# Error-prone cryptographic designs

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

---

"*The poor user is*
*given enough rope with which*
*to hang himself—something*
*a standard should not do.*"
—1992 Rivest,
commenting on nonce generation
inside Digital Signature Algorithm
(1991 proposal by NIST,
1992 credited to NSA,
1994 standardized by NIST)

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

# Error-prone cryptographic designs

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

---

"*The poor user is*
*given enough rope with which*
*to hang himself—something*
*a standard should not do.*"
—1992 Rivest,
commenting on nonce generation
inside Digital Signature Algorithm
(1991 proposal by NIST,
1992 credited to NSA,
1994 standardized by NIST)

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

# Error-prone cryptographic designs

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

---

"*The poor user is*
*given enough rope with which*
*to hang himself—something*
*a standard should not do.*"
—1992 Rivest,
commenting on nonce generation
inside Digital Signature Algorithm
(1991 proposal by NIST,
1992 credited to NSA,
1994 standardized by NIST)

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

Rivest's response: Blame DSA.
**Blame the crypto designer.**

# Error-prone cryptographic designs

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

---

"*The poor user is given enough rope with which to hang himself—something a standard should not do.*"
—1992 Rivest,
commenting on nonce generation
inside Digital Signature Algorithm
(1991 proposal by NIST,
1992 credited to NSA,
1994 standardized by NIST)

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

Rivest's response: Blame DSA.
**Blame the crypto designer.**
Change DSA to avoid this pitfall!

one cryptographic designs

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

---

*oor user is*
*ough rope with which*
*himself—something*
*ard should not do.*"
Rivest,
ting on nonce generation
igital Signature Algorithm
roposal by NIST,
edited to NSA,
ndardized by NIST)

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

Rivest's response: Blame DSA.
**Blame the crypto designer.**
Change DSA to avoid this pitfall!

## Crypto h

2005 Os
65ms to
used for
Attack p
but with

Almost a
use fast
Kernel's
influence
influenci
influenci
of the at
65ms to

ographic designs

n
is at Chicago &
siteit Eindhoven

---

*e with which*
*something*
*not do.*"

once generation
ature Algorithm
NIST,
NSA,
by NIST)

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

Rivest's response: Blame DSA.
**Blame the crypto designer.**
Change DSA to avoid this pitfall!

## Crypto horror stor

2005 Osvik–Shami
65ms to steal Linu
used for hard-disk
Attack process on
but without privile

Almost all AES im
use fast lookup ta
Kernel's secret AE
influences table-lo
influencing CPU c
influencing measur
of the attack proc
65ms to compute

esigns

ago &
hoven

*ch*

ration
orithm

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

Rivest's response: Blame DSA.
**Blame the crypto designer.**
Change DSA to avoid this pitfall!

## Crypto horror story #2

2005 Osvik–Shamir–Tromer:
65ms to steal Linux AES ke
used for hard-disk encryptio
Attack process on same CP
but without privileges.

Almost all AES implementat
use fast lookup tables.
Kernel's secret AES key
influences table-load address
influencing CPU cache state
influencing measurable timi
of the attack process.
65ms to compute influence

## Crypto horror story #1

2010 Bushing–Marcan–Segher–Sven "failOverflow" demolition of Sony PS3 security system: Sony had ignored requirement to generate new random nonce for each ECDSA signature.
⇒ Sony's signatures leaked Sony's secret code-signing key.

Traditional response: Blame Sony.
**Blame the crypto implementor.**

Rivest's response: Blame DSA.
**Blame the crypto designer.**
Change DSA to avoid this pitfall!

## Crypto horror story #2

2005 Osvik–Shamir–Tromer: 65ms to steal Linux AES key used for hard-disk encryption. Attack process on same CPU but without privileges.

Almost all AES implementations use fast lookup tables. Kernel's secret AES key influences table-load addresses, influencing CPU cache state, influencing measurable timings of the attack process. 65ms to compute influence$^{-1}$.

## horror story #1

ushing–Marcan–Segher–
ailOverflow" demolition
PS3 security system:
d ignored requirement
ate new random nonce
ECDSA signature.
's signatures leaked
ecret code-signing key.

nal response: Blame Sony.
**the crypto implementor.**

response: Blame DSA.
**the crypto designer.**
DSA to avoid this pitfall!

## Crypto horror story #2

2005 Osvik–Shamir–Tromer:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms to compute influence$^{-1}$.

2012 Mo
Shachan
data-cac
x86 proc
somehow
physical
memory
program

y #1

rcan–Segher–
v" demolition
rity system:
requirement
andom nonce
ignature.
res leaked
e-signing key.

se: Blame Sony.
**o implementor.**

Blame DSA.
**o designer.**
void this pitfall!

Crypto horror story #2

2005 Osvik–Shamir–Tromer:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms to compute influence$^{-1}$.

2012 Mowery–Kee
Shacham: "We po
data-cache timing
x86 processors tha
somehow subvert
physical indexing,
memory requireme
programs is doome

er–
ion
n:
nt
nce

ey.

e Sony.
**entor.**

SA.
**r.**
itfall!

Crypto horror story #2

2005 Osvik–Shamir–Tromer:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms to compute influence$^{-1}$.

2012 Mowery–Keelveedhi–
Shacham: "We posit that a
data-cache timing attack ag
x86 processors that does not
somehow subvert the prefet
physical indexing, and massi
memory requirements of mo
programs is doomed to fail."

## Crypto horror story #2

2005 Osvik–Shamir–Tromer:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms to compute influence$^{-1}$.

2012 Mowery–Keelveedhi–
Shacham: "We posit that any
data-cache timing attack against
x86 processors that does not
somehow subvert the prefetcher,
physical indexing, and massive
memory requirements of modern
programs is doomed to fail."

## Crypto horror story #2

2005 Osvik–Shamir–Tromer:
65ms to steal Linux AES key
used for hard-disk encryption.
Attack process on same CPU
but without privileges.

Almost all AES implementations
use fast lookup tables.
Kernel's secret AES key
influences table-load addresses,
influencing CPU cache state,
influencing measurable timings
of the attack process.
65ms to compute influence$^{-1}$.

2012 Mowery–Keelveedhi–
Shacham: "We posit that any
data-cache timing attack against
x86 processors that does not
somehow subvert the prefetcher,
physical indexing, and massive
memory requirements of modern
programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–
Sunar "Wait a minute! A fast,
Cross-VM attack on AES"
recovers "the AES keys
of OpenSSL 1.0.1 running inside
the victim VM" in 60 seconds
despite VMware virtualization.

horror story #2

vik–Shamir–Tromer:
steal Linux AES key
hard-disk encryption.
process on same CPU
hout privileges.

all AES implementations
lookup tables.
secret AES key
es table-load addresses,
ng CPU cache state,
ng measurable timings
ttack process.
compute influence$^{-1}$.

2012 Mowery–Keelveedhi–Shacham: "We posit that any data-cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, Cross-VM attack on AES" recovers "the AES keys of OpenSSL 1.0.1 running inside the victim VM" in 60 seconds despite VMware virtualization.

After ma
on imple
today we
plagued
Warning

y #2

ir–Tromer:
ux AES key
encryption.
same CPU
eges.

nplementations
bles.
ES key
ad addresses,
ache state,
rable timings
ess.
influence$^{-1}$.

2012 Mowery–Keelveedhi–Shacham: "We posit that any data-cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, Cross-VM attack on AES" recovers "the AES keys of OpenSSL 1.0.1 running inside the victim VM" in 60 seconds despite VMware virtualization.

After many, many,
on implementation
today we still have
plagued with AES
Warning: more pa

2012 Mowery–Keelveedhi–Shacham: "We posit that any data-cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, Cross-VM attack on AES" recovers "the AES keys of OpenSSL 1.0.1 running inside the victim VM" in 60 seconds despite VMware virtualization.

After many, many, many pap
on implementations and atta
today we still have an ecosy
plagued with AES vulnerabil
Warning: more papers $\neq$ se

2012 Mowery–Keelveedhi–Shacham: "We posit that any data-cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, Cross-VM attack on AES" recovers "the AES keys of OpenSSL 1.0.1 running inside the victim VM" in 60 seconds despite VMware virtualization.

After many, many, many papers on implementations and attacks, today we still have an ecosystem plagued with AES vulnerabilities. Warning: more papers $\neq$ security.

2012 Mowery–Keelveedhi–Shacham: "We posit that any data-cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, Cross-VM attack on AES" recovers "the AES keys of OpenSSL 1.0.1 running inside the victim VM" in 60 seconds despite VMware virtualization.

After many, many, many papers on implementations and attacks, today we still have an ecosystem plagued with AES vulnerabilities. Warning: more papers $\neq$ security.

