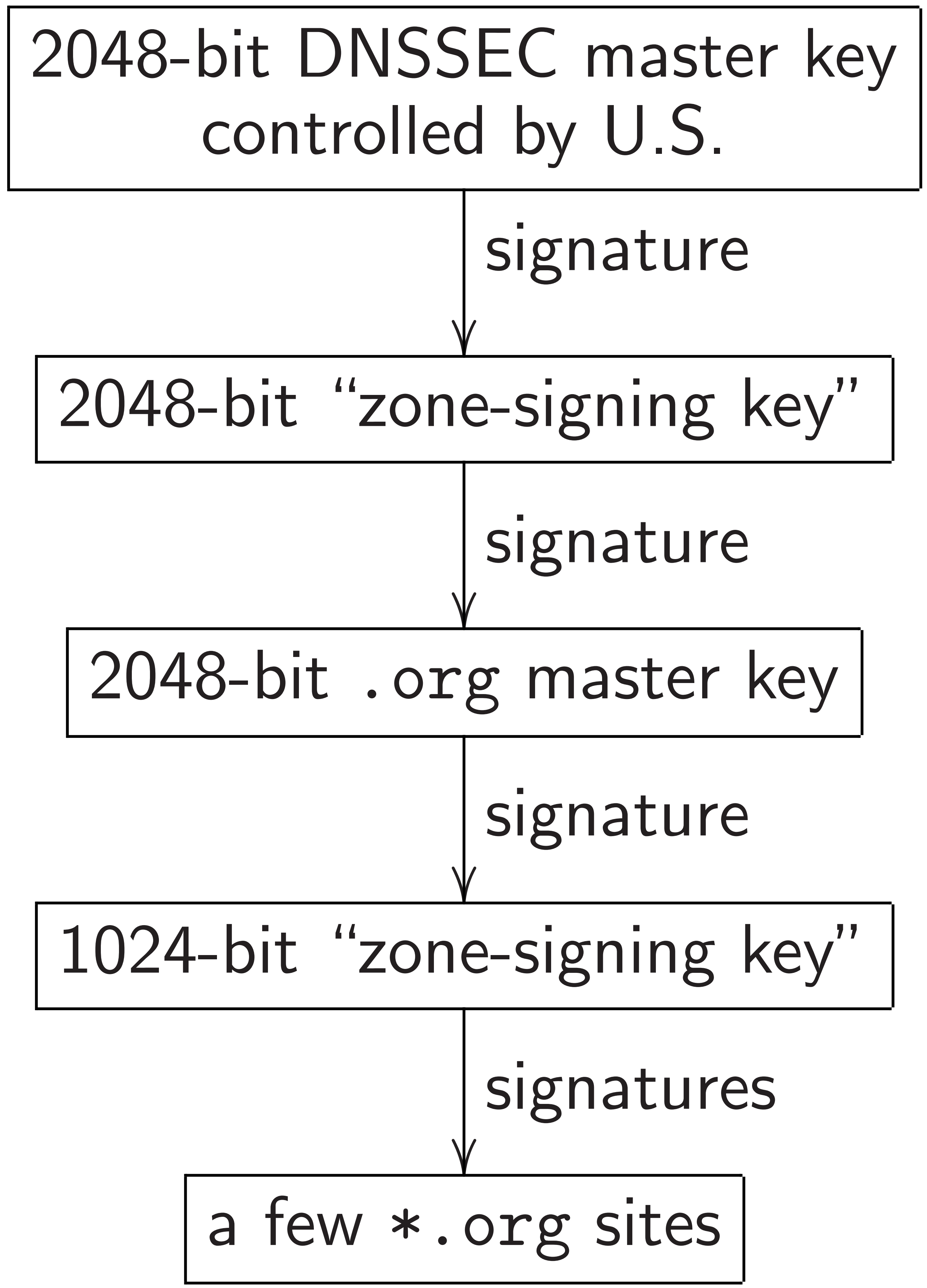
2048-

th a

rations

out

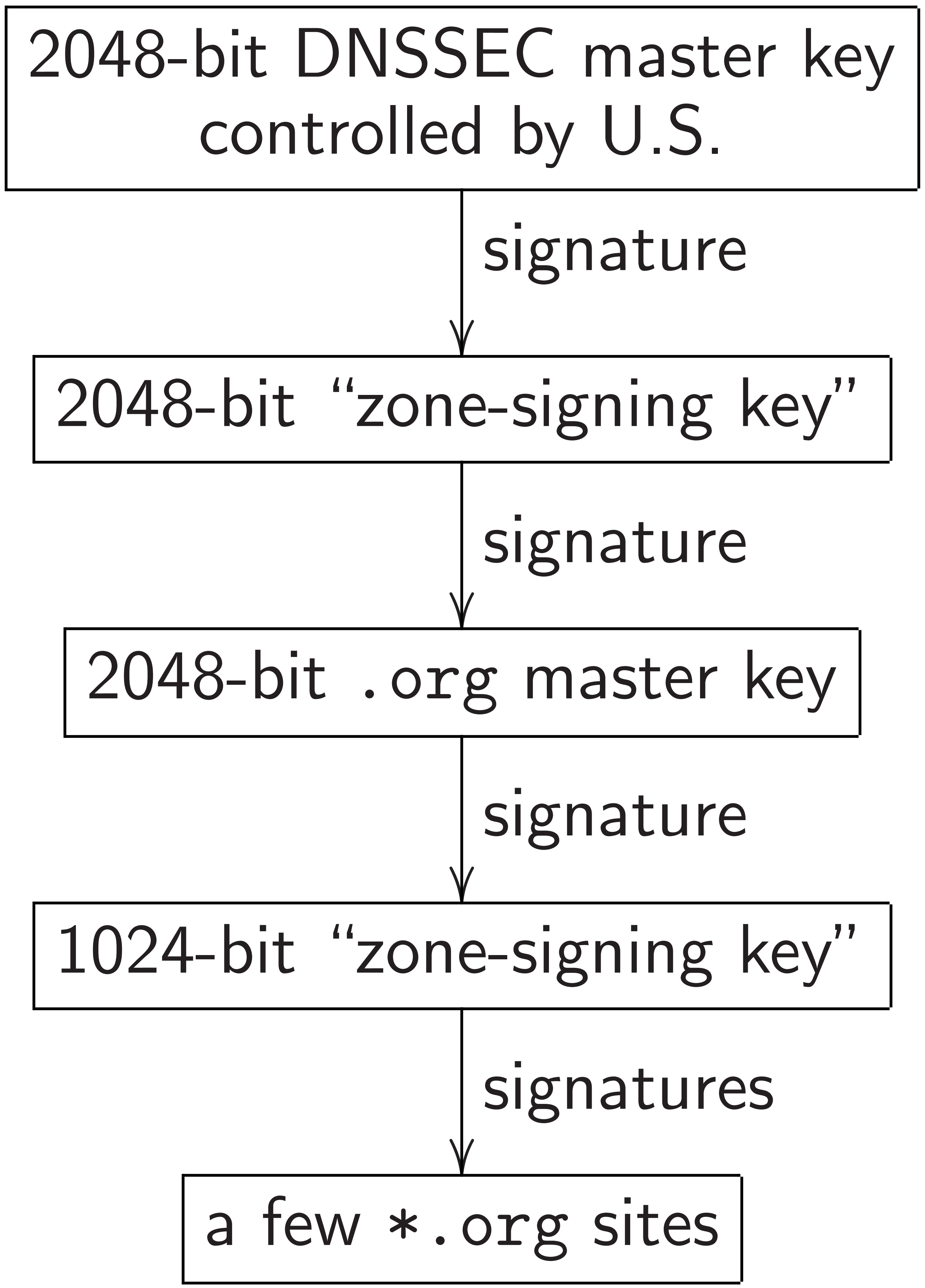
DNSSEC key sizes, 2016.11.28:



