Some challenges in heavyweight cipher design

Daniel J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

Protocol generates new AES-128 key k.

Protocol encrypts message block
m_1 as $\text{AES}_k(1) \oplus m_1$,
m_2 as $\text{AES}_k(2) \oplus m_2$,
m_3 as $\text{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\text{AES}_k(1)$.

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: "No! AES is secure! This is all safe!"

hallenges in
ight cipher design

. Bernstein

ty of Illinois at Chicago &
che Universiteit Eindhoven

Protocol generates
new AES-128 key k.

Protocol encrypts message block
m_1 as $\text{AES}_k(1) \oplus m_1$,
m_2 as $\text{AES}_k(2) \oplus m_2$,
m_3 as $\text{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\text{AES}_k(1)$.

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: "No!
AES is secure! This is all safe!"

Attacker
for, say,

Attacker
using fea
Attacker
data for

Is this 2
See 200

n
r design

n

is at Chicago &
siteit Eindhoven

Protocol generates new AES-128 key k.

Protocol encrypts message block
m_1 as $\mathrm{AES}_k(1) \oplus m_1$,
m_2 as $\mathrm{AES}_k(2) \oplus m_2$,
m_3 as $\mathrm{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\mathrm{AES}_k(1)$.

Can attacker deduce $\mathrm{AES}_k(20)$?
We constantly tell people: "No! AES is secure! This is all safe!"

Attacker learns AE
for, say, 2^{40} user k

Attacker finds *som*
using feasible 2^{88}
Attacker decrypts,
data for that user.

Is this 2^{128} "secur
See 2002 Biham "

ago &
hoven

Protocol generates new AES-128 key k.

Protocol encrypts message block
m_1 as $\text{AES}_k(1) \oplus m_1$,
m_2 as $\text{AES}_k(2) \oplus m_2$,
m_3 as $\text{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\text{AES}_k(1)$.

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: "No! AES is secure! This is all safe!"

Attacker learns $\text{AES}_k(1)$ for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computati
Attacker decrypts, maybe fo
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisi

Protocol generates new AES-128 key k.

Protocol encrypts message block
m_1 as $\text{AES}_k(1) \oplus m_1$,
m_2 as $\text{AES}_k(2) \oplus m_2$,
m_3 as $\text{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\text{AES}_k(1)$.

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: "No! AES is secure! This is all safe!"

Attacker learns $\text{AES}_k(1)$ for, say, 2^{40} user keys k.

Attacker finds *some* user key using feasible 2^{88} computation.
Attacker decrypts, maybe forges, data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Protocol generates
new AES-128 key k.

Protocol encrypts message block
m_1 as $\mathrm{AES}_k(1) \oplus m_1$,
m_2 as $\mathrm{AES}_k(2) \oplus m_2$,
m_3 as $\mathrm{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\mathrm{AES}_k(1)$.

Can attacker deduce $\mathrm{AES}_k(20)$?
We constantly tell people: "No!
AES is secure! This is all safe!"

Attacker learns $\mathrm{AES}_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols
by trying to randomize everything.

Protocol generates new AES-128 key k.

Protocol encrypts message block
m_1 as $\text{AES}_k(1) \oplus m_1$,
m_2 as $\text{AES}_k(2) \oplus m_2$,
m_3 as $\text{AES}_k(3) \oplus m_3$,
etc. Also authenticates.

First block m_1 is predictable:
`GET / HTTP/1.1\r\n`
Attacker learns $\text{AES}_k(1)$.

Can attacker deduce $\text{AES}_k(20)$?
We constantly tell people: "No! AES is secure! This is all safe!"

Attacker learns $\text{AES}_k(1)$ for, say, 2^{40} user keys k.

Attacker finds *some* user key using feasible 2^{88} computation.
Attacker decrypts, maybe forges, data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

l generates
S-128 key k.

l encrypts message block
ES$_k(1) \oplus m_1$,
ES$_k(2) \oplus m_2$,
ES$_k(3) \oplus m_3$,
o authenticates.

ck m_1 is predictable:
`TTP/1.1\r\n`
r learns AES$_k(1)$.

acker deduce AES$_k(20)$?
stantly tell people: "No!
secure! This is all safe!"

Attacker learns AES$_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols
by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

Another
about 1
quantum

Grover f
using 2^{6}
on a sm

s
k.

message block
m_1,
m_2,
m_3,
cates.

predictable:
`\n`
ES$_k(1)$.

ce AES$_k(20)$?
people: “No!
is is all safe!”

Attacker learns AES$_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} “security”?
See 2002 Biham “key collisions”.

Fragile fix: Complicate protocols
by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

Another reason to
about 128-bit ciph
quantum computi

Grover finds k fro
using 2^{64} iteration
on a small quantu

lock

e:

20)?
'No!
fe!"

Attacker learns $\text{AES}_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols
by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

Another reason to be concer
about 128-bit cipher keys:
quantum computing.

Grover finds k from $\text{AES}_k(1$
using 2^{64} iterations
on a small quantum process

Attacker learns $\text{AES}_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols
by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

Another reason to be concerned
about 128-bit cipher keys:
quantum computing.

Grover finds k from $\text{AES}_k(1)$
using 2^{64} iterations
on a small quantum processor.

Attacker learns $\text{AES}_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols
by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

Another reason to be concerned
about 128-bit cipher keys:
quantum computing.

Grover finds k from $\text{AES}_k(1)$
using 2^{64} iterations
on a small quantum processor.

Parallelize: N^2 processors,
each running $2^{64}/N$ iterations.
1999 Zalka claims this is optimal.

Attacker learns $\text{AES}_k(1)$
for, say, 2^{40} user keys k.

Attacker finds *some* user key
using feasible 2^{88} computation.
Attacker decrypts, maybe forges,
data for that user.

Is this 2^{128} "security"?
See 2002 Biham "key collisions".

Fragile fix: Complicate protocols
by trying to randomize everything.

Much simpler fix: 256-bit keys.
(Side discussion: Is 192 enough?)

Another reason to be concerned
about 128-bit cipher keys:
quantum computing.

Grover finds k from $\text{AES}_k(1)$
using 2^{64} iterations
on a small quantum processor.

Parallelize: N^2 processors,
each running $2^{64}/N$ iterations.
1999 Zalka claims this is optimal.

Multiple targets should allow
much better parallelization.
Related algos: 2009 Bernstein;
2004 Grover–Radhakrishnan.

r learns $\text{AES}_k(1)$
2^{40} user keys k.

r finds *some* user key
asible 2^{88} computation.
r decrypts, maybe forges,
that user.

^{128} "security"?
2 Biham "key collisions".

ix: Complicate protocols
g to randomize everything.

mpler fix: 256-bit keys.
scussion: Is 192 enough?)

Another reason to be concerned
about 128-bit cipher keys:
quantum computing.

Grover finds k from $\text{AES}_k(1)$
using 2^{64} iterations
on a small quantum processor.

Parallelize: N^2 processors,
each running $2^{64}/N$ iterations.
1999 Zalka claims this is optimal.

Multiple targets should allow
much better parallelization.
Related algos: 2009 Bernstein;
2004 Grover–Radhakrishnan.

Should M

To authe

Comput
different
e.g., $r^4 n$
where r

Generate
$s_n = \text{AE}$

Add to
$r^4 m_1 +$

Widely
consider

ES$_k(1)$
keys k.

ne user key
computation.
maybe forges,

ity"?
"key collisions".

icate protocols
mize everything.

256-bit keys.
s 192 enough?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds k from $\text{AES}_k(1)$ using 2^{64} iterations on a small quantum processor.

Parallelize: N^2 processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.

Multiple targets should allow much better parallelization. Related algos: 2009 Bernstein; 2004 Grover–Radhakrishnan.

Should MACs have

To authenticate (r

Compute function
differential probab
e.g., $r^4 m_1 + r^3 m_2$
where r is secret.

Generate a **one-ti**
$s_n = \text{AES}_k(n)$ fron

Add to obtain MA
$r^4 m_1 + r^3 m_2 + r^2$

Widely deployed fo
consider, e.g., GCI

y
on.
rges,

ons".

ocols
ything.

eys.
ugh?)

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds k from $\text{AES}_k(1)$ using 2^{64} iterations on a small quantum processor.

Parallelize: N^2 processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.

Multiple targets should allow much better parallelization. Related algos: 2009 Bernstein; 2004 Grover–Radhakrishnan.

Should MACs have nonces?

To authenticate $(m_1, m_2, m_3$

Compute function with smal differential probabilities. e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 +$ where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master k

Add to obtain MAC: $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$

Widely deployed for speed: consider, e.g., GCM.

Another reason to be concerned about 128-bit cipher keys: quantum computing.

Grover finds k from $\text{AES}_k(1)$ using 2^{64} iterations on a small quantum processor.

Parallelize: N^2 processors, each running $2^{64}/N$ iterations. 1999 Zalka claims this is optimal.

Multiple targets should allow much better parallelization. Related algos: 2009 Bernstein; 2004 Grover–Radhakrishnan.

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities. e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC: $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

reason to be concerned
28-bit cipher keys:
n computing.

inds k from $\text{AES}_k(1)$
^{4} iterations
all quantum processor.

ze: N^2 processors,
nning $2^{64}/N$ iterations.
lka claims this is optimal.

targets should allow
etter parallelization.
algos: 2009 Bernstein;
over–Radhakrishnan.

<u>Should MACs have nonces?</u>

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed:
consider, e.g., GCM.

2006 Jo
ntwice i
⇒ attac
can easi

be concerned
her keys:
ng.

m $\text{AES}_k(1)$
s
m processor.

ocessors,
N iterations.
this is optimal.

hould allow
lelization.
09 Bernstein;
akrishnan.

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbid
ntwice in GCM ⇒
⇒ attacker figures
can easily forge m

rned

)

or.

ns.

timal.

v

ein;

.

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack
ntwice in GCM $\Rightarrow$ repeated
$\Rightarrow$ attacker figures out r,
can easily forge messages.

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack": ntwice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack": $n$twice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response:
$\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$ "seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack": n twice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response:
$\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$ "seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"?

Should MACs have nonces?

To authenticate (m_1, m_2, m_3, m_4):

Compute function with small differential probabilities.
e.g., $r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4$, where r is secret.

Generate a **one-time** key $s_n = \text{AES}_k(n)$ from master key k.

Add to obtain MAC:
$r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

Widely deployed for speed: consider, e.g., GCM.

2006 Joux "forbidden attack": ntwice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response:
$\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$
"seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"?
Forgery chance $\le \delta + \epsilon$ where ϵ is AES PRF insecurity and $\delta \approx q^2 L / 2^{128}$ for message lengths $\le L$.

MACs have nonces?

enticate (m_1, m_2, m_3, m_4):

e function with small
ial probabilities.
$m_1 + r^3 m_2 + r^2 m_3 + r m_4$,
is secret.

e a **one-time** key
$S_k(n)$ from master key k.

obtain MAC:
$r^3 m_2 + r^2 m_3 + r m_4 + s_n$.

deployed for speed:
, e.g., GCM.

2006 Joux "forbidden attack":
ntwice in GCM $\Rightarrow$ repeated s_n
$\Rightarrow$ attacker figures out r,
can easily forge messages.

Joux's suggested response:
$\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$
"seems a safe option". (Also
suggested and analyzed in, e.g.,
2000 Bernstein; earlier refs?)

Is this 2^{128} "security"?
Forgery chance $\leq \delta + \epsilon$ where
ϵ is AES PRF insecurity and
$\delta \approx q^2 L / 2^{128}$
for message lengths $\leq L$.

ϵ is at le
Solution
(2005 B

e nonces?

m_1, m_2, m_3, m_4):

with small

ilities.

$+ r^2 m_3 + r m_4$,

me key

n master key k.

AC:

$m_3 + r m_4 + s_n$.

or speed:

M.

2006 Joux "forbidden attack":
ntwice in GCM $\Rightarrow$ repeated s_n
$\Rightarrow$ attacker figures out r,
can easily forge messages.

Joux's suggested response:
$\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$
"seems a safe option". (Also
suggested and analyzed in, e.g.,
2000 Bernstein; earlier refs?)

Is this 2^{128} "security"?
Forgery chance $\le \delta + \epsilon$ where
ϵ is AES PRF insecurity and
$\delta \approx q^2 L / 2^{128}$
for message lengths $\le L$.

ϵ is at least $q(q -$
Solution: better P
(2005 Bernstein),

$_3, m_4)$:

ll

$+ r m_4$,

key k.