AES has a serious conflict between security, simplicity, speed. It's tough to achieve security while insisting on the AES design —i.e., blaming the implementor.

2012 Mowery–Keelveedhi–Shacham: "We posit that any data-cache timing attack against x86 processors that does not somehow subvert the prefetcher, physical indexing, and massive memory requirements of modern programs is doomed to fail."

2014 Irazoqui–Inci–Eisenbarth–Sunar "Wait a minute! A fast, Cross-VM attack on AES" recovers "the AES keys of OpenSSL 1.0.1 running inside the victim VM" in 60 seconds despite VMware virtualization.

After many, many, many papers on implementations and attacks, today we still have an ecosystem plagued with AES vulnerabilities. Warning: more papers $\neq$ security.

AES has a serious conflict between security, simplicity, speed. It's tough to achieve security while insisting on the AES design —i.e., blaming the implementor.

Allowing the *design* to vary makes security much easier. Next-generation ciphers are naturally constant-time and fast.

owery–Keelveedhi–
n: “We posit that any
che timing attack against
cessors that does not
w subvert the prefetcher,
indexing, and massive
requirements of modern
s is doomed to fail.”

zoqui–Inci–Eisenbarth–
Wait a minute! A fast,
M attack on AES”
“the AES keys
SSL 1.0.1 running inside
m VM” in 60 seconds
VMware virtualization.

After many, many, many papers
on implementations and attacks,
today we still have an ecosystem
plagued with AES vulnerabilities.
Warning: more papers $\neq$ security.

AES has a serious conflict
between security, simplicity, speed.
It's tough to achieve security
while insisting on the AES design
—i.e., blaming the implementor.

Allowing the *design* to vary
makes security much easier.
Next-generation ciphers are
naturally constant-time and fast.

The big

Cry
des

Cry
impleme

elveedhi–
osit that any
attack against
t does not
the prefetcher,
and massive
ents of modern
ed to fail."

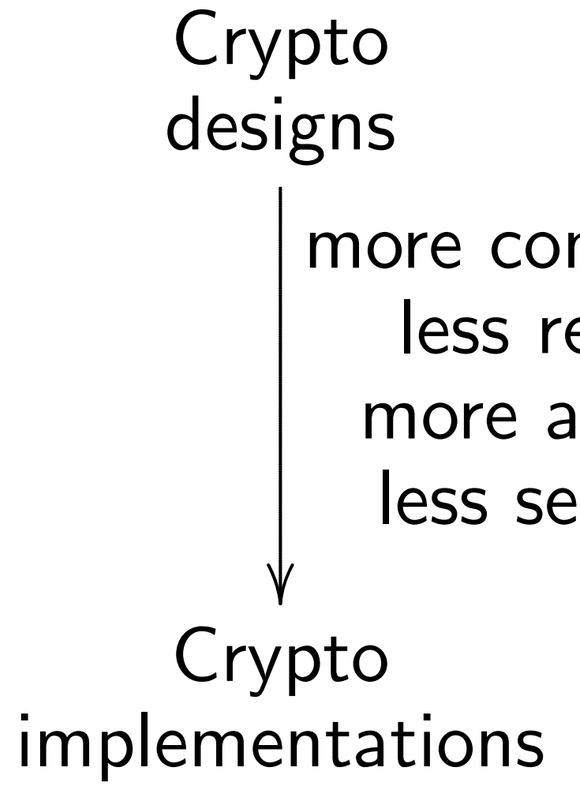
–Eisenbarth–
nute! A fast,
on AES"
keys
running inside
60 seconds
irtualization.

After many, many, many papers on implementations and attacks, today we still have an ecosystem plagued with AES vulnerabilities.
Warning: more papers $\neq$ security.

AES has a serious conflict between security, simplicity, speed.
It's tough to achieve security while insisting on the AES design —i.e., blaming the implementor.

Allowing the *design* to vary makes security much easier.
Next-generation ciphers are naturally constant-time and fast.

<u>The big picture</u>



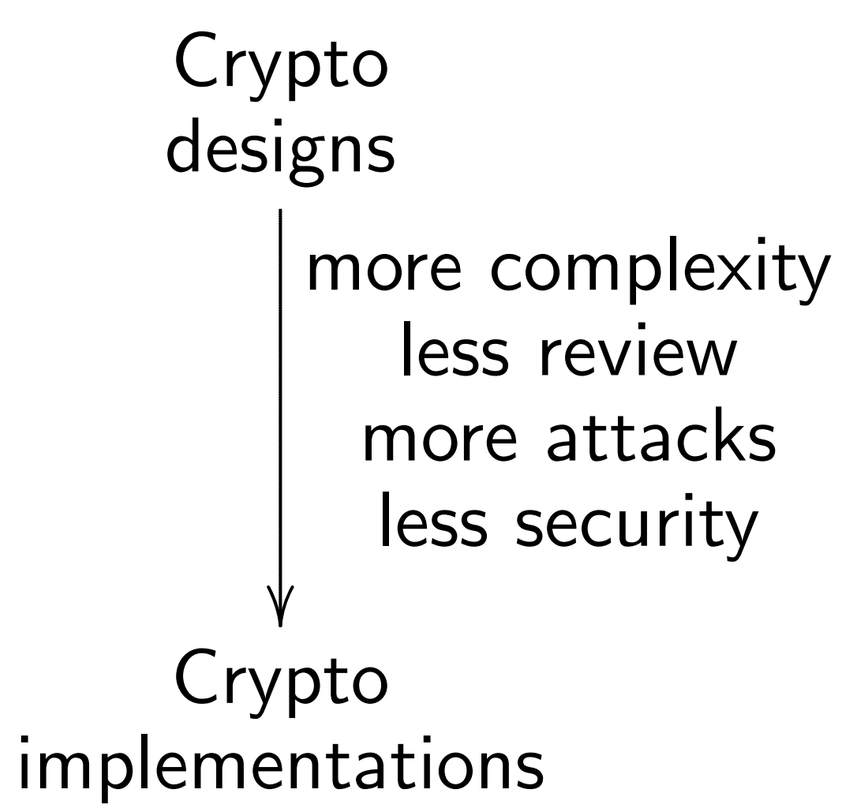
ny
ainst
t
cher,
ve
dern
,
th–
st,
nside
ds
on.

After many, many, many papers on implementations and attacks, today we still have an ecosystem plagued with AES vulnerabilities. Warning: more papers $\neq$ security.

AES has a serious conflict between security, simplicity, speed. It's tough to achieve security while insisting on the AES design —i.e., blaming the implementor.

Allowing the *design* to vary makes security much easier. Next-generation ciphers are naturally constant-time and fast.

## The big picture

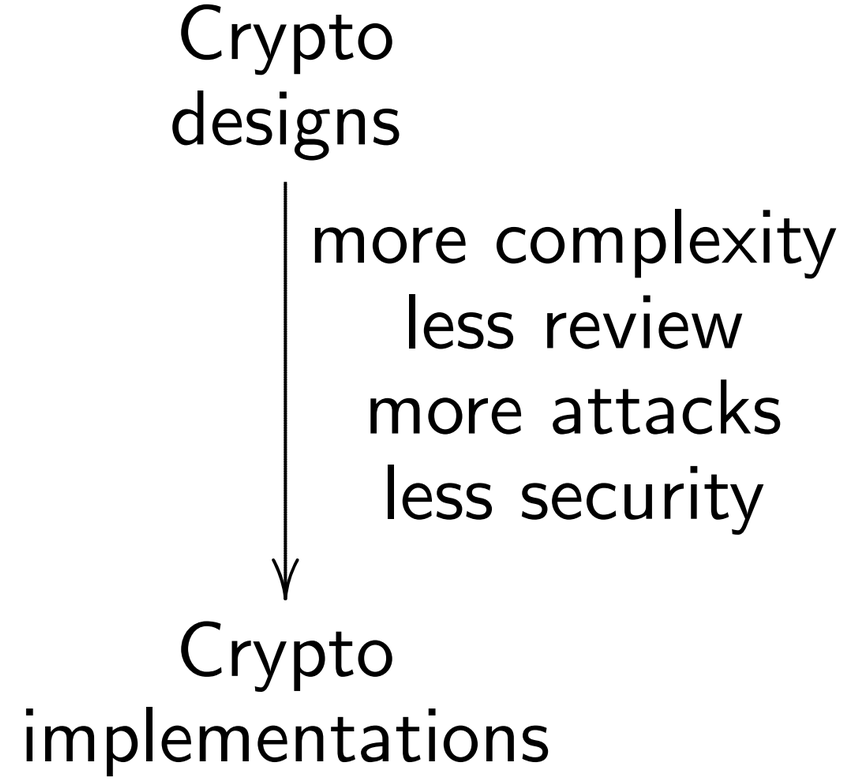


After many, many, many papers
on implementations and attacks,
today we still have an ecosystem
plagued with AES vulnerabilities.
Warning: more papers $\neq$ security.