2011 Weimerskirch survey o
security for car communicati

"V2V safety applications wil
broadcast 10 messages per
second, and a vehicle will re
1,000 or more messages per
second. There are two appro
available to process such a h
amount of messages: (1) on
messages that might impose
an actual impact to a vehicl
are processed, or (2) dedicat
security hardware to process
messages is applied."

DNSSEC key sizes, 2016.11.28:



2011 Weimerskirch survey of security for car communication:

"V2V safety applications will broadcast 10 messages per second, and a vehicle will receive 1,000 or more messages per second. There are two approaches available to process such a high amount of messages: (1) only messages that might impose an actual impact to a vehicle are processed, or (2) dedicated security hardware to process all messages is applied."

C key sizes, 2016.11.28:

t DNSSEC master key
ontrolled by U.S.

↓ signature

it "zone-signing key"

↓ signature

bit .org master key

↓ signature

it "zone-signing key"

↓ signatures

few *.org sites

2011 Weimerskirch survey of security for car communication:

"V2V safety applications will broadcast 10 messages per second, and a vehicle will receive 1,000 or more messages per second. There are two approaches available to process such a high amount of messages: (1) only messages that might impose an actual impact to a vehicle are processed, or (2) dedicated security hardware to process all messages is applied."

2014 Gh
Pullini–M
"A lightw
system f
biosenso
the recen
Keccak s
impleme
encryptio
the newl
scheme,
large am
testing p
standard

s, 2016.11.28:

C master key
y U.S.

gnature

igning key”

gnature

master key

gnature

igning key”

gnatures

g sites

2011 Weimerskirch survey of security for car communication:

“V2V safety applications will broadcast 10 messages per second, and a vehicle will receive 1,000 or more messages per second. There are two approaches available to process such a high amount of messages: (1) only messages that might impose an actual impact to a vehicle are processed, or (2) dedicated security hardware to process all messages is applied.”

2014 Ghoreishizad

Pullini–Micheli–Bu

“A lightweight cry

system for implant

biosensors”: “This

the recently stand

Keccak secure has

implemented in an

encryption mode .

the newly standard

scheme, we benefi

large amount of ar

testing performed

standardization pr

.28:

ey

y"

y

y"

2011 Weimerskirch survey of security for car communication:

"V2V safety applications will broadcast 10 messages per second, and a vehicle will receive 1,000 or more messages per second. There are two approaches available to process such a high amount of messages: (1) only messages that might impose an actual impact to a vehicle are processed, or (2) dedicated security hardware to process all messages is applied."

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Ca

"A lightweight cryptographi system for implantable biosensors": "This design us the recently standardized SH Keccak secure hash function implemented in an authentic encryption mode . . . By sele the newly standardized Kecc scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

2011 Weimerskirch survey of security for car communication:

"V2V safety applications will broadcast 10 messages per second, and a vehicle will receive 1,000 or more messages per second. There are two approaches available to process such a high amount of messages: (1) only messages that might impose an actual impact to a vehicle are processed, or (2) dedicated security hardware to process all messages is applied."

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode . . . By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

eimerskirch survey of
for car communication:

fety applications will
st 10 messages per
and a vehicle will receive
more messages per
There are two approaches
to process such a high
of messages: (1) only
s that might impose
l impact to a vehicle
essed, or (2) dedicated
hardware to process all
s is applied."

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode . . . By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

we have
of round
guarante
the Kecc

n survey of
mmunication:

cations will
sages per
icle will receive
ssages per
two approaches
ss such a high
es: (1) only
ght impose
to a vehicle
(2) dedicated
to process all
d."

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode . . . By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

we have used the
of rounds for all in
guarantee the secu
the Keccak propos

f
ion:
l
ceive
paches
high
nly
e
e
ted
all

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode . . . By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

we have used the same num
of rounds for all in order to
guarantee the security claim
the Keccak proposal.

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode ... By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. ...

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal.

2014 Ghoreishizadeh–Yalcin–Pullini–Micheli–Burleson–Carrara "A lightweight cryptographic system for implantable biosensors": "This design uses the recently standardized SHA-3 Keccak secure hash function implemented in an authenticated encryption mode . . . By selecting the newly standardized Keccak scheme, we benefit from the large amount of analysis and testing performed during the standardization process. . . .

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively."

oreishizadeh–Yalcin–
Micheli–Burleson–Carrara
weight cryptographic
for implantable
rs": "This design uses
ntly standardized SHA-3
secure hash function
nted in an authenticated
on mode . . . By selecting
ly standardized Keccak
we benefit from the
ount of analysis and
performed during the
lization process. . . .

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively."

Standard

e.g. NIS

"Securit
factor in
Rijndael
adequat
Serpent
high sec

(Empha

So why

eh–Yalcin–
urleson–Carrara
ptographic
table
s design uses
ardized SHA-3
h function
authenticated
. . By selecting
dized Keccak
t from the
nalysis and
during the
ocess. . . .

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively."

Standards pursue

e.g. NIST's final A

"Security was the
factor in the evalu
Rijndael appears t
adequate security
Serpent appears to
high security marg

(Emphasis added.)

So why didn't Ser

–
rrara
c

ses
HA-3
n
cated
ecting
cak
e
d
e

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively.”

Standards pursue speed

e.g. NIST’s final AES report

“Security was the most imp
factor in the evaluation . . .
Rijndael appears to offer an
adequate security margin. . .
Serpent appears to offer a
high security margin.”

(Emphasis added.)

So why didn’t Serpent win?