$+ s_n$.

2006 Joux "forbidden attack": ntwice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response:
$\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$
"seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"?
Forgery chance $\le \delta + \epsilon$ where
ϵ is AES PRF insecurity and
$\delta \approx q^2 L / 2^{128}$
for message lengths $\le L$.

ϵ is at least $q(q-1)/2^{129}$.
Solution: better PRP/PRF s
(2005 Bernstein), ok for $q \approx$

2006 Joux "forbidden attack": $n$twice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response: $\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$ "seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"? Forgery chance $\le \delta + \epsilon$ where ϵ is AES PRF insecurity and $\delta \approx q^2 L / 2^{128}$ for message lengths $\le L$.

ϵ is at least $q(q-1)/2^{129}$. Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

2006 Joux "forbidden attack": ntwice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response: $\mathrm{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$ "seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"? Forgery chance $\le \delta + \epsilon$ where ϵ is AES PRF insecurity and $\delta \approx q^2 L / 2^{128}$ for message lengths $\le L$.

ϵ is at least $q(q-1)/2^{129}$. Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

2006 Joux "forbidden attack": $n$twice in GCM $\Rightarrow$ repeated s_n $\Rightarrow$ attacker figures out r, can easily forge messages.

Joux's suggested response: $\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$ "seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"? Forgery chance $\le \delta + \epsilon$ where ϵ is AES PRF insecurity and $\delta \approx q^2 L/2^{128}$ for message lengths $\le L$.

ϵ is at least $q(q-1)/2^{129}$. Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: "Switch keys!"

2006 Joux "forbidden attack": n twice in GCM ⇒ repeated s_n ⇒ attacker figures out r, can easily forge messages.

Joux's suggested response: $\text{AES}_k(r^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$ "seems a safe option". (Also suggested and analyzed in, e.g., 2000 Bernstein; earlier refs?)

Is this 2^{128} "security"? Forgery chance $\le \delta + \epsilon$ where ϵ is AES PRF insecurity and $\delta \approx q^2 L/2^{128}$ for message lengths $\le L$.

ϵ is at least $q(q-1)/2^{129}$. Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: "Switch keys!"

Much simpler: 256-bit blocks.

2014 Bernstein–Chou "Auth256": 29 bit ops/message bit for differential probability $<2^{-255}$. Or try EHC from 2013 Nandi?

ux "forbidden attack":
n GCM ⇒ repeated s_n
ker figures out r,
ly forge messages.

uggested response:
$^4 m_1 + r^3 m_2 + r^2 m_3 + r m_4)$
a safe option". (Also
ed and analyzed in, e.g.,
rnstein; earlier refs?)

^{128} "security"?
chance $\leq \delta + \epsilon$ where
PRF insecurity and
$/2^{128}$
age lengths $\leq L$.

ϵ is at least $q(q-1)/2^{129}$.
Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: "Switch keys!"

Much simpler: 256-bit blocks.

2014 Bernstein–Chou "Auth256": 29 bit ops/message bit for differential probability $<2^{-255}$. Or try EHC from 2013 Nandi?

Improvin

Tor wan
easy-to-i
encumbe
509-byte
(But cur
so can c

Also: se
from eac

Tor is co
of AEZ c

See, e.g.
from RW

den attack":
repeated s_n
s out r,
essages.

response:
$_2 + r^2 m_3 + r m_4)$
on". (Also
lyzed in, e.g.,
arlier refs?)

ity"?
$\delta + \epsilon$ where
curity and

ns $\le L$.

ϵ is at least $q(q-1)/2^{129}$.
Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: "Switch keys!"

Much simpler: 256-bit blocks.

2014 Bernstein–Chou "Auth256": 29 bit ops/message bit for differential probability $<2^{-255}$. Or try EHC from 2013 Nandi?

Improving Tor

Tor wants "fast, p
easy-to-implement
encumbered, side-
509-byte blooock
(But current ciphe
so can consider co

Also: secure chain
from each blooock

Tor is considering
of AEZ or HHFHF

See, e.g., Mathew
from RWC 2013 a

k":
s_n

$+ rm_4)$
o
e.g.,
)

ere

ϵ is at least $q(q-1)/2^{129}$.
Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large. (Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: "Switch keys!"

Much simpler: 256-bit blocks.

2014 Bernstein–Chou "Auth256": 29 bit ops/message bit for differential probability $<2^{-255}$.
Or try EHC from 2013 Nandi?

Improving Tor

Tor wants "fast, proven, sec
easy-to-implement, non-pate
encumbered, side-channel-fr
509-byte blooock cipher.
(But current cipher is a disa
so can consider compromises

Also: secure chaining
from each blooock to the ne

Tor is considering deploymer
of AEZ or HHFHFH in 2016

See, e.g., Mathewson talks
from RWC 2013 and RWC 2

ϵ is at least $q(q-1)/2^{129}$.
Solution: better PRP/PRF switch (2005 Bernstein), ok for $q \approx 2^{64}$.

δ is still unacceptably large.
(Show that this is tight? See, e.g., 2005 Ferguson GCM attack.)

Fragile solution: "Switch keys!"

Much simpler: 256-bit blocks.

2014 Bernstein–Chou "Auth256": 29 bit ops/message bit for differential probability $<2^{-255}$.
Or try EHC from 2013 Nandi?

Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte blooock cipher.
(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each blooock to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.

east $q(q-1)/2^{129}$.
: better PRP/PRF switch
ernstein), ok for $q \approx 2^{64}$.

unacceptably large.
hat this is tight? See,
05 Ferguson GCM attack.)

solution: "Switch keys!"

mpler: 256-bit blocks.

rnstein–Chou "Auth256":
ps/message bit for
ial probability $<2^{-255}$.
HC from 2013 Nandi?

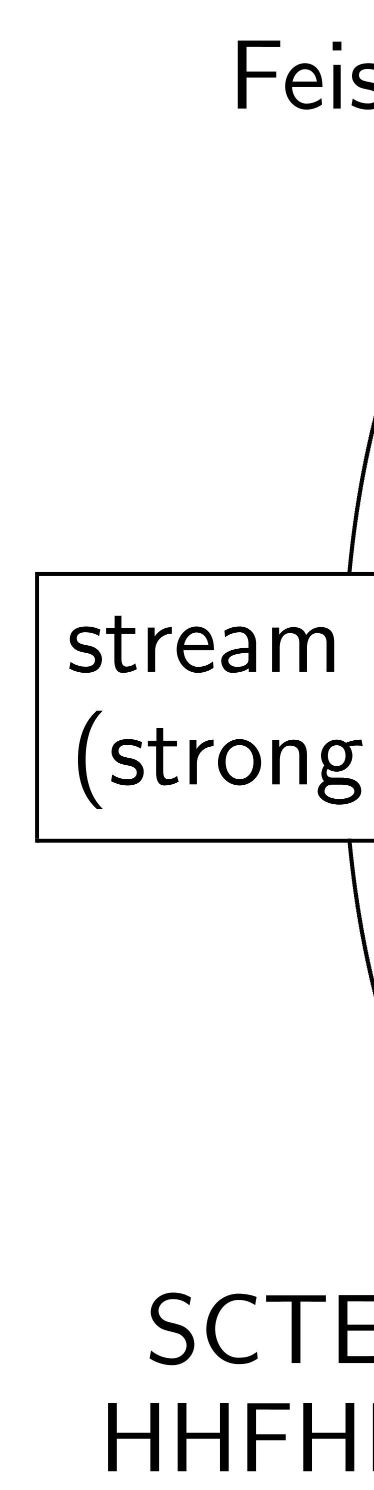
Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte blooock cipher. (But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each blooock to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.



1)/2^{129}.
RP/PRF switch
ok for $q \approx 2^{64}$.

ably large.
tight? See,
on GCM attack.)

"Switch keys!"

6-bit blocks.

hou "Auth256":
ge bit for
ility $<2^{-255}$.
2013 Nandi?

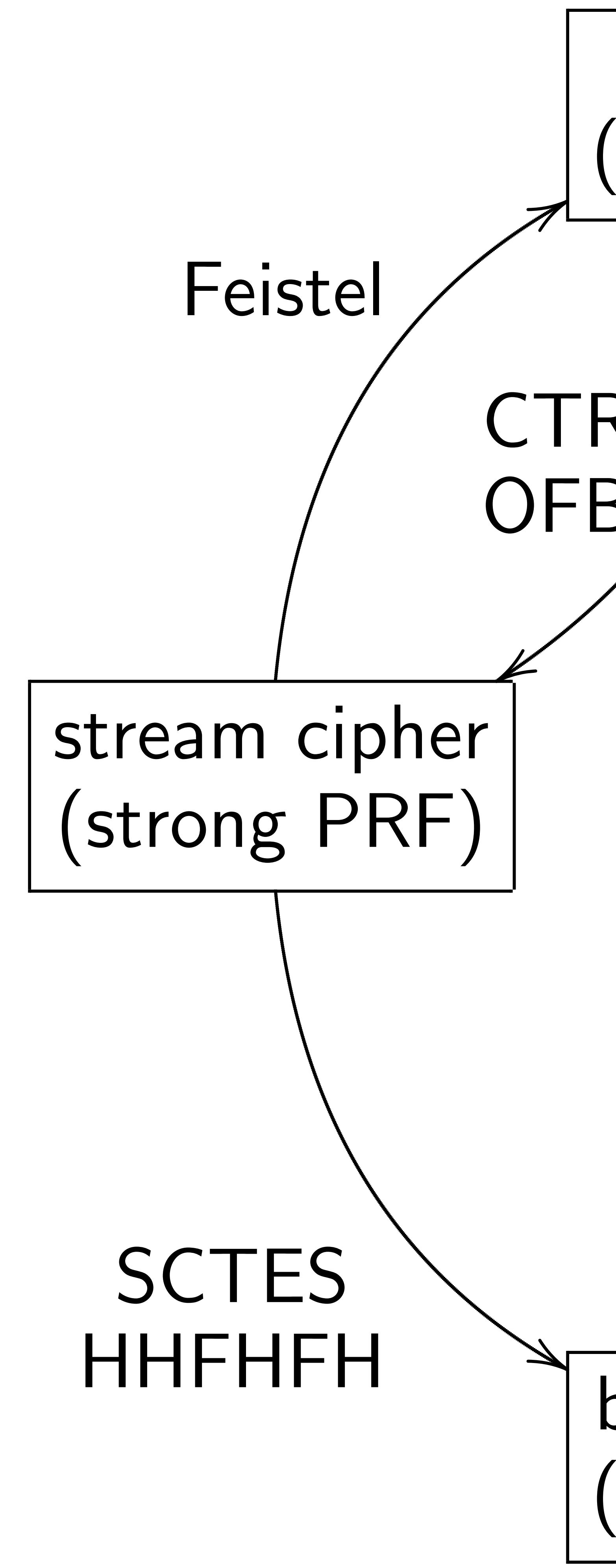
Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte blooock cipher. (But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each blooock to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.



switch
2^{64}.

e,
ttack.)

ys!"

ks.

256":

^{55}.

di?