AES has a serious conflict
between security, simplicity, speed.
It's tough to achieve security
while insisting on the AES design
—i.e., blaming the implementor.

Allowing the *design* to vary
makes security much easier.
Next-generation ciphers are
naturally constant-time and fast.

## The big picture

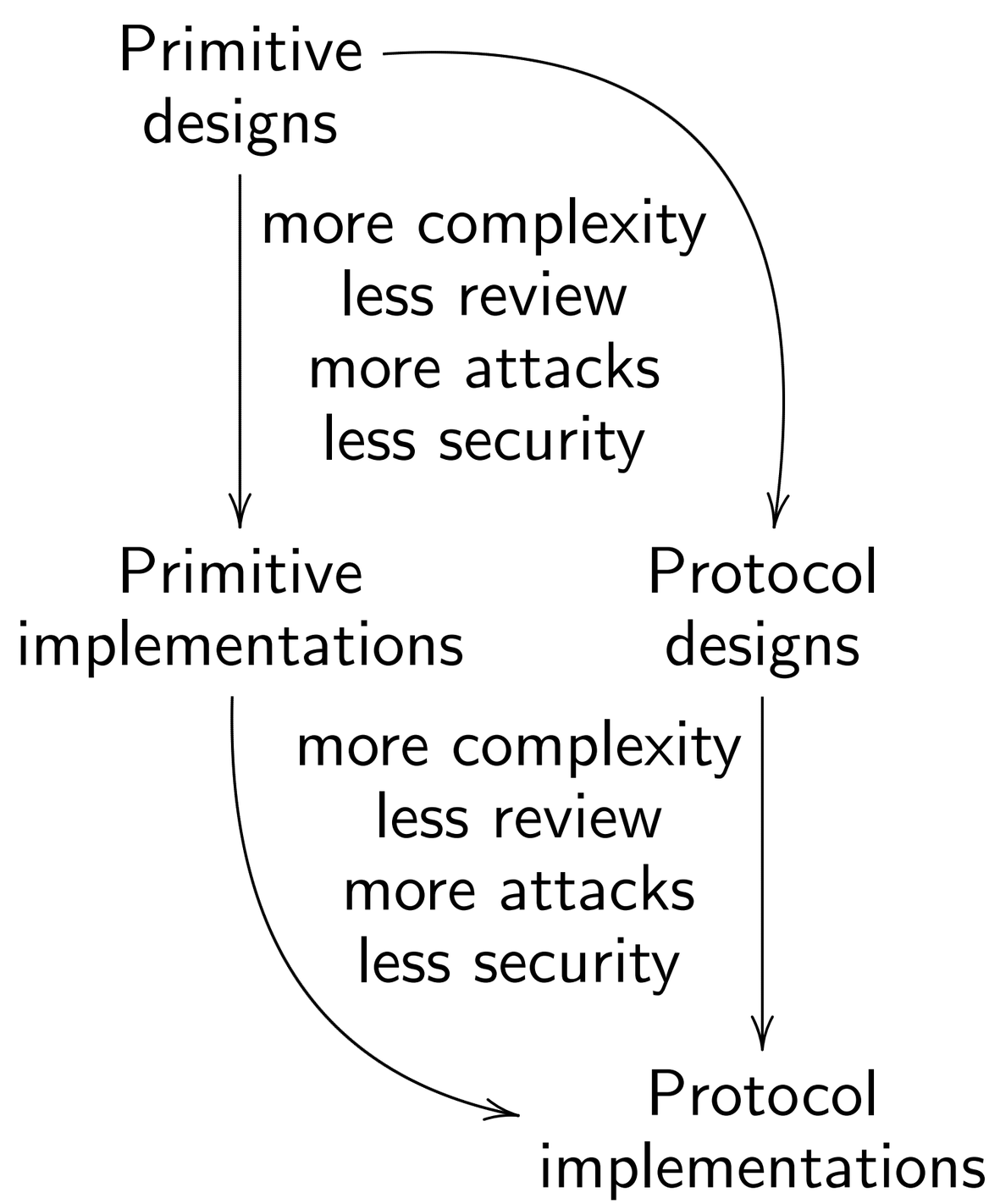


After many, many, many papers
on implementations and attacks,
today we still have an ecosystem
plagued with AES vulnerabilities.
Warning: more papers $\neq$ security.

AES has a serious conflict
between security, simplicity, speed.
It's tough to achieve security
while insisting on the AES design
—i.e., blaming the implementor.

Allowing the *design* to vary
makes security much easier.
Next-generation ciphers are
naturally constant-time and fast.

## The big picture

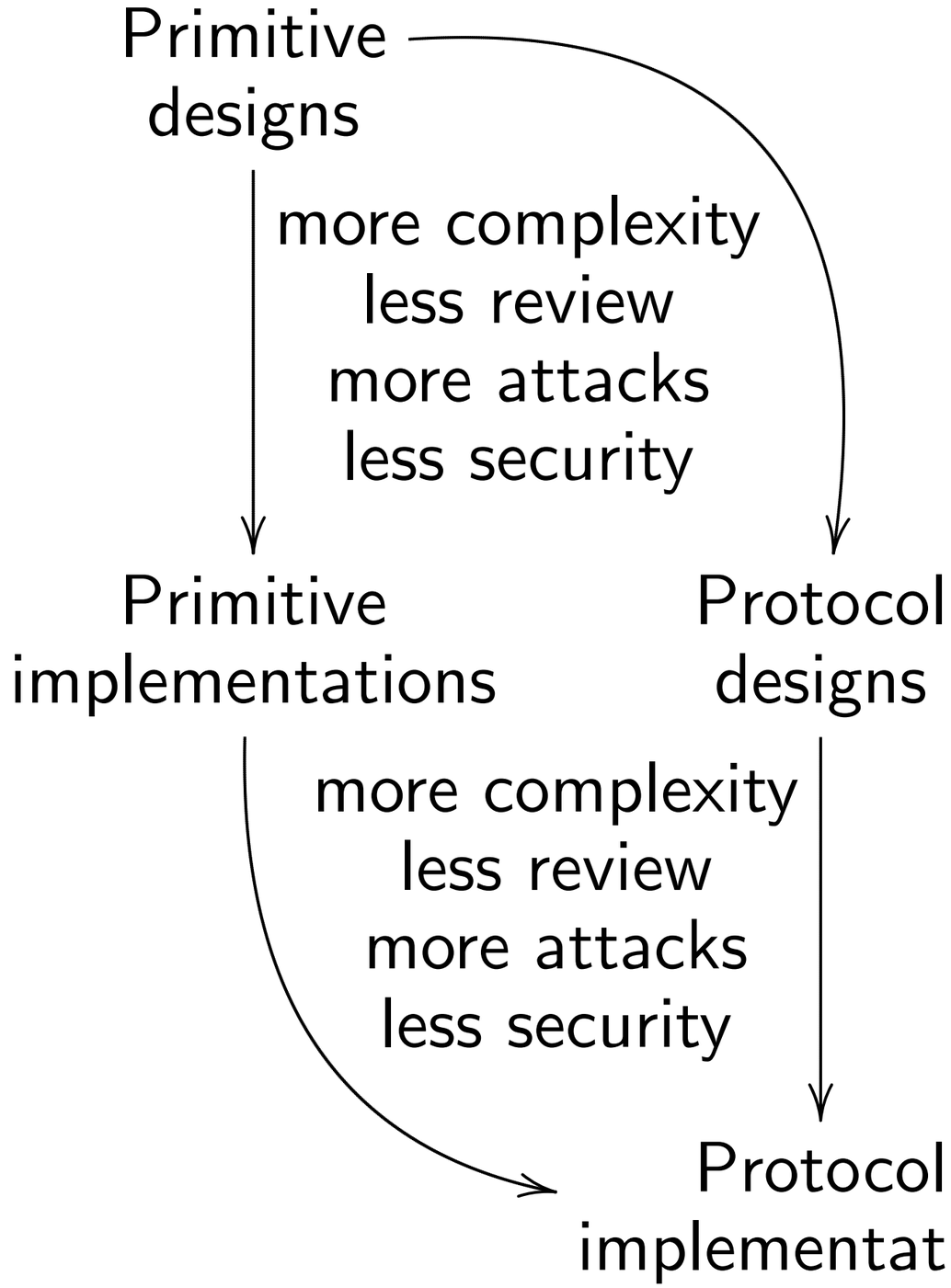


any, many, many papers
ementations and attacks,
e still have an ecosystem
with AES vulnerabilities.
: more papers $\neq$ security.

s a serious conflict
security, simplicity, speed.
gh to achieve security
sisting on the AES design
laming the implementor.

g the *design* to vary
ecurity much easier.
neration ciphers are
y constant-time and fast.