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

we have used the same number of rounds for all in order to guarantee the security claim of the Keccak proposal. However, instead of using the standard sizes for bitrate and capacity, we reduced the overall state size in order to achieve a compact implementation with a security level that would not have been possible at this cost with any other authenticated encryption scheme. The data block size and state size are selected as 4 and 100 bits, respectively."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

used the same number
ls for all in order to
ee the security claim of
cak proposal. However,
of using the standard
bitrate and capacity,
ced the overall state size
to achieve a compact
ntation with a security
t would not have been
at this cost with any
thenticated encryption
The data block size and
e are selected as 4 and
, respectively."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . .
Rijndael appears to offer an *adequate* security margin. . . .
Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The op
are amo
against t

same number
order to
urity claim of
sal. However,
he standard
nd capacity,
erall state size
e a compact
ith a security
ot have been
st with any
ed encryption
block size and
cted as 4 and
ely."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The operations u
are among the eas
against timing and

ber

of
ver,
d
y,
size
ct
rity
een
y
ion
e and
nd

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . .
Rijndael appears to offer an *adequate* security margin. . . .
Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The operations used by Ser
are among the easiest to def
against timing and power at

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Standards pursue speed

e.g. NIST's final AES report:

"Security was the most important factor in the evaluation . . . Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

(Emphasis added.)

So why didn't Serpent win?

Maybe side-channel security?

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Great! Why didn't Serpent win?

ds pursue speed

T's final AES report:

y was the most important
the evaluation . . .
appears to offer an
e security margin. . . .
appears to offer a
urity margin."

sis added.)

didn't Serpent win?

ide-channel security?

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Great! Why didn't Serpent win?

Aha: So

speed

AES report:

most important

ation . . .

o offer an

margin. . . .

o offer a

gin.”

)

pent win?

el security?

“The operations used by Serpent are among the easiest to defend against timing and power attacks.”

Hardware speed: “Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent’s speed is independent of key size.”

Great! Why didn’t Serpent win?

Aha: Software spe

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Great! Why didn't Serpent win?

Aha: Software speed!

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Great! Why didn't Serpent win?

Aha: Software speed!

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Great! Why didn't Serpent win?

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

"The operations used by Serpent are among the easiest to defend against timing and power attacks."

Hardware speed: "Serpent is well suited to restricted-space environments . . . Fully pipelined implementations of Serpent offer the highest throughput of any of the finalists for non-feedback modes. . . . Efficiency is generally very good, and Serpent's speed is independent of key size."

Great! Why didn't Serpent win?

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

Conclusion: "NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism]."

perations used by Serpent
ng the easiest to defend
timing and power attacks."

e speed: "Serpent is
ed to restricted-space
ments . . . Fully pipelined
entations of Serpent offer
est throughput of any
nalists for non-feedback
. . . Efficiency is generally
od, and Serpent's speed is
dent of key size."

Why didn't Serpent win?

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

Conclusion: "NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism]."

Want fa

Bad exa
The pur
damages
e.g. usin
e.g. usin
e.g. skip

sed by Serpent
siest to defend
d power attacks."

"Serpent is
ricted-space
Fully pipelined
of Serpent offer
ghput of any
non-feedback
ncy is generally
rpent's speed is
y size."

t Serpent win?

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

Conclusion: "NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism]."

Want fast *and* sec

Bad examples:

The pursuit of spe
damages security.

e.g. using 1024-bit
e.g. using 100-bit
e.g. skipping verifi

rpent
fend
tacks.”

s
ce
lined
offer
ny
ack
erally
eed is

win?

Aha: Software speed! “Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance.”

Conclusion: “NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism].”

Want fast *and* secure

Bad examples:
The pursuit of speed damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit “SHA-3”.
e.g. skipping verification.

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

Conclusion: "NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism]."

Want fast *and* secure

Bad examples:
The pursuit of speed damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Aha: Software speed! "Serpent is generally the slowest of the finalists in software speed for encryption and decryption. . . . Serpent provides consistently low-end performance."

Conclusion: "NIST judged Rijndael to be the best overall algorithm for the AES. Rijndael appears to be consistently a very good performer in both hardware and software [and offers good key agility, low memory, easy defense, fast defense, flexibility, parallelism]."

Want fast *and* secure

Bad examples:
The pursuit of speed damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed *without* damaging security.

If security level was too low, scale up: better security for the same performance.

ftware speed! "Serpent
ally the slowest of the
in software speed for
on and decryption. . . .
provides consistently
performance."

on: "NIST judged
to be the best overall
n for the AES. Rijndael
to be consistently a very
rformer in both hardware
ware [and offers good
ty, low memory, easy
fast defense, flexibility,
sm]."

Want fast *and* secure

Bad examples:
The pursuit of speed
damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed
without damaging security.

If security level was too low,
scale up: better security
for the same performance.

Success

Extensiv
ECC at
⇒ mode
for pract

Requires
and opti
Not just
not just

eed! "Serpent
owest of the
e speed for
cryption. . . .
consistently
nce."

T judged
best overall
AES. Rijndael
sistently a very
both hardware
offers good
emory, easy
nse, flexibility,

Want fast *and* secure

Bad examples:
The pursuit of speed damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed *without* damaging security.

If security level was too low, scale up: better security for the same performance.

Success story: EC

Extensive work on
ECC at a high sec
⇒ modern ECC is
for practically all a

Requires serious ar
and optimization o
Not just "polynom
not just "quadrati

pent
ne
r
. .
y

all
dael
very
dware
od
y
lity,

Want fast *and* secure

Bad examples:
The pursuit of speed
damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed
without damaging security.

If security level was too low,
scale up: better security
for the same performance.

Success story: ECC.

Extensive work on speed of
ECC at a high security level
⇒ modern ECC is fast enou
for practically all application

Requires serious analysis
and optimization of algorith
Not just "polynomial time";
not just "quadratic time".

Want fast *and* secure

Bad examples:
The pursuit of speed damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed *without* damaging security.

If security level was too low, scale up: better security for the same performance.

Success story: ECC.

Extensive work on speed of ECC at a high security level $\Rightarrow$ modern ECC is fast enough for practically all applications.

Requires serious analysis and optimization of algorithms.
Not just "polynomial time"; not just "quadratic time".

Want fast *and* secure

Bad examples:
The pursuit of speed
damages security.
e.g. using 1024-bit RSA.
e.g. using 100-bit "SHA-3".
e.g. skipping verification.

Good examples:
Obtain better speed
without damaging security.

If security level was too low,
scale up: better security
for the same performance.

Success story: ECC.

Extensive work on speed of
ECC at a high security level
$\Rightarrow$ modern ECC is fast enough
for practically all applications.

Requires serious analysis
and optimization of algorithms.
Not just "polynomial time";
not just "quadratic time".

RSA and Rabin–Williams are even
faster for signature verification,
but slower for keygen, signing,
sending keys, sending sigs.

st *and* secure

mples:
suit of speed
s security.
g 1024-bit RSA.
g 100-bit "SHA-3".
ping verification.

amples:
better speed
damaging security.

ty level was too low,
: better security
same performance.

Success story: ECC.

Extensive work on speed of ECC at a high security level ⇒ modern ECC is fast enough for practically all applications.

Requires serious analysis and optimization of algorithms. Not just "polynomial time"; not just "quadratic time".

RSA and Rabin–Williams are even faster for signature verification, but slower for keygen, signing, sending keys, sending sigs.