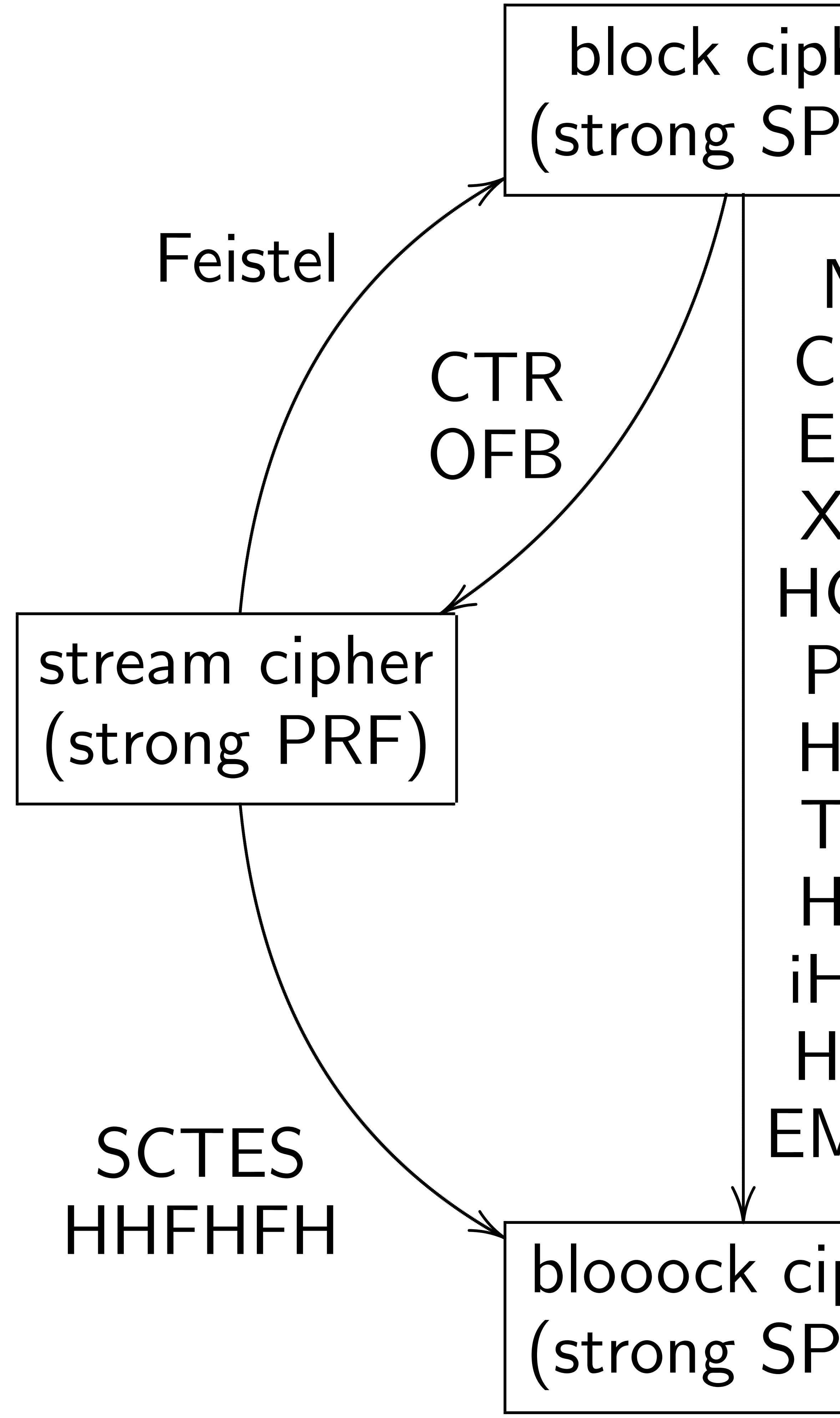
Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte blooock cipher.
(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each blooock to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.



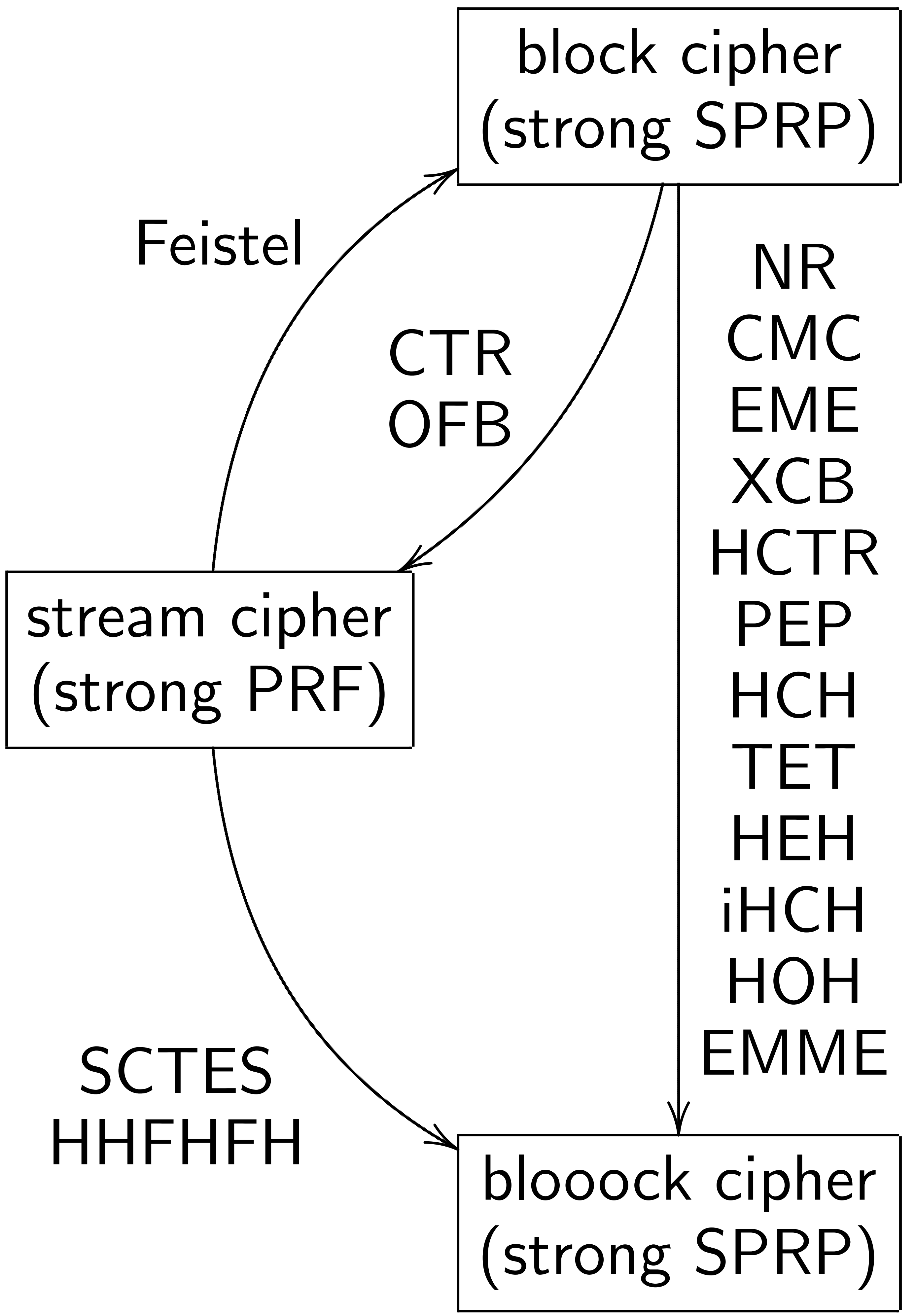
Improving Tor

Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte blooock cipher.
(But current cipher is a disaster, so can consider compromises.)

Also: secure chaining from each blooock to the next.

Tor is considering deployment of AEZ or HHFHFH in 2016.

See, e.g., Mathewson talks from RWC 2013 and RWC 2016.



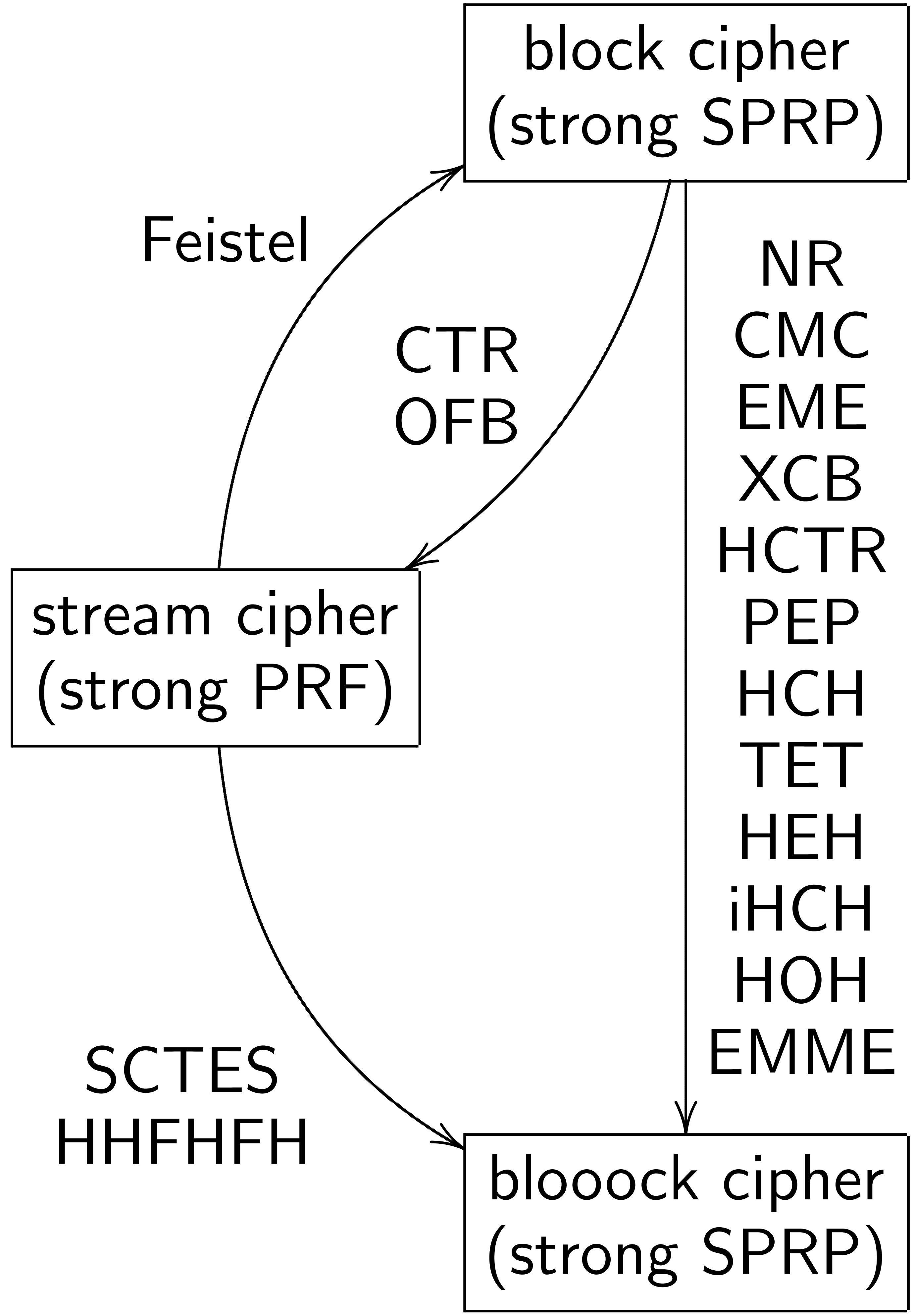
ng Tor

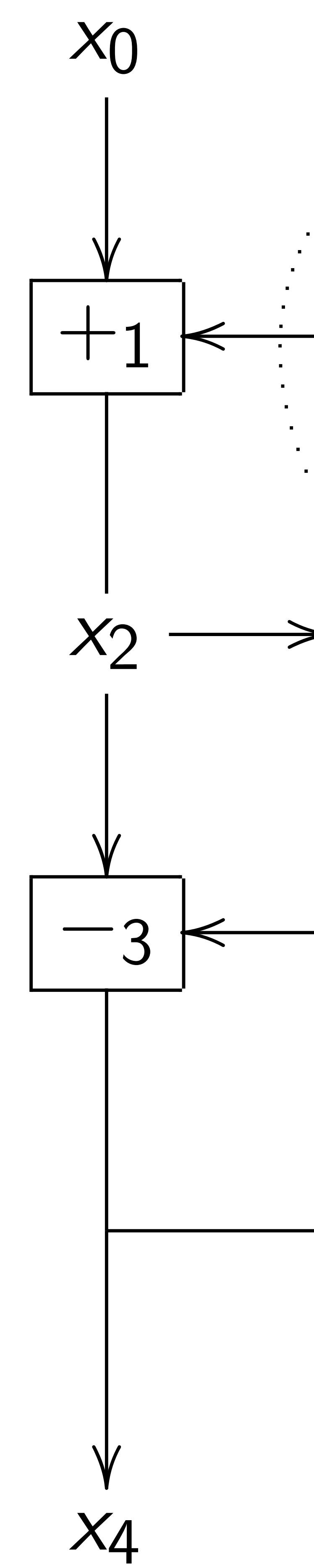
ts "fast, proven, secure,
implement, non-patent-
ered, side-channel-free"
e blooock cipher.
rrent cipher is a disaster,
onsider compromises.)

cure chaining
ch blooock to the next.

onsidering deployment
or HHFHFH in 2016.

., Mathewson talks
WC 2013 and RWC 2016.