## The big picture




Public re
towards
ignoring
protocol

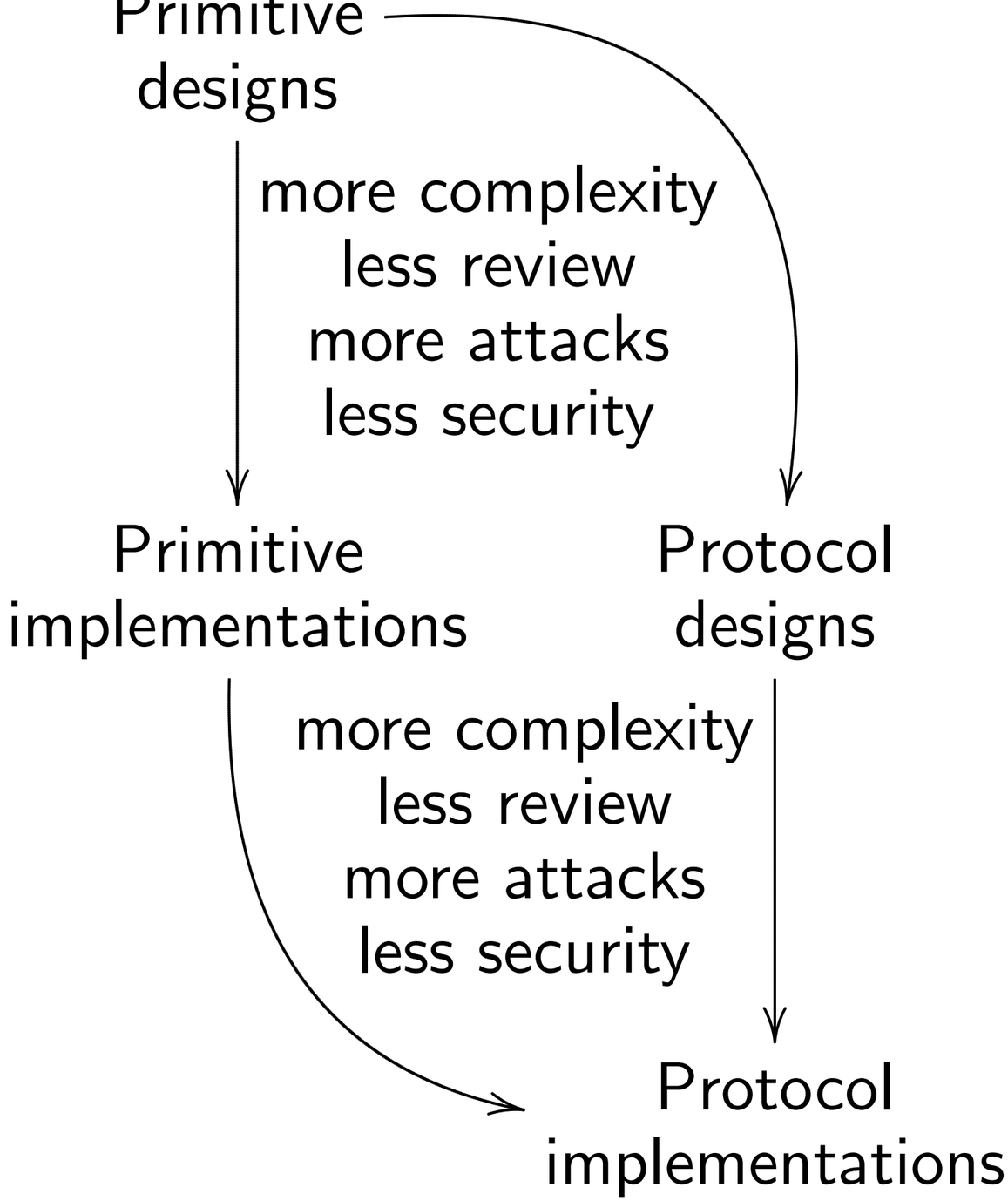
There's
of, e.g.,
than of

many papers
ns and attacks,
e an ecosystem
vulnerabilities.
pers ≠ security.

conflict
simplicity, speed.
eve security
the AES design
e implementor.

*n* to vary
uch easier.
iphers are
-time and fast.

## The big picture




Public review is na
towards the simple
ignoring complicat
protocols and imp

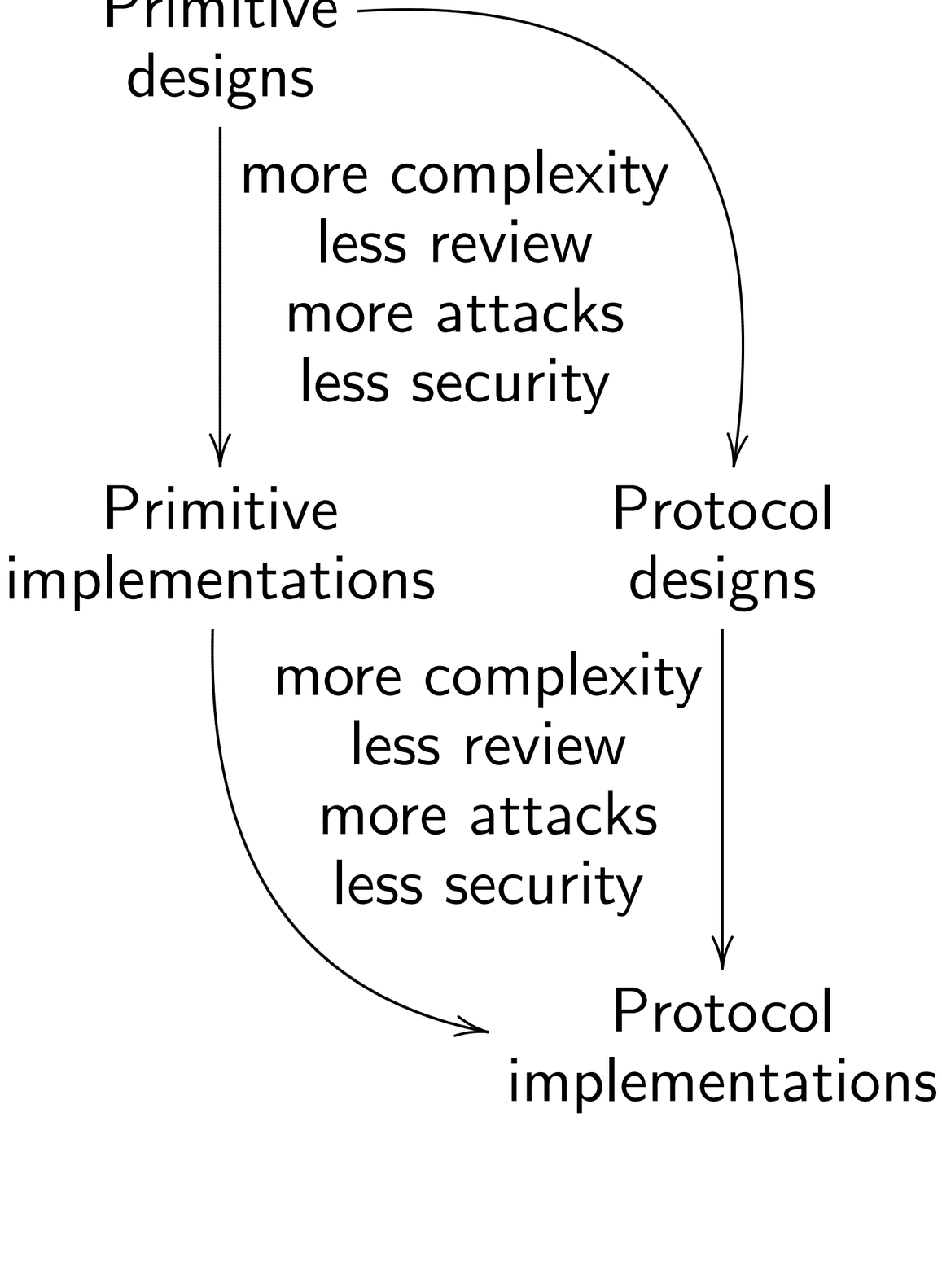
There's much mor
of, e.g., discrete lo
than of ECDSA sig

pers
acks,
stem
lities.
curity.

speed.
y
lesign
ntor.

fast.