Some sig

1985 ElG

1990 Sch
plus vari
Patented

1991 DS
later cre
with one

1999 EC
DSA wit

2011 Ed
Schnorr

cure

eed

t RSA.
"SHA-3".
cation.

ed
security.

s too low,
ecurity
rmance.

Success story: ECC.

Extensive work on speed of
ECC at a high security level
⇒ modern ECC is fast enough
for practically all applications.

Requires serious analysis
and optimization of algorithms.
Not just "polynomial time";
not just "quadratic time".

RSA and Rabin–Williams are even
faster for signature verification,
but slower for keygen, signing,
sending keys, sending sigs.

Some signature-sy

1985 ElGamal: $\mathbf{F}_p^*$

1990 Schnorr: ElG
plus various impro
Patented until 200

1991 DSA, annour
later credited to N
with one Schnorr i

1999 ECDSA: repl
DSA with an ellipt

2011 EdDSA (e.g.
Schnorr plus more

Success story: ECC.

Extensive work on speed of ECC at a high security level $\Rightarrow$ modern ECC is fast enough for practically all applications.

Requires serious analysis and optimization of algorithms. Not just "polynomial time"; not just "quadratic time".

RSA and Rabin–Williams are even faster for signature verification, but slower for keygen, signing, sending keys, sending sigs.

Some signature-system histo

1985 ElGamal: $\mathbf{F}_p^*$ signature

1990 Schnorr: ElGamal plus various improvements. Patented until 2008.

1991 DSA, announced by N later credited to NSA: ElGa with one Schnorr improveme

1999 ECDSA: replacing $\mathbf{F}_p^*$ i DSA with an elliptic-curve g

2011 EdDSA (e.g., Ed2551 Schnorr plus more improvem

Success story: ECC.

Extensive work on speed of ECC at a high security level $\Rightarrow$ modern ECC is fast enough for practically all applications.

Requires serious analysis and optimization of algorithms. Not just "polynomial time"; not just "quadratic time".

RSA and Rabin–Williams are even faster for signature verification, but slower for keygen, signing, sending keys, sending sigs.

Some signature-system history

1985 ElGamal: $\mathbf{F}_p^*$ signatures.

1990 Schnorr: ElGamal plus various improvements. Patented until 2008.

1991 DSA, announced by NIST, later credited to NSA: ElGamal with one Schnorr improvement.

1999 ECDSA: replacing $\mathbf{F}_p^*$ in DSA with an elliptic-curve group.

2011 EdDSA (e.g., Ed25519): Schnorr plus more improvements.

story: ECC.

e work on speed of
a high security level
ern ECC is fast enough
tically all applications.

s serious analysis
mization of algorithms.
"polynomial time";
"quadratic time".

d Rabin–Williams are even
r signature verification,
er for keygen, signing,
keys, sending sigs.

Some signature-system history

1985 ElGamal: $\mathbf{F}_p^*$ signatures.

1990 Schnorr: ElGamal
plus various improvements.
Patented until 2008.

1991 DSA, announced by NIST,
later credited to NSA: ElGamal
with one Schnorr improvement.

1999 ECDSA: replacing $\mathbf{F}_p^*$ in
DSA with an elliptic-curve group.

2011 EdDSA (e.g., Ed25519):
Schnorr plus more improvements.

ElGamal
(R, S) is
if $B^{H(M)}$
and R, S

Here p i
B is sta
A is sig
$H(M)$ is

Secret k
Public k
To sign
compute
$S = r^{-1}$

C.

speed of
urity level
fast enough
applications.

nalysis
of algorithms.
nial time";
c time".

Villiams are even
e verification,
gen, signing,
ling sigs.

Some signature-system history

1985 ElGamal: $\mathbf{F}_p^*$ signatures.

1990 Schnorr: ElGamal
plus various improvements.
Patented until 2008.

1991 DSA, announced by NIST,
later credited to NSA: ElGamal
with one Schnorr improvement.

1999 ECDSA: replacing $\mathbf{F}_p^*$ in
DSA with an elliptic-curve group.

2011 EdDSA (e.g., Ed25519):
Schnorr plus more improvements.

ElGamal verificatio
(R, S) is signature
if $B^{H(M)} \equiv A^R R^S$
and $R, S \in \{0, 1, .$

Here p is standard
B is standard base
A is signer's public
$H(M)$ is hash of m

Secret key: randon
Public key: $A = B$
To sign M: genera
compute $R = B^r$ m
$S = r^{-1}(H(M) -$

gh
s.
ms.
e even
on,
g,

Some signature-system history

1985 ElGamal: $\mathbf{F}_p^*$ signatures.

1990 Schnorr: ElGamal
plus various improvements.
Patented until 2008.

1991 DSA, announced by NIST,
later credited to NSA: ElGamal
with one Schnorr improvement.

1999 ECDSA: replacing $\mathbf{F}_p^*$ in
DSA with an elliptic-curve group.

2011 EdDSA (e.g., Ed25519):
Schnorr plus more improvements.

ElGamal verification:
(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod p$
and $R, S \in \{0, 1, \ldots, p-2\}$

Here p is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate randon
compute $R = B^r \bmod p$,
$S = r^{-1}(H(M) - aR)$ mod

Some signature-system history

1985 ElGamal: $\mathbf{F}_p^*$ signatures.

1990 Schnorr: ElGamal
plus various improvements.
Patented until 2008.

1991 DSA, announced by NIST,
later credited to NSA: ElGamal
with one Schnorr improvement.

1999 ECDSA: replacing $\mathbf{F}_p^*$ in
DSA with an elliptic-curve group.

2011 EdDSA (e.g., Ed25519):
Schnorr plus more improvements.

ElGamal verification:
(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod{p}$
and $R, S \in \{0, 1, \ldots, p-2\}$.

Here p is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate random r,
compute $R = B^r \bmod p$,
$S = r^{-1}(H(M) - aR) \bmod p - 1$.

gnature-system history

Gamal: $\mathbf{F}_p^*$ signatures.

hnorr: ElGamal
ous improvements.
d until 2008.

SA, announced by NIST,
dited to NSA: ElGamal
e Schnorr improvement.

CDSA: replacing $\mathbf{F}_p^*$ in
th an elliptic-curve group.

DSA (e.g., Ed25519):
plus more improvements.

ElGamal verification:
(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod p$
and $R, S \in \{0, 1, \ldots, p-2\}$.

Here p is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate random r,
compute $R = B^r \bmod p$,
$S = r^{-1}(H(M) - aR) \bmod p - 1$.

Hash the

Tweak:
if $B^{H(M)}$
and R, S

Signer:
$r^{-1}(H(M$

Speed in
Hashing

Security
serious c
strategy
a particu

stem history

, signatures.

Gamal
vements.
08.

nced by NIST,
SA: ElGamal
improvement.

lacing $\mathbf{F}_p^*$ in
tic-curve group.

, Ed25519):
improvements.

ElGamal verification:

(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod{p}$
and $R, S \in \{0, 1, \ldots, p-2\}$.