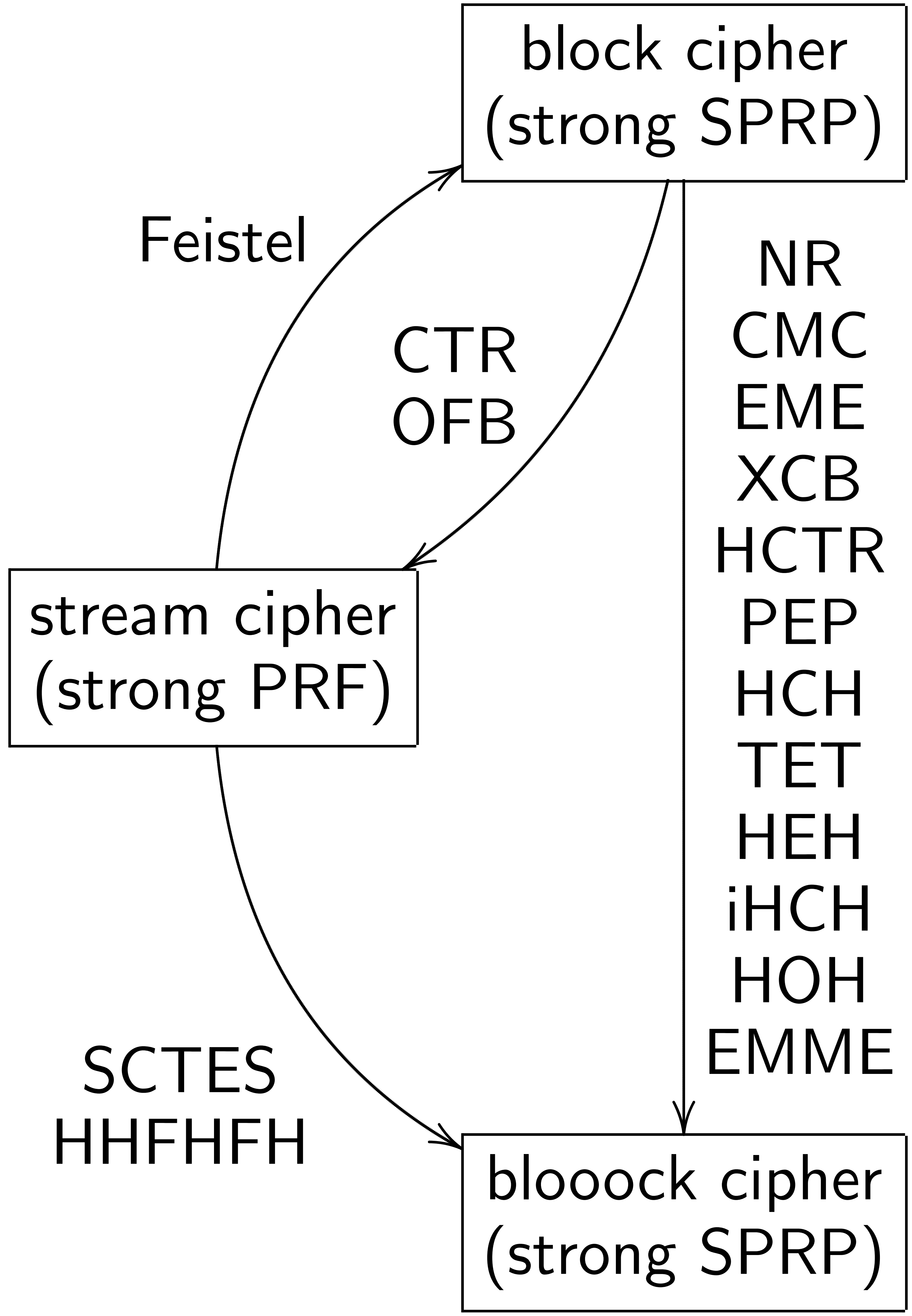


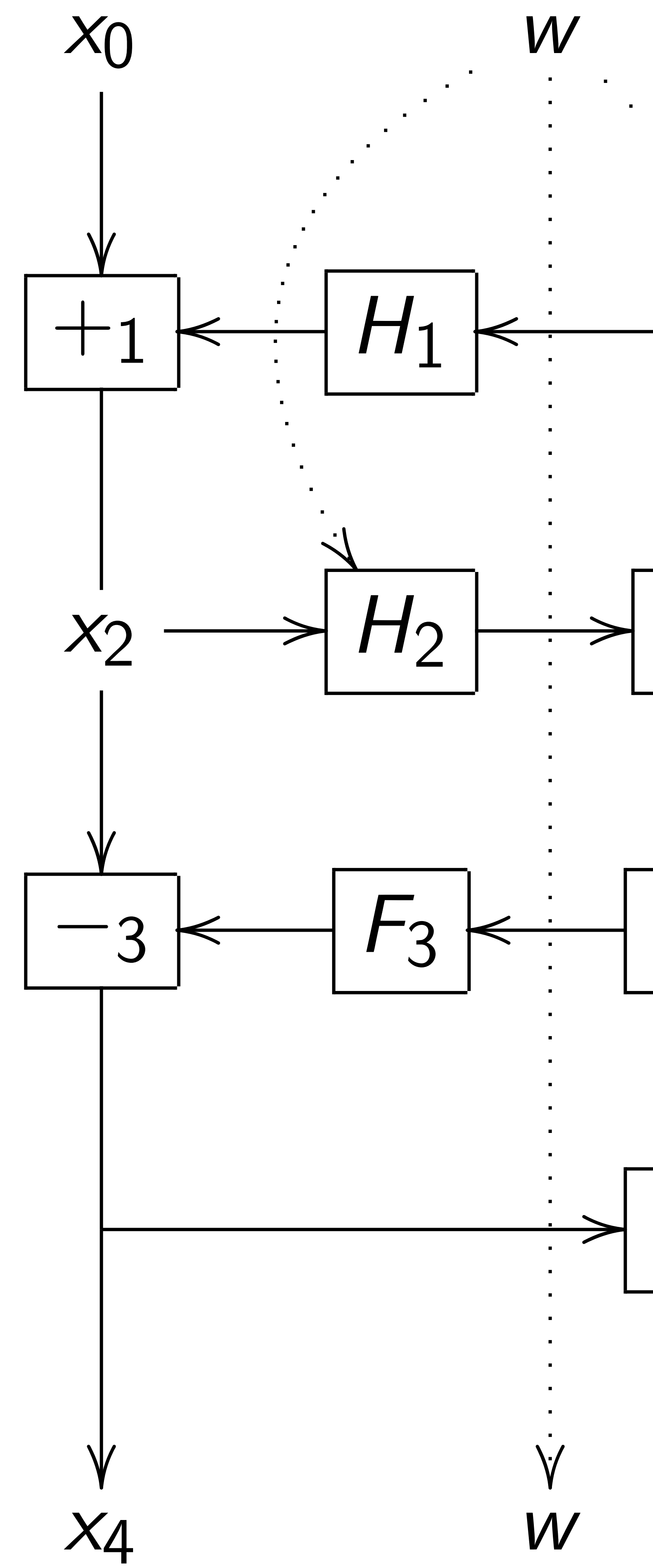
roven, secure,
, non-patent-
channel-free"
cipher.
er is a disaster,
mpromises.)

ing
k to the next.

deployment
FH in 2016.

son talks
nd RWC 2016.





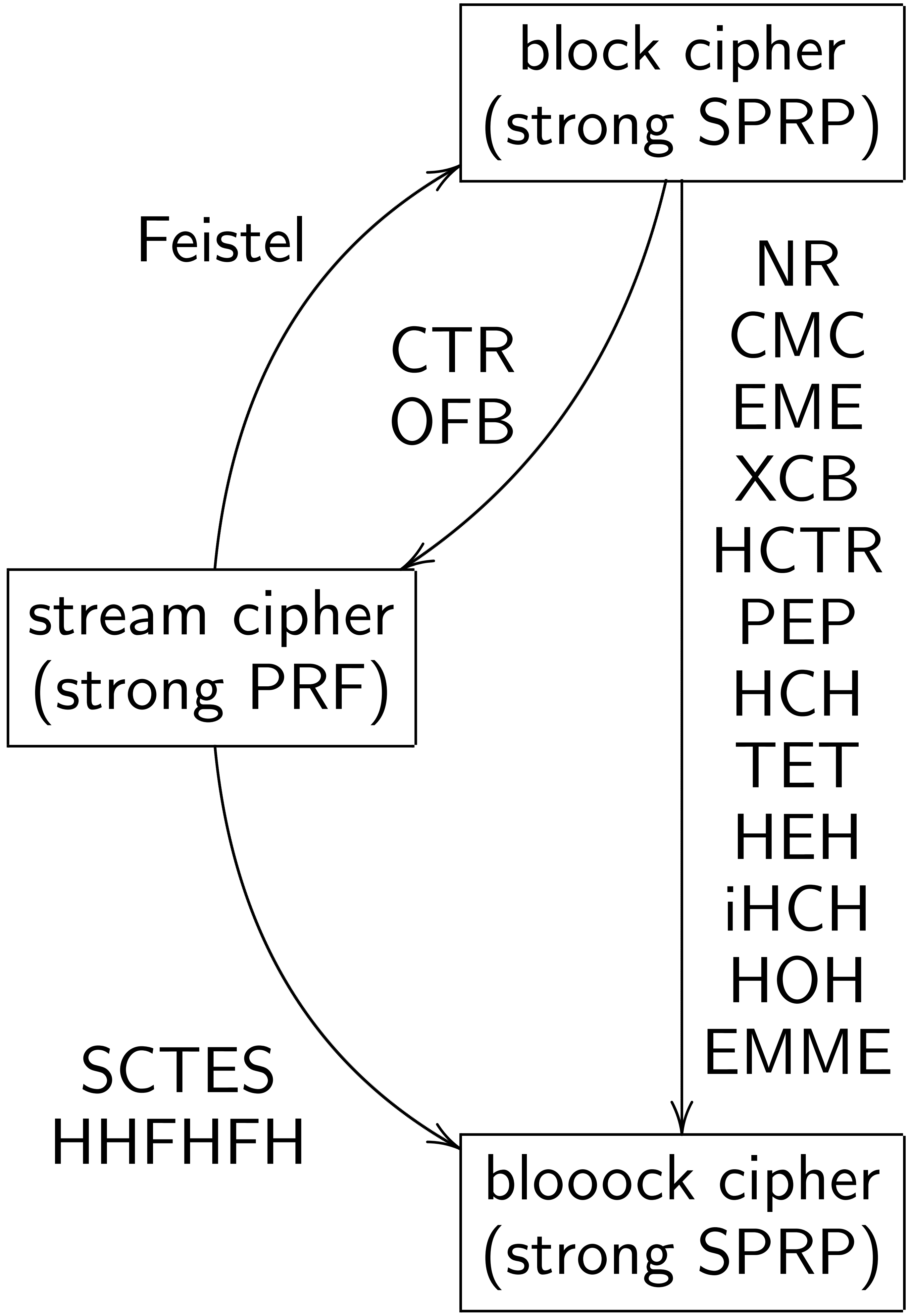
ure,
ent-
ee"