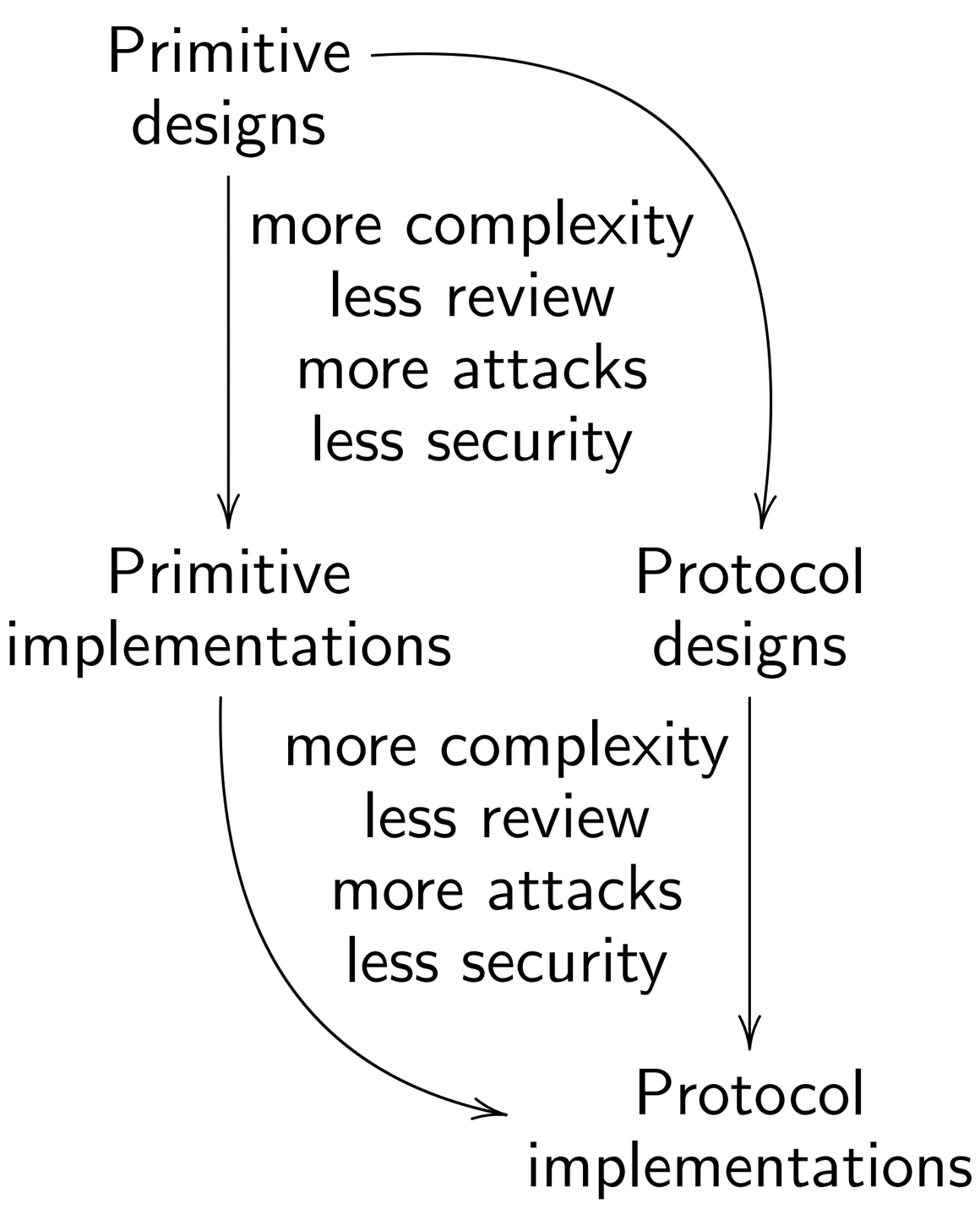
## The big picture




Public review is naturally bia
towards the simplest targets
ignoring complications of
protocols and implementatio

There's much more public re
of, e.g., discrete logarithms
than of ECDSA signatures.

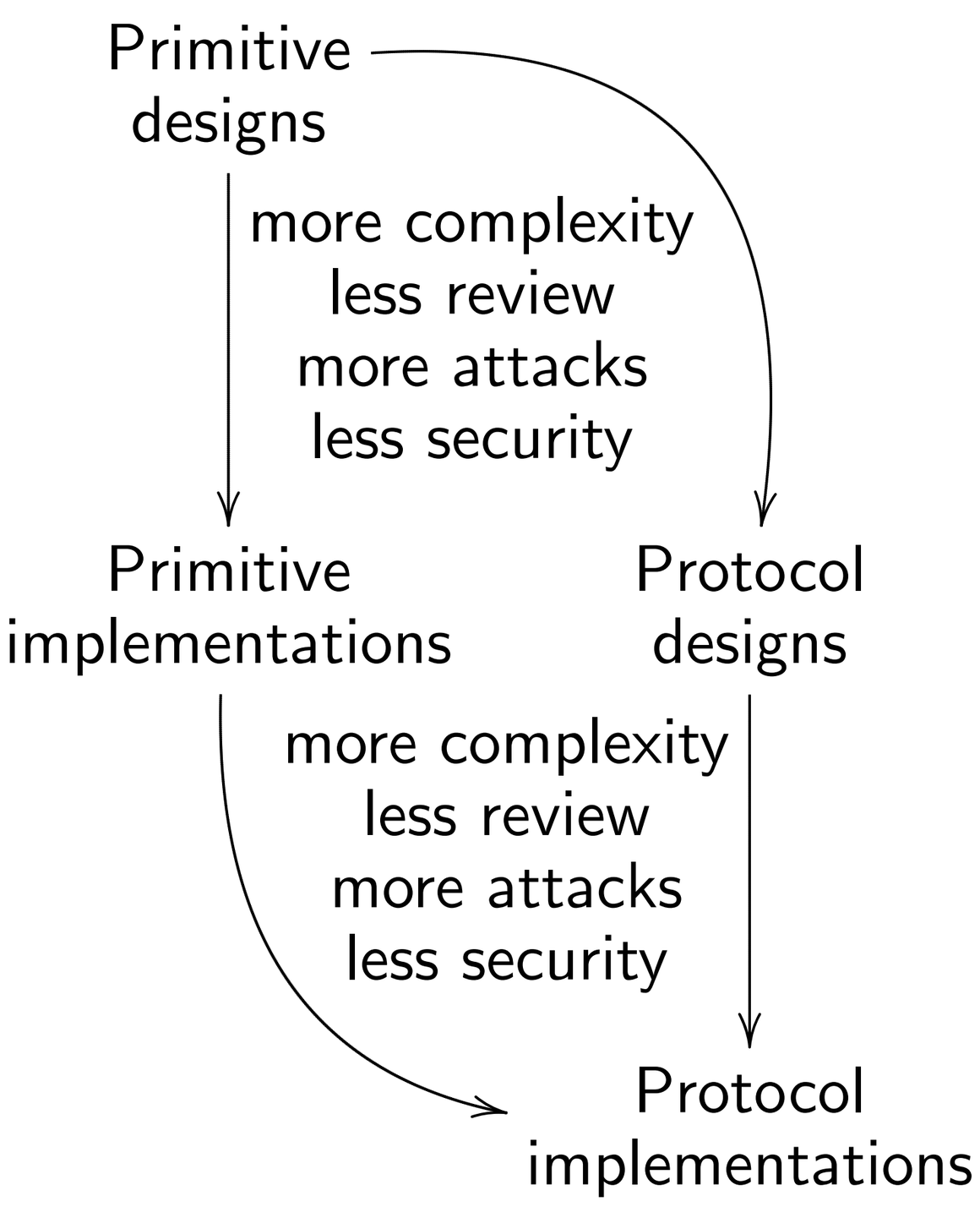
## The big picture




Public review is naturally biased towards the simplest targets, ignoring complications of protocols and implementations.

There's much more public review of, e.g., discrete logarithms than of ECDSA signatures.