Here p is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate random r,
compute $R = B^r \bmod p$,
$S = r^{-1}(H(M) - aR) \bmod p - 1$.

Hash the exponent

Tweak: (R, S) is s
if $B^{H(M)} \equiv A^{H(R)}$
and $R, S \in \{0, 1, .$

Signer: as before e
$r^{-1}(H(M) - aH($

Speed impact: neg
Hashing R is very

Security impact: s
serious obstacle to
strategy that relies
a particular A expo

ory

es.

IST,

mal

ent.

in

group.

)):

ents.

ElGamal verification:

(R, S) is signature of M
if $B^{H(M)} \equiv A^R R^S \pmod{p}$
and $R, S \in \{0, 1, \ldots, p-2\}$.

Here p is standard prime,
B is standard base,
A is signer's public key,
$H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate random r,
compute $R = B^r \bmod p$,
$S = r^{-1}(H(M) - aR) \bmod p - 1$.

Hash the exponent

Tweak: (R, S) is signature o
if $B^{H(M)} \equiv A^{H(R)} R^S$ (moc
and $R, S \in \{0, 1, \ldots, p-2\}$

Signer: as before except S =
$r^{-1}(H(M) - aH(R)) \bmod p$

Speed impact: negligible.
Hashing R is very fast.

Security impact: seems to b
serious obstacle to any attac
strategy that relies on choos
a particular A exponent.

ElGamal verification:

(R, S) is signature of M if $B^{H(M)} \equiv A^R R^S \pmod{p}$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Here p is standard prime, B is standard base, A is signer's public key, $H(M)$ is hash of message.

Secret key: random a.
Public key: $A = B^a \bmod p$.
To sign M: generate random r, compute $R = B^r \bmod p$, $S = r^{-1}(H(M) - aR) \bmod p - 1$.

Hash the exponent

Tweak: (R, S) is signature of M if $B^{H(M)} \equiv A^{H(R)} R^S \pmod{p}$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Signer: as before except $S = r^{-1}(H(M) - aH(R)) \bmod p - 1$.

Speed impact: negligible.
Hashing R is very fast.

Security impact: seems to be serious obstacle to any attack strategy that relies on choosing a particular A exponent.

verification:

s signature of M

$) \equiv A^R R^S \pmod p$

$S \in \{0, 1, \ldots, p-2\}$.

s standard prime,

ndard base,

ner's public key,

s hash of message.

ey: random a.

ey: $A = B^a \bmod p$.

M: generate random r,

e $R = B^r \bmod p$,

$(H(M) - aR) \bmod p - 1$.

Hash the exponent

Tweak: (R, S) is signature of M if $B^{H(M)} \equiv A^{H(R)} R^S \pmod p$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Signer: as before except $S = r^{-1}(H(M) - aH(R)) \bmod p - 1$.

Speed impact: negligible. Hashing R is very fast.

Security impact: seems to be serious obstacle to any attack strategy that relies on choosing a particular A exponent.

Prime-or

Choose

standard

e.g. take

Again ve

ECC: $H($

Signer:

$S = r^{-1}$

Simpler

Speed a

(when q

Less tim

on:
of M
(mod p)
$\ldots, p-2\}$.
prime,
e,
c key,
message.
m a.
$B^a \bmod p$.
ate random r,
mod p,
$aR) \bmod p-1$.

Hash the exponent

Tweak: (R, S) is signature of M if $B^{H(M)} \equiv A^{H(R)} R^S \pmod p$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Signer: as before except $S = r^{-1}(H(M) - aH(R)) \bmod p-1$.

Speed impact: negligible. Hashing R is very fast.

Security impact: seems to be serious obstacle to any attack strategy that relies on choosing a particular A exponent.

Prime-order subgr

Choose B to have
standard prime div
e.g. take 3000-bit

Again verify $B^{H(M}$
ECC: $H(M)B = H$

Signer: same exce
$S = r^{-1}(H(M) -$

Simpler security a

Speed advantage:
(when q is smalle
Less time to trans

)

.

n r,

$p - 1$.

Hash the exponent

Tweak: (R, S) is signature of M if $B^{H(M)} \equiv A^{H(R)} R^S \pmod p$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Signer: as before except $S = r^{-1}(H(M) - aH(R)) \bmod p - 1$.

Speed impact: negligible.
Hashing R is very fast.

Security impact: seems to be serious obstacle to any attack strategy that relies on choosing a particular A exponent.

Prime-order subgroup

Choose B to have order q fo
standard prime divisor q of
e.g. take 3000-bit p, 256-bit

Again verify $B^{H(M)} \equiv A^{H(R}$
ECC: $H(M)B = H(R)A + S$

Signer: same except now
$S = r^{-1}(H(M) - aH(R))$ m

Simpler security analysis.

Speed advantage: Smaller S
(when q is smaller than $p -$
Less time to transmit signat

Hash the exponent

Tweak: (R, S) is signature of M if $B^{H(M)} \equiv A^{H(R)} R^S \pmod p$ and $R, S \in \{0, 1, \ldots, p-2\}$.

Signer: as before except $S = r^{-1}(H(M) - aH(R)) \bmod p - 1$.

Speed impact: negligible. Hashing R is very fast.

Security impact: seems to be serious obstacle to any attack strategy that relies on choosing a particular A exponent.

Prime-order subgroup

Choose B to have order q for standard prime divisor q of $p-1$. e.g. take 3000-bit p, 256-bit q.

Again verify $B^{H(M)} \equiv A^{H(R)} R^S$. ECC: $H(M)B = H(R)A + SR$.

Signer: same except now $S = r^{-1}(H(M) - aH(R)) \bmod q$.

Simpler security analysis.

Speed advantage: Smaller S (when q is smaller than $p-1$). Less time to transmit signature.

e exponent

(R, S) is signature of M
$) \equiv A^{H(R)} R^S \pmod{p}$
$S \in \{0, 1, \ldots, p-2\}$.

as before except $S =$
$M) - aH(R))$ mod $p - 1$.

mpact: negligible.
R is very fast.

impact: seems to be
obstacle to any attack
that relies on choosing
ular A exponent.

Prime-order subgroup

Choose B to have order q for standard prime divisor q of $p - 1$. e.g. take 3000-bit p, 256-bit q.

Again verify $B^{H(M)} \equiv A^{H(R)} R^S$.
ECC: $H(M)B = H(R)A + SR$.

Signer: same except now
$S = r^{-1}(H(M) - aH(R))$ mod q.

Simpler security analysis.

Speed advantage: Smaller S (when q is smaller than $p - 1$). Less time to transmit signature.

Two sca

Verify B
A
ECC: $(H$
A

Safe to a
ever finc

No secur
if $B^{H(R)}$
then B^{H}

Speed ac
outweigh

t

signature of M

$R^S \pmod{p}$

$\ldots, p-2\}$.

except $S =$

$R))$ mod $p-1$.

gligible.

fast.

seems to be

any attack

s on choosing

onent.