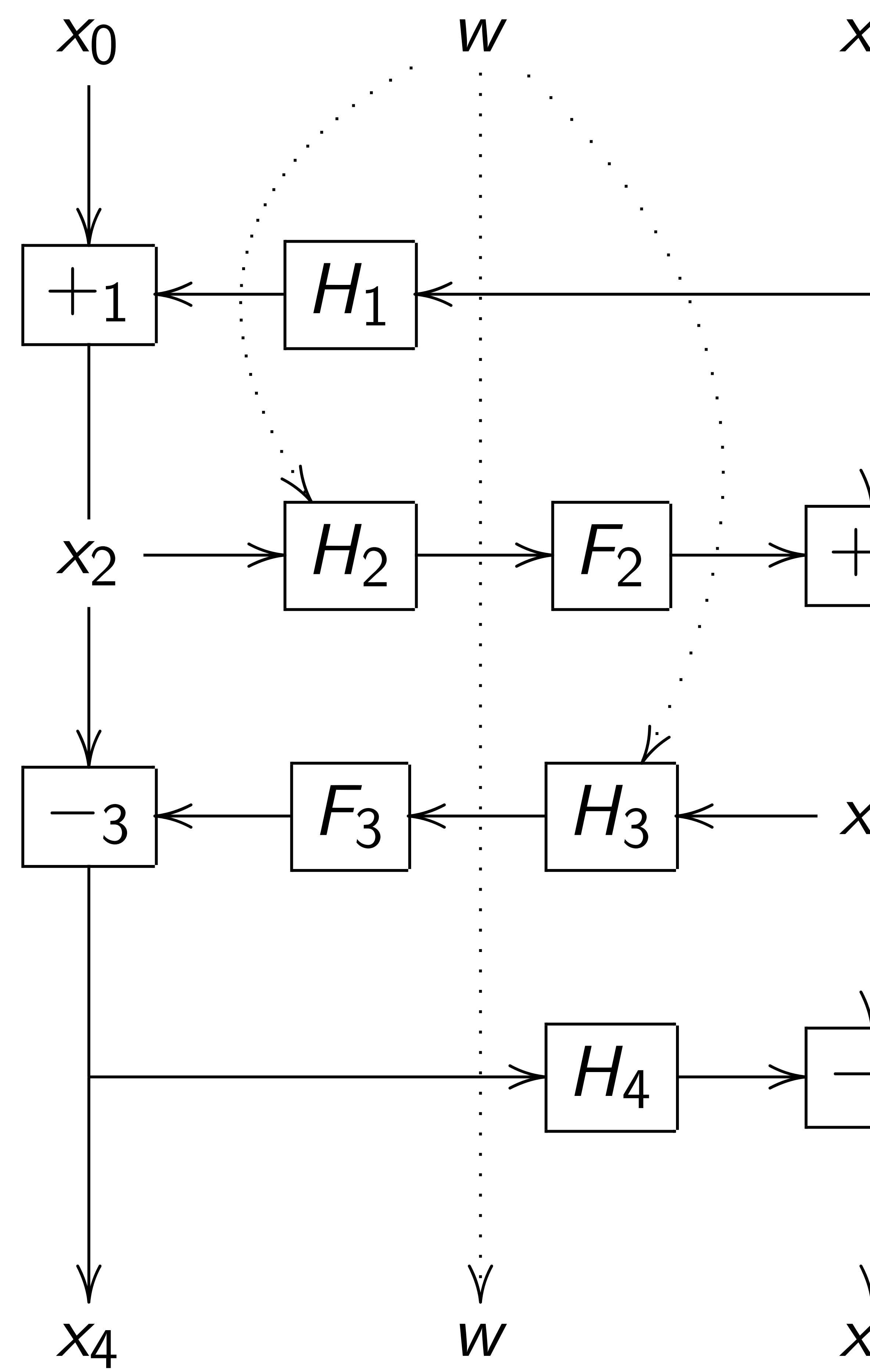
ster,
s.)

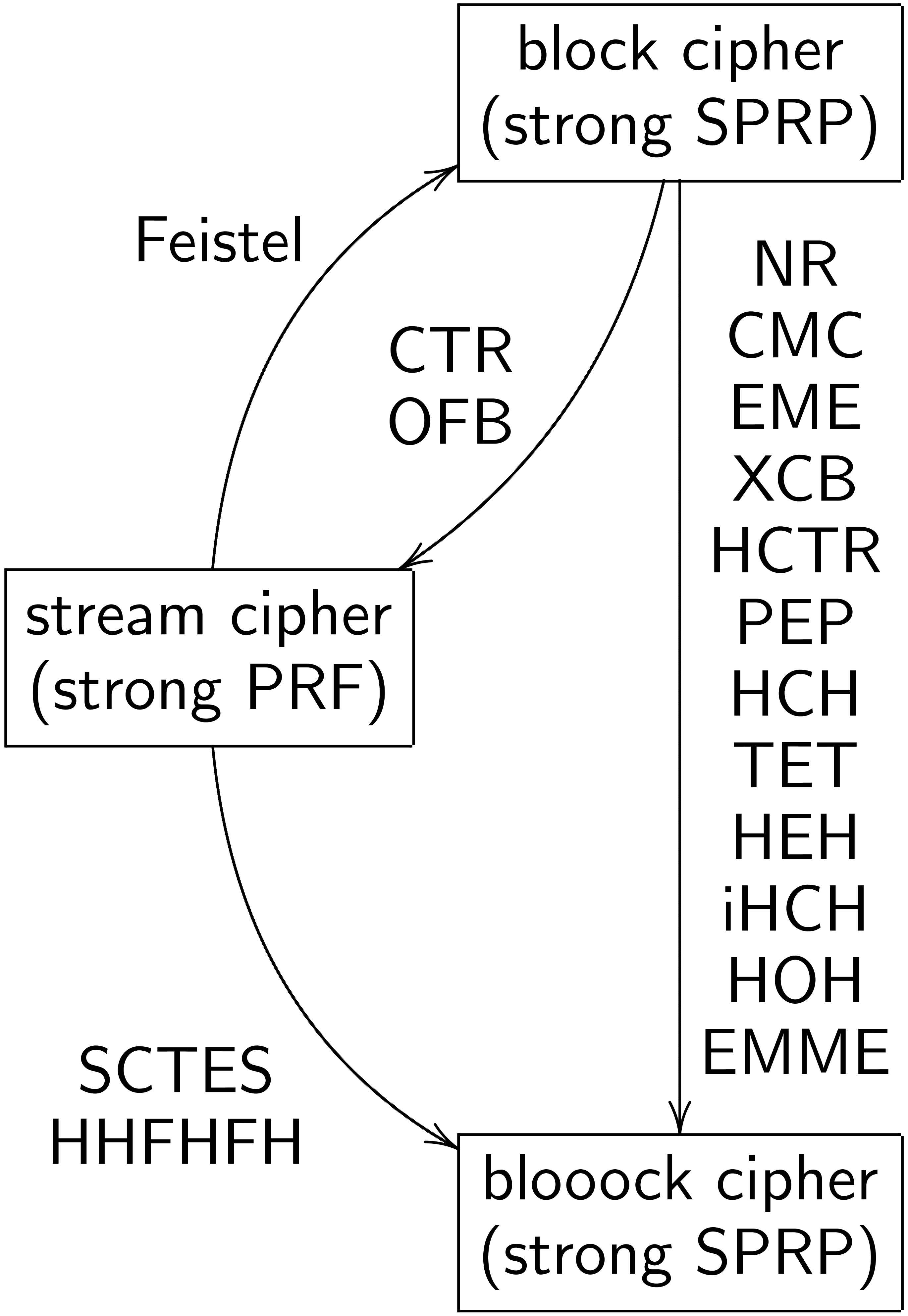
ext.

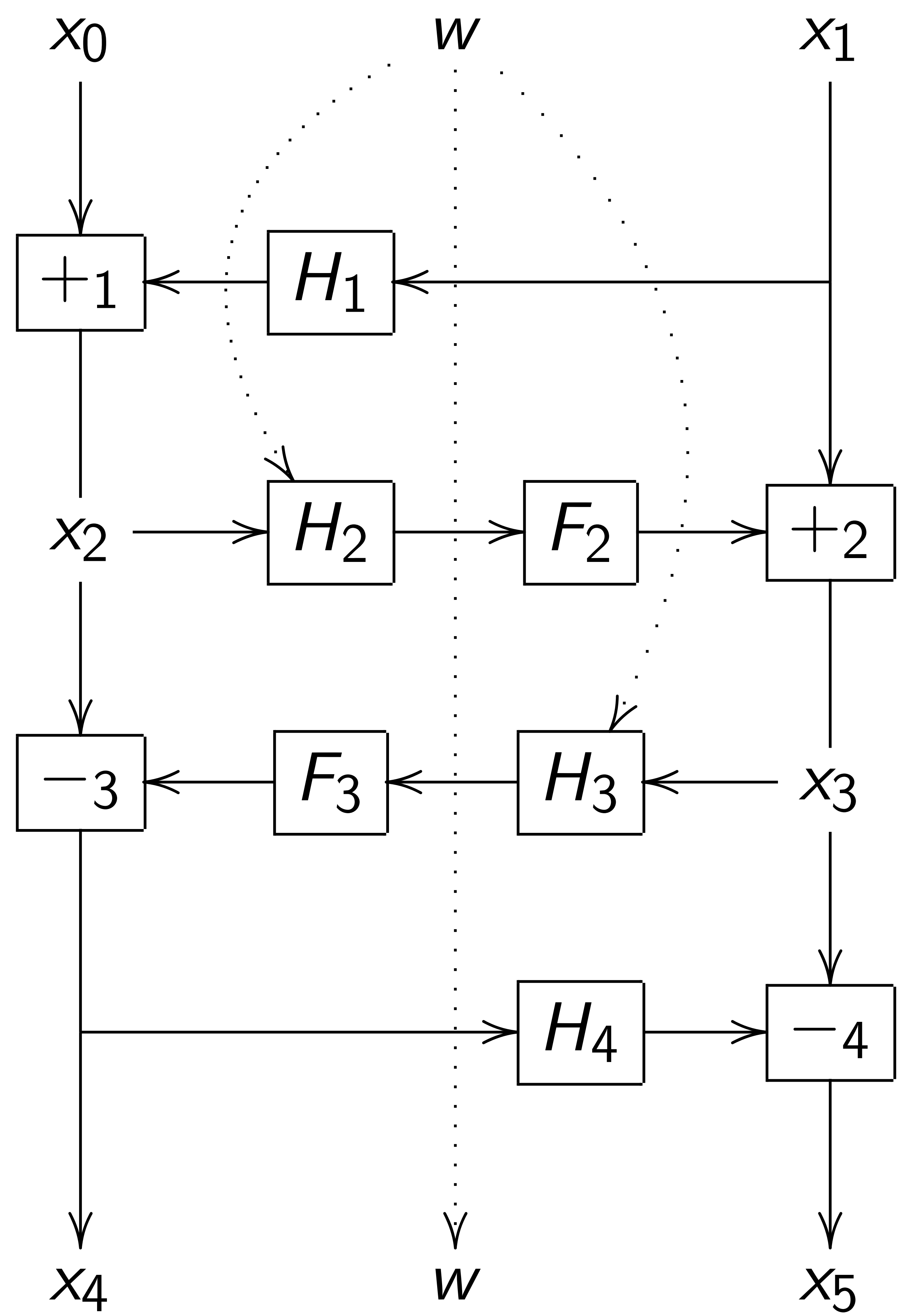
nt
6.

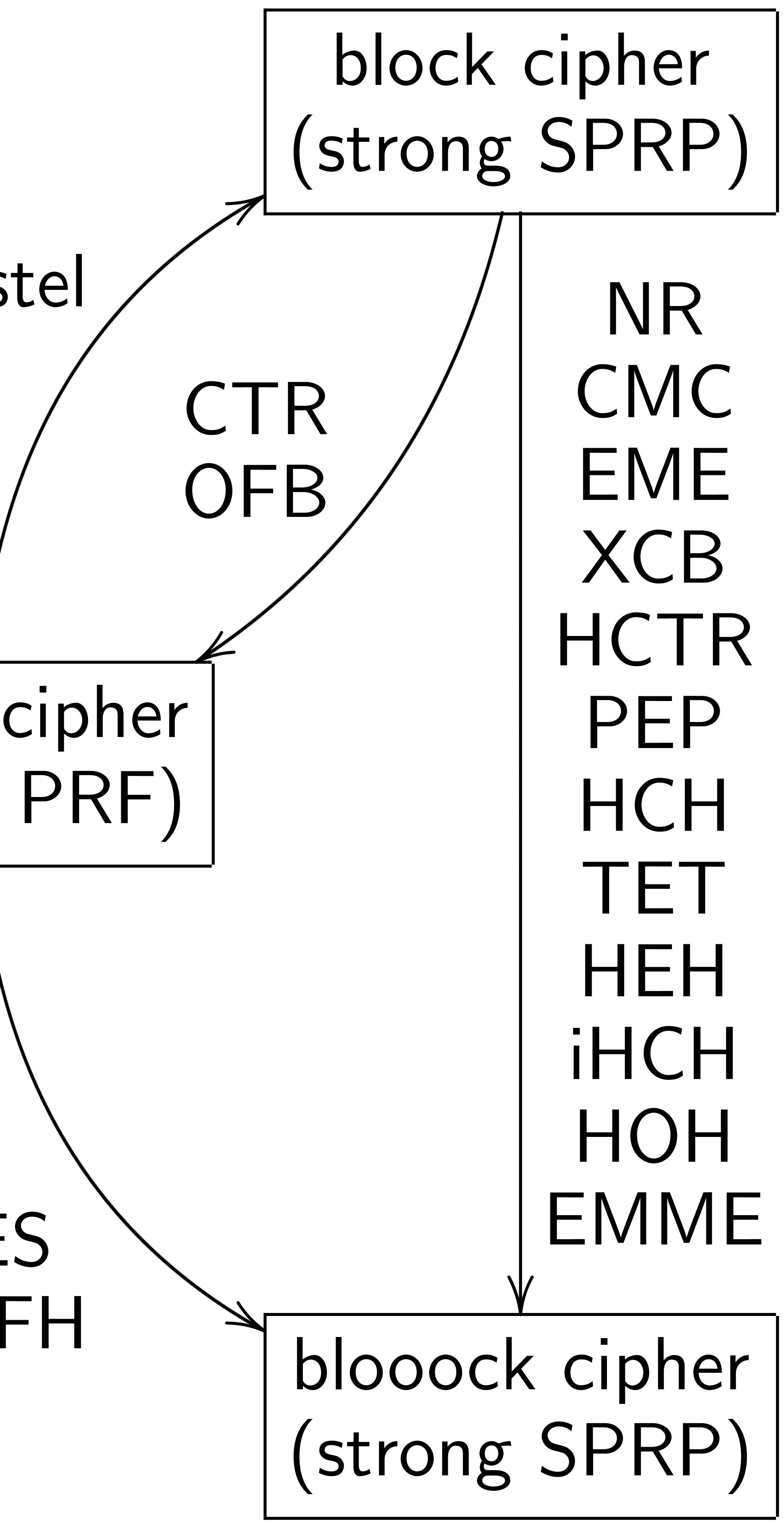
2016.