## The big picture

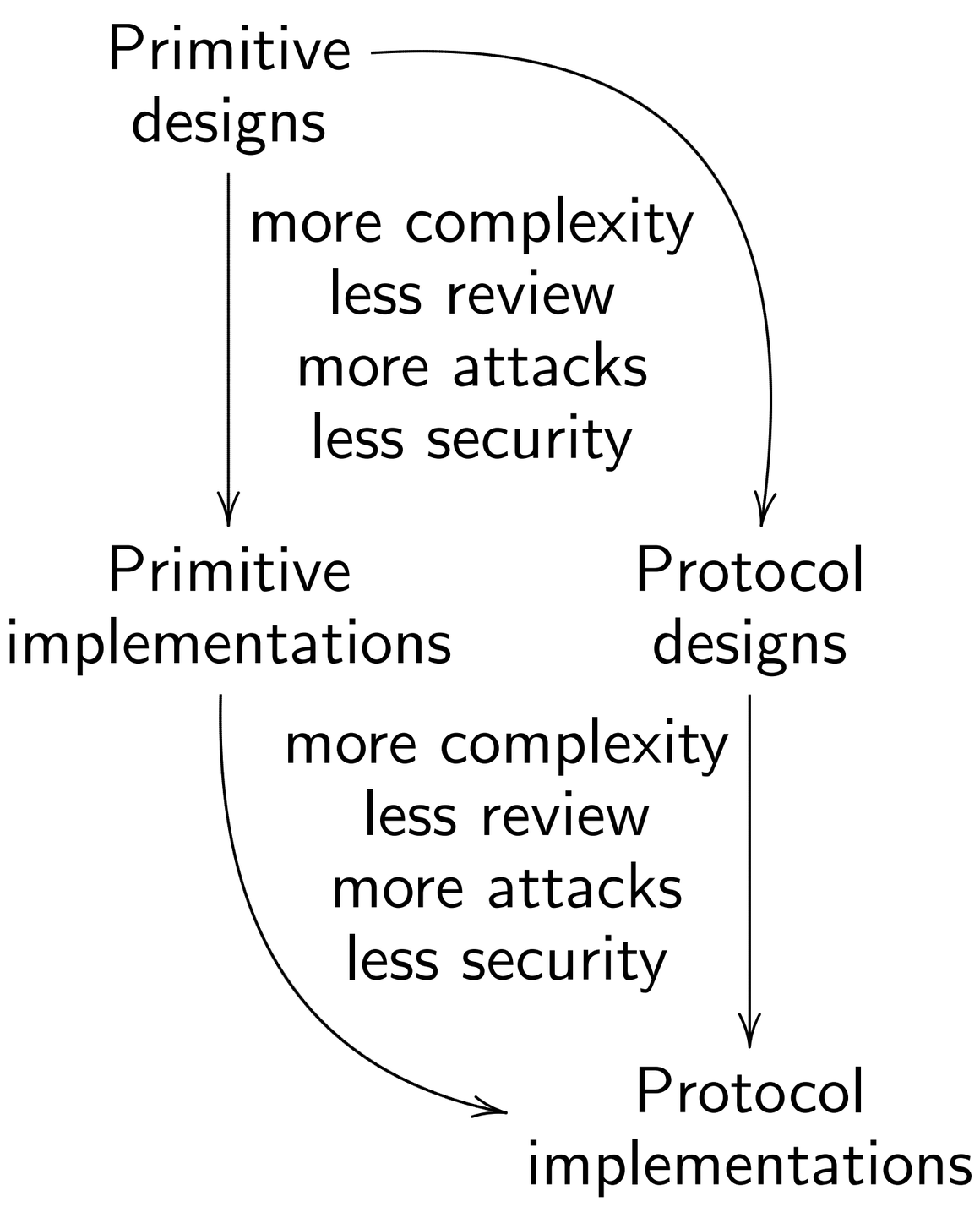



Public review is naturally biased towards the simplest targets, ignoring complications of protocols and implementations.

There's much more public review of, e.g., discrete logarithms than of ECDSA signatures.

There's much more public review of the ECDSA design than of ECDSA implementations.

## The big picture



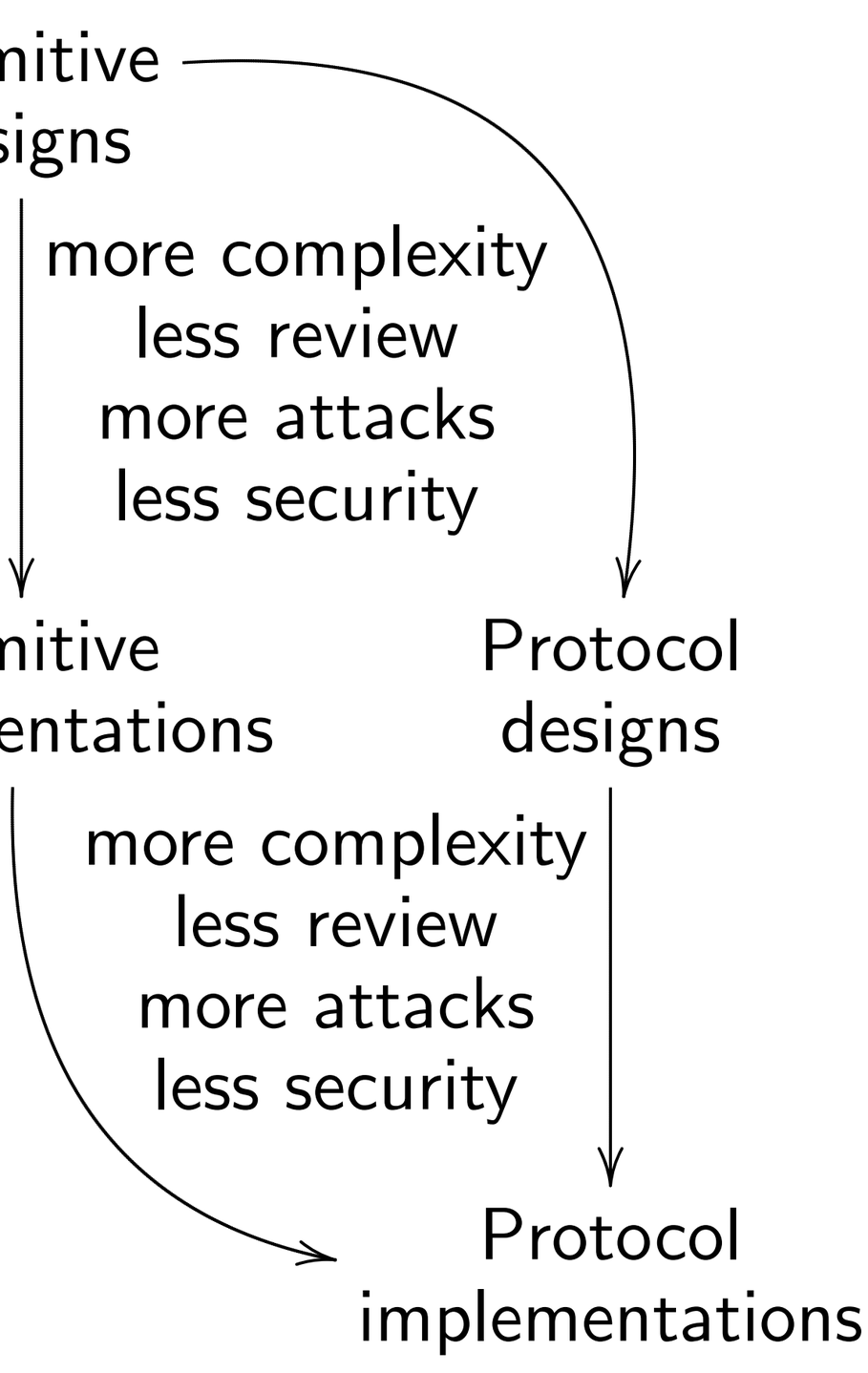

Public review is naturally biased towards the simplest targets, ignoring complications of protocols and implementations.

There's much more public review of, e.g., discrete logarithms than of ECDSA signatures.

There's much more public review of the ECDSA design than of ECDSA implementations.

There's much more public review of ECDSA implementations than of ECDSA applications.

## picture




Public review is naturally biased towards the simplest targets, ignoring complications of protocols and implementations.

There's much more public review of, e.g., discrete logarithms than of ECDSA signatures.