Prime-order subgroup

Choose B to have order q for standard prime divisor q of $p-1$. e.g. take 3000-bit p, 256-bit q.

Again verify $B^{H(M)} \equiv A^{H(R)} R^S$.
ECC: $H(M)B = H(R)A + SR$.

Signer: same except now
$S = r^{-1}(H(M) - aH(R)) \bmod q$.

Simpler security analysis.

Speed advantage: Smaller S (when q is smaller than $p-1$).
Less time to transmit signature.

Two scalars

Verify $B^{H(R)^{-1}H(M}$

$AR^{H(R)^{-1}S}$

ECC: $(H(R)^{-1}H($

$A + (H(R)^{-}$

Safe to assume th

ever find $H(R)$ div

No security loss:
if $B^{H(R)^{-1}H(M)} =$
then $B^{H(M)} = A^{H}$

Speed advantage:
outweighing cost o

of M
d p)
.
=
$- 1$.
e
ck
sing

Prime-order subgroup

Choose B to have order q for standard prime divisor q of $p - 1$. e.g. take 3000-bit p, 256-bit q.

Again verify $B^{H(M)} = A^{H(R)} R^S$.
ECC: $H(M)B = H(R)A + SR$.

Signer: same except now
$S = r^{-1}(H(M) - aH(R)) \bmod q$.

Simpler security analysis.

Speed advantage: Smaller S (when q is smaller than $p - 1$).
Less time to transmit signature.

Two scalars

Verify $B^{H(R)^{-1}H(M)} =$
$AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B =$
$A + (H(R)^{-1}S)R$.

Safe to assume that nobody
ever find $H(R)$ divisible by

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-}}$
then $B^{H(M)} = A^{H(R)} R^S$.

Speed advantage: fewer sca
outweighing cost of $H(R)^{-1}$

Prime-order subgroup

Choose B to have order q for standard prime divisor q of $p-1$. e.g. take 3000-bit p, 256-bit q.

Again verify $B^{H(M)} \equiv A^{H(R)} R^S$.
ECC: $H(M)B = H(R)A + SR$.

Signer: same except now
$S = r^{-1}(H(M) - aH(R)) \bmod q$.

Simpler security analysis.

Speed advantage: Smaller S (when q is smaller than $p-1$).
Less time to transmit signature.

Two scalars

Verify $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B = A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^S$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

rder subgroup

B to have order q for
l prime divisor q of $p-1$.
e 3000-bit p, 256-bit q.

erify $B^{H(M)} \equiv A^{H(R)} R^S$.
$(M)B = H(R)A + SR$.

same except now
$(H(M) - aH(R))$ mod q.

security analysis.

dvantage: Smaller S
is smaller than $p-1$).
e to transmit signature.

Two scalars

Verify $B^{H(R)^{-1}H(M)} =$
$AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B =$
$A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^S$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

Precomp

Notatio

Send $(R$
signatur
signer in

Verify B
ECC: $(H$

Signer c
$r^{-1}(H($

oup

order q for
visor q of $p-1$.
p, 256-bit q.

$)\equiv A^{H(R)}R^{S}$.
$H(R)A+SR$.

pt now
$aH(R))$ mod q.

nalysis.

Smaller S
than $p-1$).
mit signature.

Two scalars

Verify $B^{H(R)^{-1}H(M)} =$
$AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B =$
$A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^{S}$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

Precomputing quo

Notation: $\underline{S} = H($

Send $(R, \underline{S})$ instea
signature: i.e., $\underline{S}$ c
signer instead of v

Verify $B^{H(R)^{-1}H(M}$
ECC: $(H(R)^{-1}H($

Signer computes $\underline{S}$
$r^{-1}(H(R)^{-1}H(M)$

or

$p - 1$.

q.

$)R^S$.

SR.

od q.

S

1).

ure.

Two scalars

Verify $B^{H(R)^{-1}H(M)} =$
$AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B =$
$A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^S$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of $(R, S$
signature: i.e., $\underline{S}$ computed
signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$
ECC: $(H(R)^{-1}H(M))B = A$

Signer computes $\underline{S} =$
$r^{-1}(H(R)^{-1}H(M) - a)$ mod

Two scalars

Verify $B^{H(R)^{-1}H(M)} =$
$AR^{H(R)^{-1}S}$.
ECC: $(H(R)^{-1}H(M))B =$
$A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^{S}$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A + \underline{S}R$.

Signer computes $\underline{S} =$
$r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

Two scalars

Verify $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$.

ECC: $(H(R)^{-1}H(M))B = A + (H(R)^{-1}S)R$.

Safe to assume that nobody will ever find $H(R)$ divisible by q.

No security loss:
if $B^{H(R)^{-1}H(M)} = AR^{H(R)^{-1}S}$
then $B^{H(M)} = A^{H(R)}R^{S}$.

Speed advantage: fewer scalars, outweighing cost of $H(R)^{-1}$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A + \underline{S}R$.

Signer computes $\underline{S} = r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

From now on: Rename $\underline{S}$ as S.

lars

$^{H(R)^{-1}H(M)} =$

$R^{H(R)^{-1}S}$.

$(R)^{-1}H(M))B =$

$+ (H(R)^{-1}S)R$.

assume that nobody will

l $H(R)$ divisible by q.

rity loss:

$^{-1}H(M) = AR^{H(R)^{-1}S}$

$^{(M)} = A^{H(R)}R^{S}$.

dvantage: fewer scalars,

hing cost of $H(R)^{-1}$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A+\underline{S}R$.

Signer computes $\underline{S} =$
$r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

From now on: Rename $\underline{S}$ as S.

Merge h

$B^{H(R,M)}$

ECC: $H($

Speed a

is faster

Security

attacker

innocent

with $H($

Using H

signs M

same sig

Using H

$M) =$

.

$M))B =$
$^{1}S)R$.

at nobody will
visible by q.

$AR^{H(R)^{-1}S}$
$(R)R^S$.

fewer scalars,
of $H(R)^{-1}$.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A + \underline{S}R$.

Signer computes $\underline{S} =$
$r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

From now on: Rename $\underline{S}$ as S.

Merge hashes: col

$B^{H(R,M)} = AR^S$.
ECC: $H(R, M)B =$

Speed advantage:
is faster than $H(R$

Security advantag
attacker somehow
innocent M and d
with $H(M) = H(M$
Using $H(R)^{-1}H(M$
signs M then atta
same signature for
Using $H(R, M)$: n

will

q.

^{1}S

lars,

.

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A + \underline{S}R$.

Signer computes $\underline{S} = r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

From now on: Rename $\underline{S}$ as S.

Merge hashes: collision resil

$B^{H(R,M)} = AR^S$.
ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$
is faster than $H(R)^{-1}H(M)$

Security advantage: Imagine
attacker somehow finding
innocent M and dangerous
with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if sign
signs M then attacker reuse
same signature for M'.
Using $H(R, M)$: no problem

Precomputing quotient

Notation: $\underline{S} = H(R)^{-1}S$.