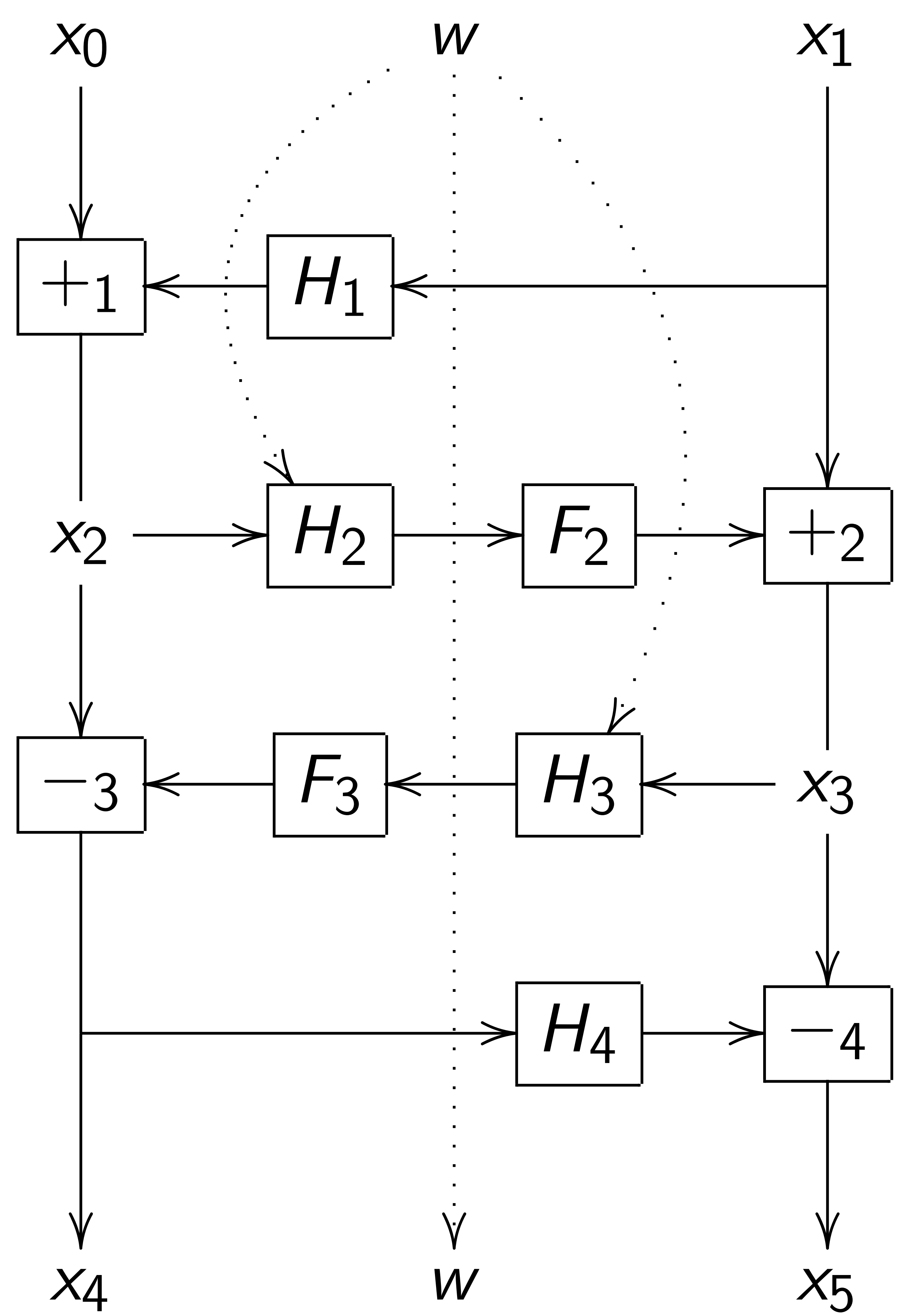





block cipher
(strong SPRP)
Feistel
CTR
OFB
NR
CMC
EME
XCB
HCTR
PEP
HCH
TET
HEH
iHCH
HOH
EMME
stream cipher
(strong PRF)
SCTES
HHFHFH
blooock cipher
(strong SPRP)