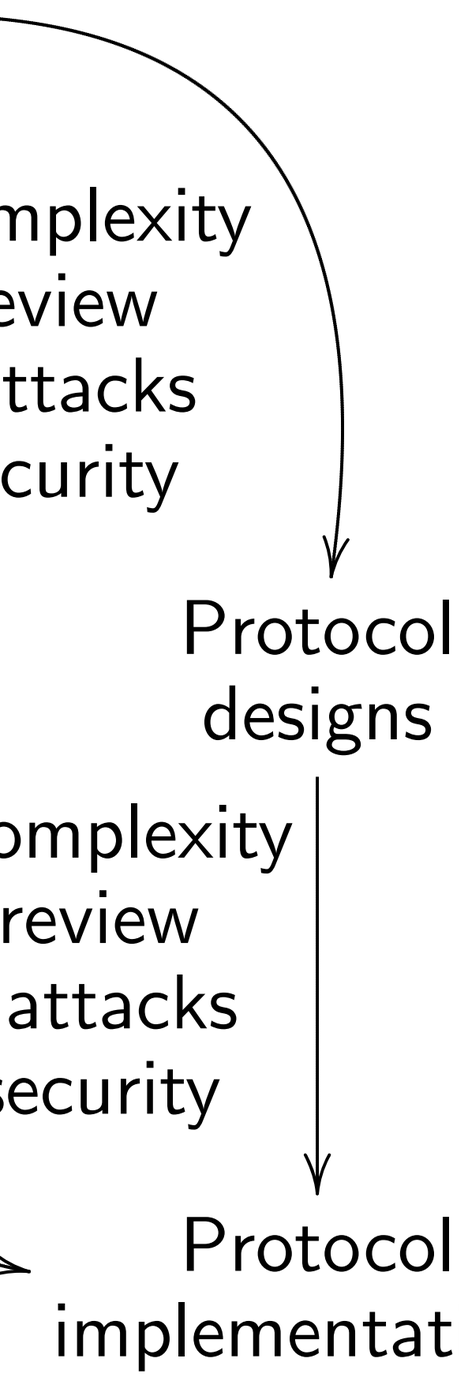
There's much more public review of the ECDSA design than of ECDSA implementations.

There's much more public review of ECDSA implementations than of ECDSA applications.

## What ab

The fun
of "prov
Prove th
is as sec

i.e.: Pro
is as sec
Prove th
is as sec
Then it'
to focus




Public review is naturally biased towards the simplest targets, ignoring complications of protocols and implementations.

There's much more public review of, e.g., discrete logarithms than of ECDSA signatures.

There's much more public review of the ECDSA design than of ECDSA implementations.

There's much more public review of ECDSA implementations than of ECDSA applications.

What about securi

The fundamental g
of "provable secur
Prove that the wh
is as secure as the

i.e.: Prove that th
is as secure as the
Prove that the imp
is as secure as the
Then it's safe for
to focus on the pr

ocol
gns

ocol
ntations

Public review is naturally biased
towards the simplest targets,
ignoring complications of
protocols and implementations.

There's much more public review
of, e.g., discrete logarithms
than of ECDSA signatures.

There's much more public review
of the ECDSA design
than of ECDSA implementations.

There's much more public review
of ECDSA implementations
than of ECDSA applications.

What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementati
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive des

Public review is naturally biased
towards the simplest targets,
ignoring complications of
protocols and implementations.

There's much more public review
of, e.g., discrete logarithms
than of ECDSA signatures.

There's much more public review
of the ECDSA design
than of ECDSA implementations.

There's much more public review
of ECDSA implementations
than of ECDSA applications.

## What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Public review is naturally biased
towards the simplest targets,
ignoring complications of
protocols and implementations.

There's much more public review
of, e.g., discrete logarithms
than of ECDSA signatures.

There's much more public review
of the ECDSA design
than of ECDSA implementations.

There's much more public review
of ECDSA implementations
than of ECDSA applications.

What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

eview is naturally biased
the simplest targets,
complications of
s and implementations.

much more public review
discrete logarithms
ECDSA signatures.

much more public review
CDSA design
ECDSA implementations.

much more public review
SA implementations
ECDSA applications.

What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem
Proofs a
rarely re

aturally biased
est targets,
tions of
lementations.

re public review
ogarithms
gnatures.

re public review
sign
nplementations.

re public review
nentations
pplications.

What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem 1: "Proo
Proofs are increasi
rarely reviewed, ra

ased

,

ons.

eview

eview

tions.

eview

.

What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem 1: "Proofs" have e
Proofs are increasingly comp
rarely reviewed, rarely autom

## What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

## What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

## What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

## What about security proofs?

The fundamental goal
of "provable security":
Prove that the whole system
is as secure as the primitive.

i.e.: Prove that the protocol
is as secure as the primitive.
Prove that the implementation
is as secure as the design.
Then it's safe for reviewers
to focus on the primitive design.

Maybe will succeed someday, but
needs to overcome huge problems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

Problem 4: Maybe the only way
to achieve the fundamental goal
is to switch to weak primitives.

bout security proofs?

damental goal
able security":
hat the whole system
ure as the primitive.

ve that the protocol
ure as the primitive.
hat the implementation
ure as the design.
s safe for reviewers
on the primitive design.

will succeed someday, but
overcome huge problems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

Problem 4: Maybe the only way
to achieve the fundamental goal
is to switch to weak primitives.

Some ac

Creating
**Think a**
**implem**
What wi
What er
to appea
Can you

ity proofs?

goal
ity":
ole system
primitive.

e protocol
primitive.
plementation
design.
reviewers
imitive design.

d someday, but
e huge problems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

Problem 4: Maybe the only way
to achieve the fundamental goal
is to switch to weak primitives.

Some advice to cr

Creating or evalua
**Think about the**
**implementations**
What will the imp
What errors are lik
to appear in imple
Can you compensa

?

n

l

ion

sign.

y, but

blems.

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

Problem 4: Maybe the only way
to achieve the fundamental goal
is to switch to weak primitives.

Some advice to crypto desig

Creating or evaluating a des
**Think about the**
**implementations.**
What will the implementors
What errors are likely
to appear in implementation
Can you compensate for this

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

Problem 4: Maybe the only way
to achieve the fundamental goal
is to switch to weak primitives.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the**
**implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Problem 1: "Proofs" have errors.
Proofs are increasingly complex,
rarely reviewed, rarely automated.

Problem 2: Most proofs are of
security bounds that aren't tight:
e.g. forking-lemma "security"
is pure deception for typical sizes.

Problem 3: "Security" definitions
prioritize simplicity over accuracy.
e.g. is MAC-pad-encrypt secure?

Problem 4: Maybe the only way
to achieve the fundamental goal
is to switch to weak primitives.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

1: "Proofs" have errors.
re increasingly complex,
viewed, rarely automated.

2: Most proofs are of
bounds that aren't tight:
ing-lemma "security"
deception for typical sizes.

3: "Security" definitions
e simplicity over accuracy.
MAC-pad-encrypt secure?

4: Maybe the only way
ve the fundamental goal
tch to weak primitives.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

## Crypto h

HTTPS.

fs" have errors.
ingly complex,
rely automated.

proofs are of
at aren't tight:
a "security"
for typical sizes.

rity" definitions
y over accuracy.
ncrypt secure?

e the only way
damental goal
ak primitives.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

## Crypto horror stor

HTTPS.

errors.
plex,
nated.

of
tight:
"

sizes.

itions
uracy.
cure?

way
goal
ves.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

## Crypto horror story #3

HTTPS.

## Some advice to crypto designers

Creating or evaluating a design?

**Think about the implementations.**

What will the implementors do?

What errors are likely to appear in implementations?

Can you compensate for this?

Is the design a primitive?

**Think about the protocols.**

Is the design a protocol?

**Think about the higher-level protocols.**

Will the system be secure?

## Crypto horror story #3

HTTPS.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!"
—The eavesdroppers will start forging (more) packets.

## Some advice to crypto designers

Creating or evaluating a design?
**Think about the implementations.**
What will the implementors do?
What errors are likely
to appear in implementations?
Can you compensate for this?

Is the design a primitive?
**Think about the protocols.**
Is the design a protocol?
**Think about the higher-level protocols.**
Will the system be secure?

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!"
—The eavesdroppers will start forging (more) packets.

"Then we'll know they're there!"
—Yes, we knew that already.
What we want is *security*.

## dvice to crypto designers

g or evaluating a design?
**bout the**
**entations.**
ill the implementors do?
rors are likely
ar in implementations?
compensate for this?

esign a primitive?
**bout the protocols.**
esign a protocol?
**bout the**
**evel protocols.**
system be secure?

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!"
—The eavesdroppers will start forging (more) packets.

"Then we'll know they're there!"
—Yes, we knew that already.
What we want is *security*.

## 2013.01

State-of-
are obvi

e.g. Defe
requires
assumpt

ypto designers

ting a design?

.
lementors do?
kely
mentations?
ate for this?

mitive?
**protocols.**
otocol?

**ocols.**
e secure?

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!"
—The eavesdroppers will start forging (more) packets.

"Then we'll know they're there!"
—Yes, we knew that already.
What we want is *security*.

2013.01 Green:

State-of-the-art T
are obviously unsa

e.g. Defense vs. B
requires "goofy ma
assumption" for pr

gners

ign?

do?

s?

s?

s.

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!"
—The eavesdroppers will start forging (more) packets.