Send $(R, \underline{S})$ instead of (R, S) as signature: i.e., $\underline{S}$ computed by signer instead of verifier.

Verify $B^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
ECC: $(H(R)^{-1}H(M))B = A+\underline{S}R$.

Signer computes $\underline{S} =$
$r^{-1}(H(R)^{-1}H(M) - a) \bmod q$.

From now on: Rename $\underline{S}$ as S.

Merge hashes: collision resilience

$B^{H(R,M)} = AR^{S}$.
ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$ is faster than $H(R)^{-1}H(M)$.

Security advantage: Imagine attacker somehow finding innocent M and dangerous M' with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if signer signs M then attacker reuses same signature for M'.
Using $H(R, M)$: no problem.

puting quotient

n: $\underline{S} = H(R)^{-1}S$.

$, \underline{S})$ instead of (R, S) as
e: i.e., $\underline{S}$ computed by
stead of verifier.

$^{H(R)^{-1}H(M)} = AR^{\underline{S}}$.
$(R)^{-1}H(M))B = A + \underline{S}R$.

omputes $\underline{S} =$
$R)^{-1}H(M) - a)$ mod q.

w on: Rename $\underline{S}$ as S.

Merge hashes: collision resilience

$B^{H(R,M)} = AR^S$.
ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$
is faster than $H(R)^{-1}H(M)$.

Security advantage: Imagine
attacker somehow finding
innocent M and dangerous M'
with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if signer
signs M then attacker reuses
same signature for M'.
Using $H(R, M)$: no problem.

Eliminat

$B^S = R$
ECC: SB

Signer in
$S = r^{-1}$

Signer in
$S = r +$

Speed ad
Skip all

Security
slightly s
2000 Po

otient

$R)^{-1}S$.

nd of (R, S) as
computed by
erifier.

$M) = AR^{\underline{S}}$.
$M))B = A + \underline{S}R$.

$\underline{S} =$
$) - a) \bmod q$.

name $\underline{S}$ as S.

Merge hashes: collision resilience

$B^{H(R,M)} = AR^S$.
ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$ is faster than $H(R)^{-1}H(M)$.

Security advantage: Imagine attacker somehow finding innocent M and dangerous M' with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if signer signs M then attacker reuses same signature for M'.
Using $H(R, M)$: no problem.

Eliminate divisions

$B^S = RA^{H(R,M)}$.
ECC: $SB = R + H$

Signer in previous
$S = r^{-1}(H(R, M)$

Signer in this syst
$S = r + H(R, M)a$

Speed advantage:
Skip all inversions.

Security analysis is
slightly simpler. S
2000 Pointcheval–

S) as
by
.
$A + \underline{S}R$.
d q.
S.

Merge hashes: collision resilience

$B^{H(R,M)} = AR^S$.

ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$
is faster than $H(R)^{-1}H(M)$.

Security advantage: Imagine
attacker somehow finding
innocent M and dangerous M'
with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if signer
signs M then attacker reuses
same signature for M'.
Using $H(R, M)$: no problem.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:
$S = r^{-1}(H(R, M) - a) \bmod$

Signer in this system:
$S = r + H(R, M)a \bmod q$.

Speed advantage:
Skip all inversions.

Security analysis is similar,
slightly simpler. See, e.g.,
2000 Pointcheval–Stern.

Merge hashes: collision resilience

$B^{H(R,M)} = AR^S$.

ECC: $H(R, M)B = A + SR$.

Speed advantage: $H(R, M)$ is faster than $H(R)^{-1}H(M)$.

Security advantage: Imagine attacker somehow finding innocent M and dangerous M' with $H(M) = H(M')$.
Using $H(R)^{-1}H(M)$: if signer signs M then attacker reuses same signature for M'.
Using $H(R, M)$: no problem.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:
$S = r^{-1}(H(R, M) - a) \bmod q$.

Signer in this system:
$S = r + H(R, M)a \bmod q$.

Speed advantage:
Skip all inversions.

Security analysis is similar, slightly simpler. See, e.g., 2000 Pointcheval–Stern.

ashes: collision resilience

$) = AR^S$.

$(R, M)B = A + SR$.

dvantage: $H(R, M)$

than $H(R)^{-1}H(M)$.

advantage: Imagine

somehow finding

t M and dangerous M'

$M) = H(M')$.

$(R)^{-1}H(M)$: if signer

then attacker reuses

gnature for M'.

(R, M): no problem.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:

$S = r^{-1}(H(R, M) - a) \bmod q$.

Signer in this system:

$S = r + H(R, M)a \bmod q$.

Speed advantage:

Skip all inversions.

Security analysis is similar,

slightly simpler. See, e.g.,

2000 Pointcheval–Stern.

Signatur

Schnorr

$(H(R, M$

Given $(h$

recovers

checks h

ECC: R

Speed ad

when $H($

No secur

anyone c

lision resilience

$= A + SR$.

$H(R, M)$

$)^{-1}H(M)$.

e: Imagine

finding

angerous M'

$M')$.

$M)$: if signer

cker reuses

M'.

o problem.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:

$S = r^{-1}(H(R, M) - a) \bmod q$.

Signer in this system:

$S = r + H(R, M)a \bmod q$.

Speed advantage:

Skip all inversions.

Security analysis is similar, slightly simpler. See, e.g., 2000 Pointcheval–Stern.

Signature compres

Schnorr signature

$(H(R, M), S)$ inste

Given (h, S): verif

recovers $R = B^S/$

checks $h = H(R,$

ECC: $R = SB - h$

Speed advantage s

when $H(R, M)$ is s

No security impac

anyone can compr

ience

.

e

M'

er

s

.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:
$S = r^{-1}(H(R, M) - a) \bmod q$.

Signer in this system:
$S = r + H(R, M)a \bmod q$.

Speed advantage:
Skip all inversions.

Security analysis is similar,
slightly simpler. See, e.g.,
2000 Pointcheval–Stern.

Signature compression

Schnorr signature is
$(H(R, M), S)$ instead of $(R,$

Given (h, S): verifier
recovers $R = B^S/A^h$,
checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sig
when $H(R, M)$ is shorter tha

No security impact:
anyone can compress sigs.

Eliminate divisions

$B^S = RA^{H(R,M)}$.

ECC: $SB = R + H(R, M)A$.

Signer in previous system:
$S = r^{-1}(H(R, M) - a) \bmod q$.

Signer in this system:
$S = r + H(R, M)a \bmod q$.

Speed advantage:
Skip all inversions.

Security analysis is similar, slightly simpler. See, e.g., 2000 Pointcheval–Stern.

Signature compression

Schnorr signature is $(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier recovers $R = B^S/A^h$, checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs when $H(R, M)$ is shorter than R.

No security impact: anyone can compress sigs.

e divisions

$A^{H(R,M)}$.

$B = R + H(R,M)A$.

n previous system:

$(H(R,M) - a) \bmod q$.

n this system:

$H(R,M)a \bmod q$.

dvantage:

inversions.

analysis is similar,
simpler. See, e.g.,
intcheval–Stern.