x_0
w
x_1
$+_1$
H_1
x_2
H_2
F_2
$+_2$
$-_3$
F_3
H_3
x_3
H_4
$-_4$
x_4
w
x_5

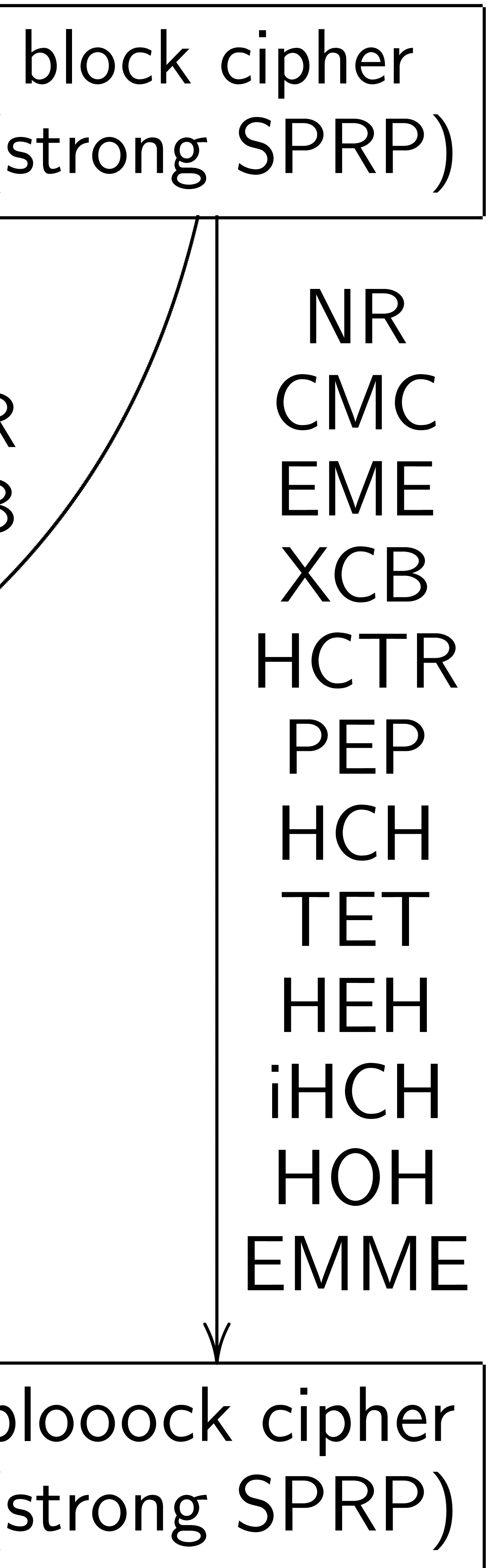


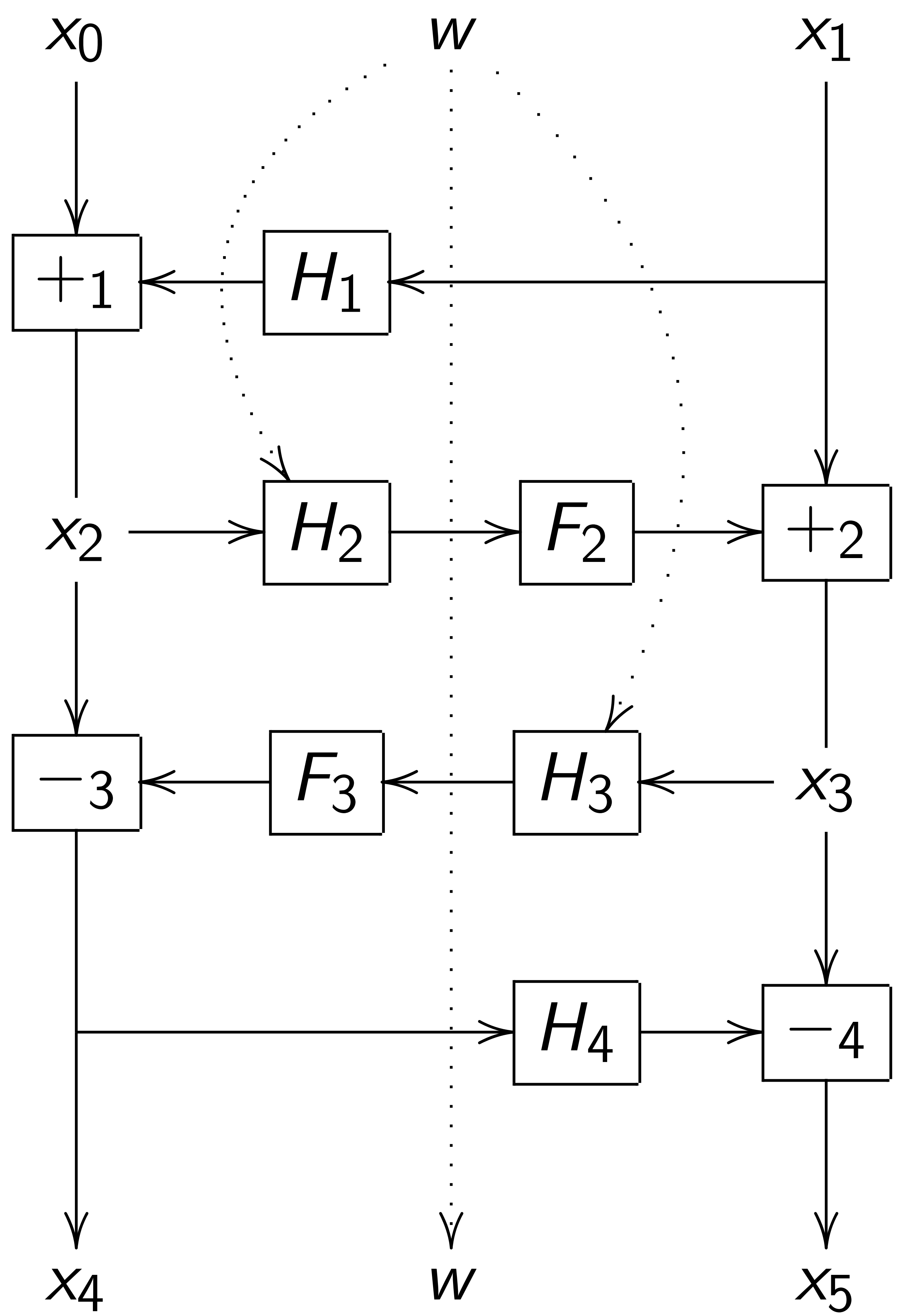


Previous
(Bernste
H is pur
F is a st

Ingredie
H at top
bottom
H_2, H_3 a
H_1, H_4 a
XCB/HC
than 200

Allow o
unify H_1
unify H_3





Previous slide: HH
(Bernstein–Nandi–
H is purely combi
F is a stream ciph

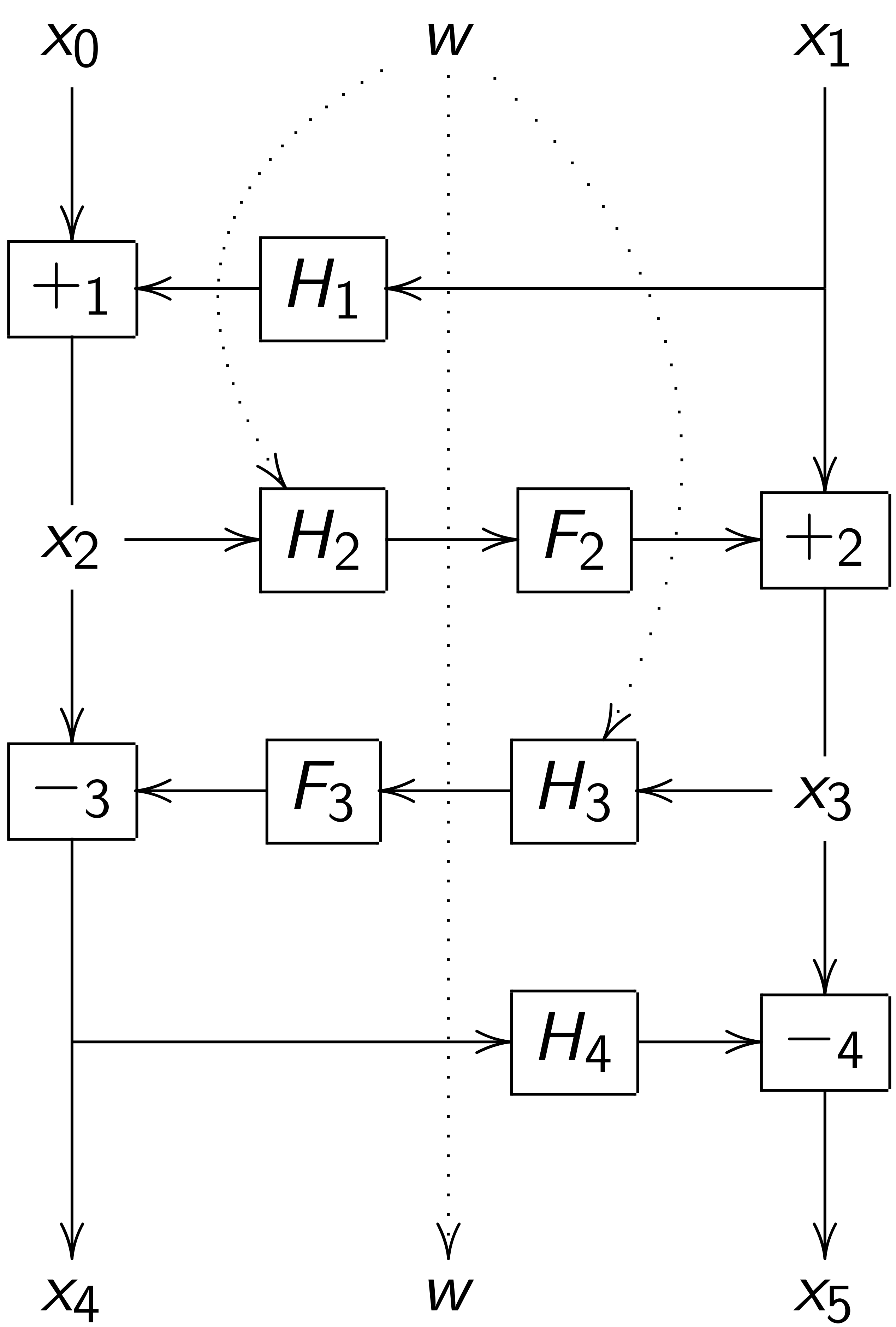
Ingredients: 4-rou
H at top (1996 Lu
bottom (1997 Na
H_2, H_3 allow one-
H_1, H_4 are stretch
XCB/HCTR-style
than 2002 Liskov–

Allow one $H_1, H_2,$
unify H_1, H_2 hypo
unify H_3, H_4 hypo

her
RP)

NR
MC
ME
CB
CTR
EP
CH
ET
EH
HCH
OH
MME

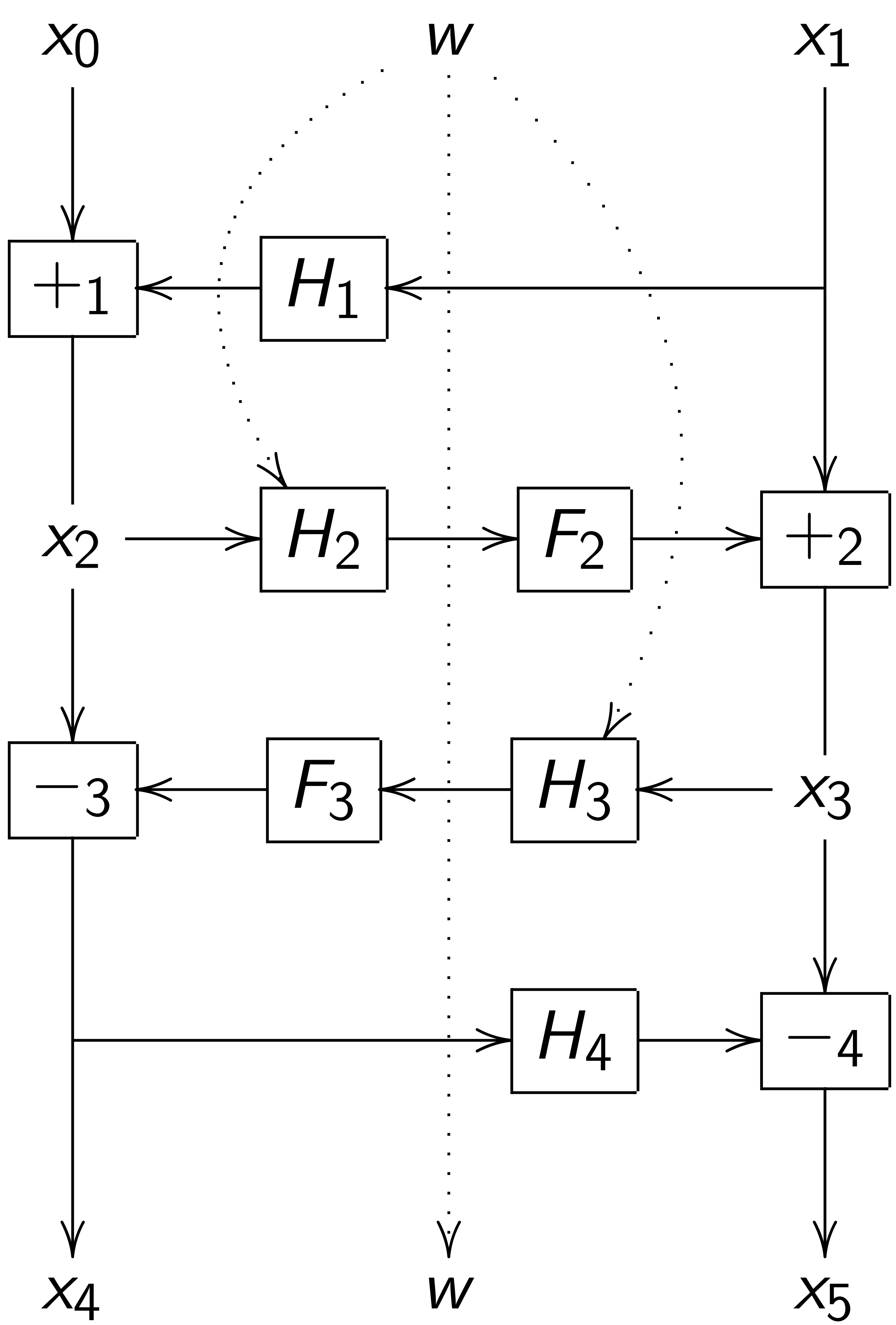
pher
RP)



Previous slide: HHFHFH
(Bernstein–Nandi–Sarkar).
H is purely combinatorial;
F is a stream cipher.

Ingredients: 4-round Feistel;
H at top (1996 Lucks),
bottom (1997 Naor–Reingol
H_2, H_3 allow one-block nonc
H_1, H_4 are stretched by 0-pa
XCB/HCTR-style tweak, fas
than 2002 Liskov–Rivest–Wa

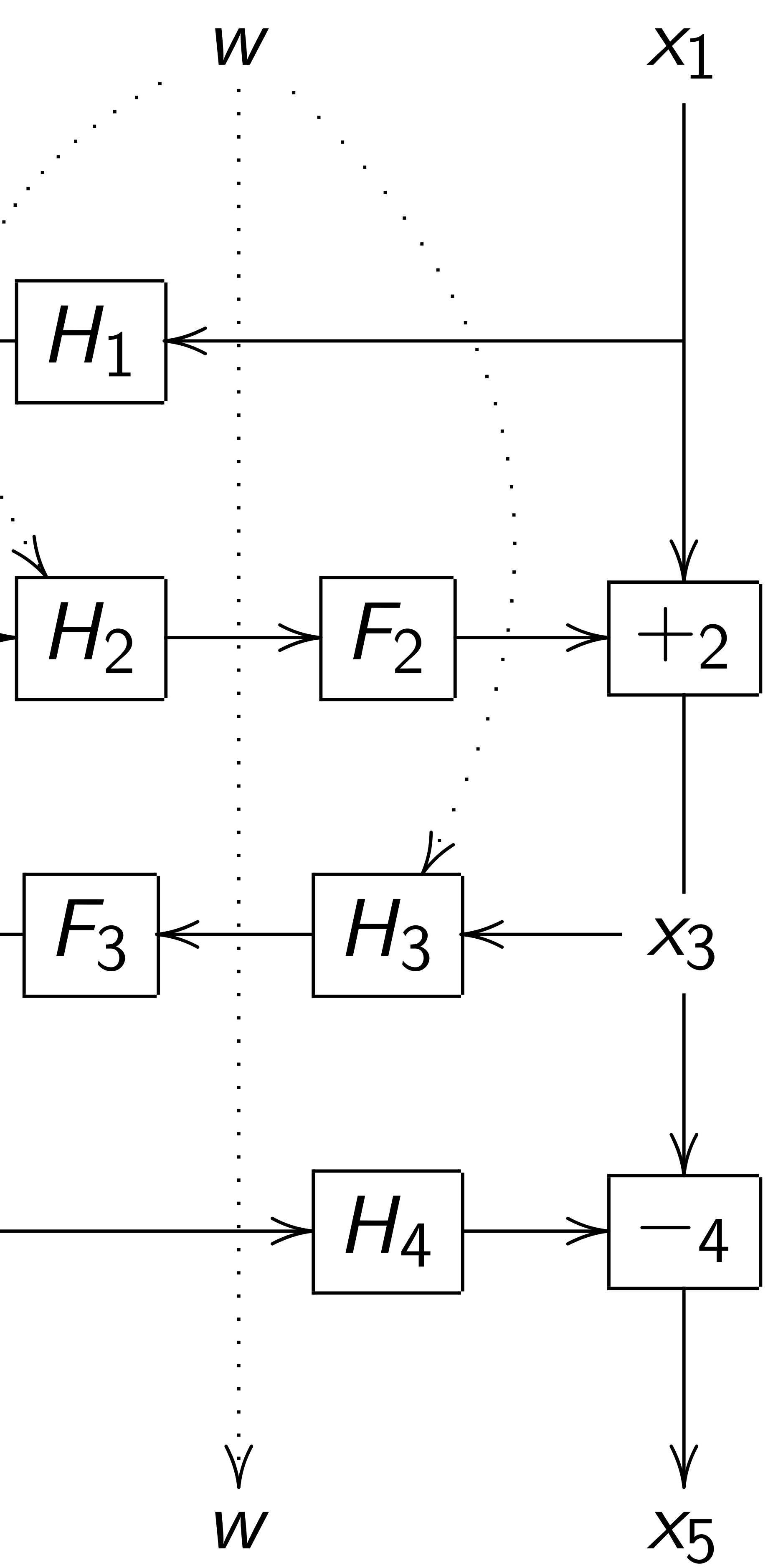
Allow one H_1, H_2, H_3, H_4 ke
unify H_1, H_2 hypotheses;
unify H_3, H_4 hypotheses.



Previous slide: HHFHFH (Bernstein–Nandi–Sarkar).
H is purely combinatorial;
F is a stream cipher.