"Then we'll know they're there!"
—Yes, we knew that already.
What we want is *security*.

2013.01 Green:

State-of-the-art TLS proofs are obviously unsatisfactory.

e.g. Defense vs. Bleichenbac requires "goofy made-up assumption" for proof.

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used.
But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!"
—The eavesdroppers will start forging (more) packets.

"Then we'll know they're there!"
—Yes, we knew that already.
What we want is *security*.

2013.01 Green:

State-of-the-art TLS proofs are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher requires "goofy made-up assumption" for proof.

## Crypto horror story #3

HTTPS.

`letsencrypt.org` is a huge improvement in HTTPS usability; will obviously be widely used. But is HTTPS actually secure?

"It's not so bad against passive eavesdroppers!" —The eavesdroppers will start forging (more) packets.

"Then we'll know they're there!" —Yes, we knew that already. What we want is *security*.

2013.01 Green:

State-of-the-art TLS proofs are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher requires "goofy made-up assumption" for proof.

But that was "the *good news* ... The problem with TLS is that we are cursed with *implementations*."

e.g. Defense vs. Bleichenbacher is in wrong order in OpenSSL. Does this allow timing attacks?

horror story #3

crypt.org is a huge
ment in HTTPS usability;
ously be widely used.
TTPS actually secure?

t so bad
passive eavesdroppers!"
avesdroppers will start
(more) packets.

we'll know they're there!"
we knew that already.
e want is *security*.

2013.01 Green:

State-of-the-art TLS proofs
are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher
requires "goofy made-up
assumption" for proof.

But that was "the *good*
*news* . . . The problem with
TLS is that we are cursed with
*implementations*."

e.g. Defense vs. Bleichenbacher
is in wrong order in OpenSSL.
Does this allow timing attacks?

2014.08
Weiss–S
Successf
exploitin
variation
NITROX

y #3

g is a huge
TTPS usability;
widely used.
tually secure?

vesdroppers!"
ers will start
ckets.

they're there!"
hat already.
*security*.

2013.01 Green:

State-of-the-art TLS proofs
are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher
requires "goofy made-up
assumption" for proof.

But that was "the *good*
*news* . . . The problem with
TLS is that we are cursed with
*implementations*."

e.g. Defense vs. Bleichenbacher
is in wrong order in OpenSSL.
Does this allow timing attacks?

2014.08 Meyer–Sc
Weiss–Schwenk–S
Successful Bleiche
exploiting analogo
variations in Java
NITROX SSL acce

e
bility;
d.
re?

rs!”
art

ere!”
y.

2013.01 Green:

State-of-the-art TLS proofs
are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher
requires “goofy made-up
assumption” for proof.

But that was “the *good*
*news* . . . The problem with
TLS is that we are cursed with
*implementations*.”

e.g. Defense vs. Bleichenbacher
is in wrong order in OpenSSL.
Does this allow timing attacks?

2014.08 Meyer–Somorovsky-
Weiss–Schwenk–Schinzel–Te
Successful Bleichenbacher a
exploiting analogous timing
variations in Java SSE, Cavi
NITROX SSL accelerator ch

2013.01 Green:

State-of-the-art TLS proofs are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher requires "goofy made-up assumption" for proof.

But that was "the *good* *news* . . . The problem with TLS is that we are cursed with *implementations*."

e.g. Defense vs. Bleichenbacher is in wrong order in OpenSSL. Does this allow timing attacks?

2014.08 Meyer–Somorovsky–Weiss–Schwenk–Schinzel–Tews: Successful Bleichenbacher attacks, exploiting analogous timing variations in Java SSE, Cavium NITROX SSL accelerator chip.

2013.01 Green:

State-of-the-art TLS proofs
are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher
requires "goofy made-up
assumption" for proof.

But that was "the *good*
*news* ... The problem with
TLS is that we are cursed with
*implementations*."

e.g. Defense vs. Bleichenbacher
is in wrong order in OpenSSL.
Does this allow timing attacks?

2014.08 Meyer–Somorovsky–
Weiss–Schwenk–Schinzel–Tews:
Successful Bleichenbacher attacks,
exploiting analogous timing
variations in Java SSE, Cavium
NITROX SSL accelerator chip.

The whole concept of a
"public-key cryptosystem"
is a historical accident,
dangerously unauthenticated.

2013.01 Green:

State-of-the-art TLS proofs
are obviously unsatisfactory.

e.g. Defense vs. Bleichenbacher
requires "goofy made-up
assumption" for proof.

But that was "the *good*
*news* . . . The problem with
TLS is that we are cursed with
*implementations*."

e.g. Defense vs. Bleichenbacher
is in wrong order in OpenSSL.
Does this allow timing attacks?

2014.08 Meyer–Somorovsky–
Weiss–Schwenk–Schinzel–Tews:
Successful Bleichenbacher attacks,
exploiting analogous timing
variations in Java SSE, Cavium
NITROX SSL accelerator chip.

The whole concept of a
"public-key cryptosystem"
is a historical accident,
dangerously unauthenticated.

Do we seriously believe that
we'll make HTTPS secure
by fixing the implementations?
**Fix the bad crypto design.**

Green:

-the-art TLS proofs
ously unsatisfactory.

ense vs. Bleichenbacher
"goofy made-up
ion" for proof.

t was "the *good*
. The problem with
hat we are cursed with
*entations*."

ense vs. Bleichenbacher
ong order in OpenSSL.
is allow timing attacks?

2014.08 Meyer–Somorovsky–Weiss–Schwenk–Schinzel–Tews: Successful Bleichenbacher attacks, exploiting analogous timing variations in Java SSE, Cavium NITROX SSL accelerator chip.

The whole concept of a "public-key cryptosystem" is a historical accident, dangerously unauthenticated.

Do we seriously believe that we'll make HTTPS secure by fixing the implementations? **Fix the bad crypto design.**

Exercise
failures

Renegot
Diginota
BEAST
Trustwa
CRIME
Lucky 1
RC4 key
TLS tru
gotofail
Triple H
Heartble
POODL
Winshoc

LS proofs
tisfactory.

leichenbacher
ade-up
roof.

*good*
blem with
e cursed with

leichenbacher
n OpenSSL.
ming attacks?

2014.08 Meyer–Somorovsky–Weiss–Schwenk–Schinzel–Tews: Successful Bleichenbacher attacks, exploiting analogous timing variations in Java SSE, Cavium NITROX SSL accelerator chip.

The whole concept of a "public-key cryptosystem" is a historical accident, dangerously unauthenticated.

Do we seriously believe that we'll make HTTPS secure by fixing the implementations? **Fix the bad crypto design.**

Exercise: How ma
failures can a *desi*

Renegotiation atta
Diginotar CA com
BEAST CBC attac
Trustwave HTTPS
CRIME compressio
Lucky 13 padding
RC4 keystream bia
TLS truncation.
gotofail signature-
Triple Handshake.
Heartbleed buffer
POODLE padding
Winshock buffer o

cher

ith

cher

L.

ks?

2014.08 Meyer–Somorovsky–Weiss–Schwenk–Schinzel–Tews: Successful Bleichenbacher attacks, exploiting analogous timing variations in Java SSE, Cavium NITROX SSL accelerator chip.

The whole concept of a "public-key cryptosystem" is a historical accident, dangerously unauthenticated.

Do we seriously believe that we'll make HTTPS secure by fixing the implementations? **Fix the bad crypto design.**

Exercise: How many of thes failures can a *designer* addre

Renegotiation attack.
Diginotar CA compromise.
BEAST CBC attack.
Trustwave HTTPS intercept
CRIME compression attack.
Lucky 13 padding/timing at
RC4 keystream bias.
TLS truncation.
gotofail signature-verificatio
Triple Handshake.
Heartbleed buffer overread.
POODLE padding-oracle att
Winshock buffer overflow.

2014.08 Meyer–Somorovsky–Weiss–Schwenk–Schinzel–Tews: Successful Bleichenbacher attacks, exploiting analogous timing variations in Java SSE, Cavium NITROX SSL accelerator chip.

The whole concept of a "public-key cryptosystem" is a historical accident, dangerously unauthenticated.

Do we seriously believe that we'll make HTTPS secure by fixing the implementations? **Fix the bad crypto design.**

Exercise: How many of these TLS failures can a *designer* address?

Renegotiation attack.
Diginotar CA compromise.
BEAST CBC attack.
Trustwave HTTPS interception.
CRIME compression attack.
Lucky 13 padding/timing attack.
RC4 keystream bias.
TLS truncation.
gotofail signature-verification bug.
Triple Handshake.
Heartbleed buffer overread.
POODLE padding-oracle attack.
Winshock buffer overflow.