Signature compression

Schnorr signature is
$(H(R,M), S)$ instead of (R,S).

Given (h,S): verifier
recovers $R = B^S/A^h$,
checks $h = H(R,M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs
when $H(R,M)$ is shorter than R.

No security impact:
anyone can compress sigs.

Half-size

Schnorr
e.g., 128

Advanta

s

$H(R, M)A$.

system:

$- a)$ mod q.

em:

a mod q.

.

s similar,

ee, e.g.,

-Stern.

Signature compression

Schnorr signature is $(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier recovers $R = B^S / A^h$, checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs when $H(R, M)$ is shorter than R.

No security impact: anyone can compress sigs.

Half-size H output

Schnorr chooses h

e.g., 128 bits inste

Advantage: smalle

q.

Signature compression

Schnorr signature is $(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier recovers $R = B^S/A^h$, checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs when $H(R, M)$ is shorter than R.

No security impact: anyone can compress sigs.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256

Advantage: smaller $(H(R, M$

Signature compression

Schnorr signature is
$(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier
recovers $R = B^S/A^h$,
checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs
when $H(R, M)$ is shorter than R.

No security impact:
anyone can compress sigs.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Signature compression

Schnorr signature is $(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier recovers $R = B^S/A^h$, checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs when $H(R, M)$ is shorter than R.

No security impact: anyone can compress sigs.

Half-size H output

Schnorr chooses half-size H: e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Signature compression

Schnorr signature is $(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier recovers $R = B^S/A^h$, checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs when $H(R, M)$ is shorter than R.

No security impact: anyone can compress sigs.

Half-size H output

Schnorr chooses half-size H: e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem: Recall that Schnorr's system is collision-resilient.

Signature compression

Schnorr signature is $(H(R, M), S)$ instead of (R, S).

Given (h, S): verifier recovers $R = B^S/A^h$, checks $h = H(R, M)$.

ECC: $R = SB - hA$.

Speed advantage sending sigs when $H(R, M)$ is shorter than R.

No security impact: anyone can compress sigs.

Half-size H output

Schnorr chooses half-size H: e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem: Recall that Schnorr's system is collision-resilient.

More serious objection: multi-target preimage attacks.

re compression

signature is
$M), S)$ instead of (R, S).

$h, S)$: verifier
$R = B^S/A^h$,
$h = H(R, M)$.

$= SB - hA$.

dvantage sending sigs
(R, M) is shorter than R.

rity impact:
can compress sigs.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem:
Recall that Schnorr's system is collision-resilient.

More serious objection:
multi-target preimage attacks.

DSA and

DSA is E
- prime-
- A^{-1} in
- two sc

Much w
- does n
- does n
- is not
- require
- require
(or thr

ssion

is
ead of (R, S).

fier
A^h,
$M)$.

A.

sending sigs
shorter than R.

t:
ess sigs.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash
functions allow collisions!"

Not an obvious problem:
Recall that Schnorr's system
is collision-resilient.

More serious objection:
multi-target preimage attacks.

DSA and ECDSA

DSA is ElGamal p

- prime-order subg
- A^{-1} instead of
- two scalars.

Much worse than

- does not hash R
- does not merge
- is not collision-re
- requires inversio
- requires inversio
  (or three expone

S).

gs

an R.

Half-size H output

Schnorr chooses half-size H:
e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem:
Recall that Schnorr's system is collision-resilient.

More serious objection:
multi-target preimage attacks.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: D

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signe
- requires inversion for verifi
  (or three exponents).

Half-size H output

Schnorr chooses half-size H: e.g., 128 bits instead of 256 bits.

Advantage: smaller $(H(R, M), S)$.

Objection: "128-bit hash functions allow collisions!"

Not an obvious problem: Recall that Schnorr's system is collision-resilient.

More serious objection: multi-target preimage attacks.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

e H output

chooses half-size H:

3 bits instead of 256 bits.

ge: smaller $(H(R, M), S)$.

on: "128-bit hash

s allow collisions!"

obvious problem:

hat Schnorr's system

on-resilient.

rious objection:

rget preimage attacks.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

EdDSA

EdDSA

- compl
- no sig
- double
- A as e
- determ

t

alf-size H:
ead of 256 bits.

er $(H(R, M), S)$.

it hash
llisions!"

oblem:
rr's system
t.

ction:
age attacks.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

EdDSA

EdDSA is Schnorr

- complete twisted
- no signature co
- double-size H ou
- A as extra H inp
- deterministic R.

bits.

$M), S)$.

ks.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

DSA and ECDSA

DSA is ElGamal plus

- prime-order subgroups;
- A^{-1} instead of A;
- two scalars.

Much worse than Schnorr: DSA

- does not hash R;
- does not merge hashes;
- is not collision-resilient;
- requires inversion for signer;
- requires inversion for verifier (or three exponents).

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

Extra H input: $H(R, A, M)$.
Speed impact: negligible.
Alleviates concerns that several public keys could be attacked simultaneously.

d ECDSA

ElGamal plus
order subgroups;
nstead of A;
alars.

orse than Schnorr: DSA
not hash R;
not merge hashes;
collision-resilient;
es inversion for signer;
es inversion for verifier
ree exponents).

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

Extra H input: $H(R, A, M)$.
Speed impact: negligible.
Alleviates concerns that several public keys could be attacked simultaneously.

Why no

1. ECC
even wit
64 bytes
using hig

2. Secur
needs th

3. Doub
concerns

4. Avoid
allows a
batch sig

lus
groups;
A;

Schnorr: DSA
R;
hashes;
esilient;
n for signer;
n for verifier
ents).

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

Extra H input: $H(R, A, M)$.
Speed impact: negligible.
Alleviates concerns that several public keys could be attacked simultaneously.

Why no signature

1. ECC signatures
even without comp
64 bytes for signat
using high-security

2. Security of sho
needs thorough an

3. Double-size H
concerns regarding

4. Avoiding compr
allows another spe
batch signature ve

DSA

er;

ier

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

Extra H input: $H(R, A, M)$.
Speed impact: negligible.
Alleviates concerns that several public keys could be attacked simultaneously.

Why no signature compressi

1. ECC signatures are short even without compression. 64 bytes for signature using high-security curve.

2. Security of shorter H needs thorough analysis.

3. Double-size H alleviates concerns regarding H securit

4. Avoiding compression allows another speedup: batch signature verification.

EdDSA

EdDSA is Schnorr with

- complete twisted Edwards curve;
- no signature compression;
- double-size H output;
- A as extra H input;
- deterministic R.

Extra H input: $H(R, A, M)$.
Speed impact: negligible.
Alleviates concerns that several public keys could be attacked simultaneously.

Why no signature compression:

1. ECC signatures are short even without compression. 64 bytes for signature using high-security curve.

2. Security of shorter H needs thorough analysis.

3. Double-size H alleviates concerns regarding H security.

4. Avoiding compression allows another speedup: batch signature verification.