Ingredients: 4-round Feistel;
H at top (1996 Lucks),
bottom (1997 Naor–Reingold);
H_2, H_3 allow one-block nonces;
H_1, H_4 are stretched by 0-pad;
XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one H_1, H_2, H_3, H_4 key;
unify H_1, H_2 hypotheses;
unify H_3, H_4 hypotheses.



Previous slide: HHFHFH (Bernstein–Nandi–Sarkar). H is purely combinatorial; F is a stream cipher.

Ingredients: 4-round Feistel; H at top (1996 Lucks), bottom (1997 Naor–Reingold); H_2, H_3 allow one-block nonces; H_1, H_4 are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one H_1, H_2, H_3, H_4 key; unify H_1, H_2 hypotheses; unify H_3, H_4 hypotheses.

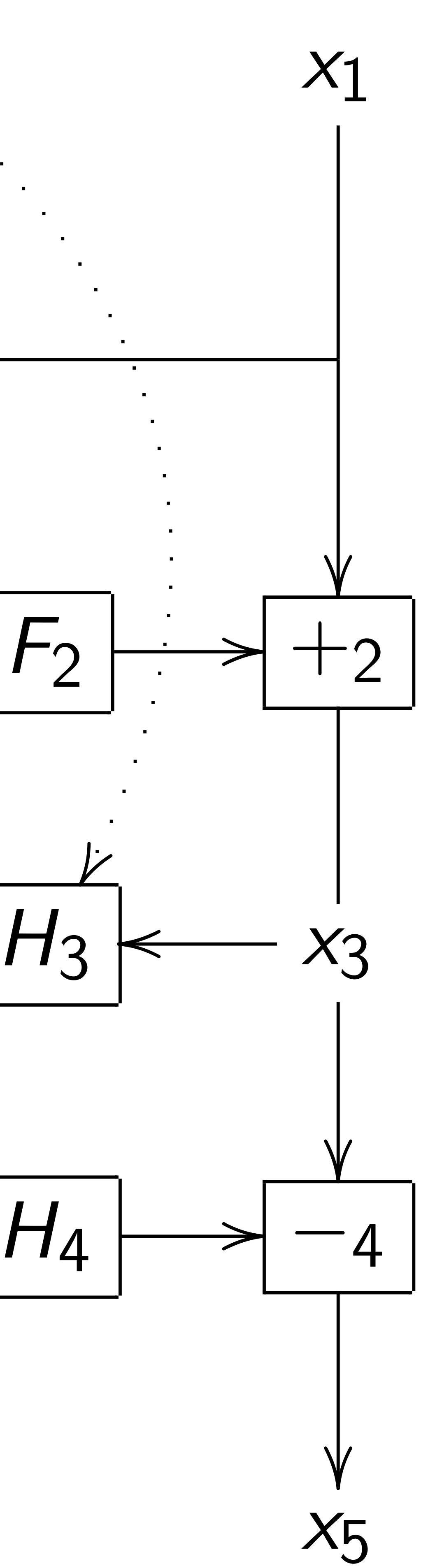
One pos
permuta

Full-widt
beats sq
and CTF

Also cho
We're st

Use sing
"chopT(
w as tru

HHFHFI
twice, w
Somethi
more loc



Previous slide: HHFHFH (Bernstein–Nandi–Sarkar). H is purely combinatorial; F is a stream cipher.

Ingredients: 4-round Feistel; H at top (1996 Lucks), bottom (1997 Naor–Reingold); H_2, H_3 allow one-block nonces; H_1, H_4 are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one H_1, H_2, H_3, H_4 key; unify H_1, H_2 hypotheses; unify H_3, H_4 hypotheses.

One possibility for
permutation in EM

Full-width permuta
beats squeezing fo
and CTR is highly

Also choose highly
We're still optimiz

Use single-block tw
"chopTC": chain
w as truncation of

HHFHFH reads ea
twice, writes each
Something I'm wo
more locality insid

Previous slide: HHFHFH (Bernstein–Nandi–Sarkar). H is purely combinatorial; F is a stream cipher.

Ingredients: 4-round Feistel; H at top (1996 Lucks), bottom (1997 Naor–Reingold); H_2, H_3 allow one-block nonces; H_1, H_4 are stretched by 0-pad; XCB/HCTR-style tweak, faster than 2002 Liskov–Rivest–Wagner.

Allow one H_1, H_2, H_3, H_4 key; unify H_1, H_2 hypotheses; unify H_3, H_4 hypotheses.

One possibility for F: permutation in EM in CTR.

Full-width permutation outp
beats squeezing for long out
and CTR is highly parallel.

Also choose highly parallel H
We're still optimizing choice

Use single-block tweak w.
"chopTC": chain by choosin
w as truncation of $P \oplus C$.

HHFHFH reads each bit in a
twice, writes each bit once.
Something I'm working on n
more locality inside permuta

Previous slide: HHFHFH
(Bernstein–Nandi–Sarkar).
H is purely combinatorial;
F is a stream cipher.

Ingredients: 4-round Feistel;
H at top (1996 Lucks),
bottom (1997 Naor–Reingold);
H_2, H_3 allow one-block nonces;
H_1, H_4 are stretched by 0-pad;
XCB/HCTR-style tweak, faster
than 2002 Liskov–Rivest–Wagner.

Allow one H_1, H_2, H_3, H_4 key;
unify H_1, H_2 hypotheses;
unify H_3, H_4 hypotheses.

One possibility for F:
permutation in EM in CTR.

Full-width permutation output
beats squeezing for long output;
and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing
w as truncation of $P \oplus C$.

HHFHFH reads each bit in array
twice, writes each bit once.
Something I'm working on now:
more locality inside permutation.

s slide: HHFHFH
ein–Nandi–Sarkar).
ely combinatorial;
tream cipher.

nts: 4-round Feistel;
(1996 Lucks),
(1997 Naor–Reingold);
allow one-block nonces;
are stretched by 0-pad;
CTR-style tweak, faster
02 Liskov–Rivest–Wagner.

ne H_1, H_2, H_3, H_4 key;
$_1, H_2$ hypotheses;
$_3, H_4$ hypotheses.

One possibility for F:
permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.
Something I'm working on now: more locality inside permutation.

Security
compare
basically
assumin
and typi

Is this 2

HFHFH
-Sarkar).
natorial;
er.

nd Feistel;
ucks),
or–Reingold);
block nonces;
ed by 0-pad;
tweak, faster
-Rivest–Wagner.

H_3, H_4 key;
theses;
theses.

One possibility for F:
permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.
Something I'm working on now: more locality inside permutation.

Security loss of mo
compared to secur
basically $q^2/2^{128}$,
assuming 128-bit
and typical choice

Is this 2^{128} "secur

d);
ces;
ad;
ster
agner.

ey;

One possibility for F:
permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.
Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

One possibility for F:
permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.
Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

One possibility for F:
permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.
Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

One possibility for F: permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H. We're still optimizing choices.

Use single-block tweak w. "chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once. Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

One possibility for F:
permutation in EM in CTR.

Full-width permutation output beats squeezing for long output; and CTR is highly parallel.

Also choose highly parallel H.
We're still optimizing choices.

Use single-block tweak w.
"chopTC": chain by choosing w as truncation of $P \oplus C$.

HHFHFH reads each bit in array twice, writes each bit once.
Something I'm working on now: more locality inside permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

sibility for F:
tion in EM in CTR.

th permutation output
ueezing for long output;
R is highly parallel.

oose highly parallel H.
till optimizing choices.

gle-block tweak w.
C": chain by choosing
uncation of $P \oplus C$.

H reads each bit in array
rites each bit once.
ng I'm working on now:
cality inside permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

Heavywe

Interesti
≥256 bi
≥256-bi
≥256-bi

F:
M in CTR.

ation output
or long output;
parallel.

parallel H.
ing choices.

weak w.
by choosing
f $P \oplus C$.

ach bit in array
bit once.
rking on now:
e permutation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

Heavyweight ciphe

Interesting **cipher**
$\geq$256 bits for all p
$\geq$256-bit keys, $\geq$2
$\geq$256-bit subkeys,

ut

tput;

H.

es.

ng

array

now:

ation.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

Heavyweight ciphers

Interesting **cipher**-design spa

$\geq$256 bits for all pipes.

$\geq$256-bit keys, $\geq$256-bit out

$\geq$256-bit subkeys, etc.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

Heavyweight ciphers

Interesting **cipher**-design space:
$\geq$256 bits for all pipes.
$\geq$256-bit keys, $\geq$256-bit outputs,
$\geq$256-bit subkeys, etc.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

Heavyweight ciphers

Interesting **cipher**-design space: $\geq$256 bits for all pipes. $\geq$256-bit keys, $\geq$256-bit outputs, $\geq$256-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, ... This needs far more attention, optimization. **Hash** designs are usually overkill.

Security loss of mode compared to security of F: basically $q^2/2^{128}$, assuming 128-bit blocks and typical choice of H.

Is this 2^{128} "security"?

Fragile fix: "beyond-birthday-bound security." Complicates implementation, security analysis.

Simpler fix: "bigger-birthday-bound security." Use 256-bit blocks, security $q^2/2^{256}$.

Is 256-bit n safe in ChaCha?

Heavyweight ciphers

Interesting **cipher**-design space: $\geq$256 bits for all pipes. $\geq$256-bit keys, $\geq$256-bit outputs, $\geq$256-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, ... This needs far more attention, optimization. **Hash** designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!

loss of mode
ed to security of F:
$q^2/2^{128}$,
g 128-bit blocks
cal choice of H.

^{128} "security"?

ix: "beyond-birthday-
ecurity." Complicates
ntation, security analysis.

fix: "bigger-birthday-
ecurity." Use 256-bit
security $q^2/2^{256}$.

it n safe in ChaCha?

Heavyweight ciphers

Interesting **cipher**-design space:
$\geq$256 bits for all pipes.
$\geq$256-bit keys, $\geq$256-bit outputs,
$\geq$256-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, ... This needs far more attention, optimization.
Hash designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!

Another
PRF insi

EdDSA
random
truncate
H is SH

2015 Be
truncate
high-sec

Even wit
reusing
surely ca
in both

ode
rity of F:

blocks
of H.

ity"?

nd-birthday-
Complicates
ecurity analysis.

er-birthday-
Use 256-bit
$^2/2^{256}$.

n ChaCha?

Heavyweight ciphers

Interesting **cipher**-design space:
≥256 bits for all pipes.
≥256-bit keys, ≥256-bit outputs,
≥256-bit subkeys, etc.

Occasional designs: Rijndael,
OMD (SHA-2), Keccak, BLAKE2,
NORX, Simpira, ... This needs
far more attention, optimization.
Hash designs are usually overkill.

Is 256 fundamentally much slower,
or much less energy-efficient,
than 128? My guess: No!

Another optimizat
PRF inside EdDSA

EdDSA generates
random number m
truncated hash: H
H is SHA-512; s i

2015 Bellare–Bern
truncated prefixed
high-security mult

Even with the con
reusing preimage-r
surely can build be
in both software a

y-
es
alysis.

y-
t

?

Heavyweight ciphers

Interesting **cipher**-design space:
$\geq$256 bits for all pipes.
$\geq$256-bit keys, $\geq$256-bit outputs,
$\geq$256-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, ... This needs far more attention, optimization.
Hash designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!

Another optimization target
PRF inside EdDSA signature

EdDSA generates per-signat
random number mod 256-bi
truncated hash: $H(s, m)$ mo
H is SHA-512; s is subkey.

2015 Bellare–Bernstein–Tess
truncated prefixed MD hash
high-security multi-user MA

Even with the constraint of
reusing preimage-resistant h
surely can build better desig
in both software and hardwa

Heavyweight ciphers

Interesting **cipher**-design space: $\geq$256 bits for all pipes. $\geq$256-bit keys, $\geq$256-bit outputs, $\geq$256-bit subkeys, etc.

Occasional designs: Rijndael, OMD (SHA-2), Keccak, BLAKE2, NORX, Simpira, ... This needs far more attention, optimization. **Hash** designs are usually overkill.

Is 256 fundamentally much slower, or much less energy-efficient, than 128? My guess: No!

Another optimization target: PRF inside EdDSA signatures.

EdDSA generates per-signature random number mod 256-bit ℓ as truncated hash: $H(s, m) \bmod \ell$. H is SHA-512; s is subkey.

2015 Bellare–Bernstein–Tessaro: truncated prefixed MD hash is a high-security multi-user MAC.

Even with the constraint of reusing preimage-resistant hash, surely can build better design in both software and hardware.