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 $\mathrm{AES}_k(1) \oplus m_1$,
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 $\text{AES}_k(1)$
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".

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.

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 $\mathrm{AES}_k(1)$
using 2^{64} iterations
on a small quantum processor.

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.

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

Grover finds k from $\mathrm{AES}_k(1)$ 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
$s_n = \mathrm{AES}_k(n)$ 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.

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?)

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”?

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

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.)

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!”

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?

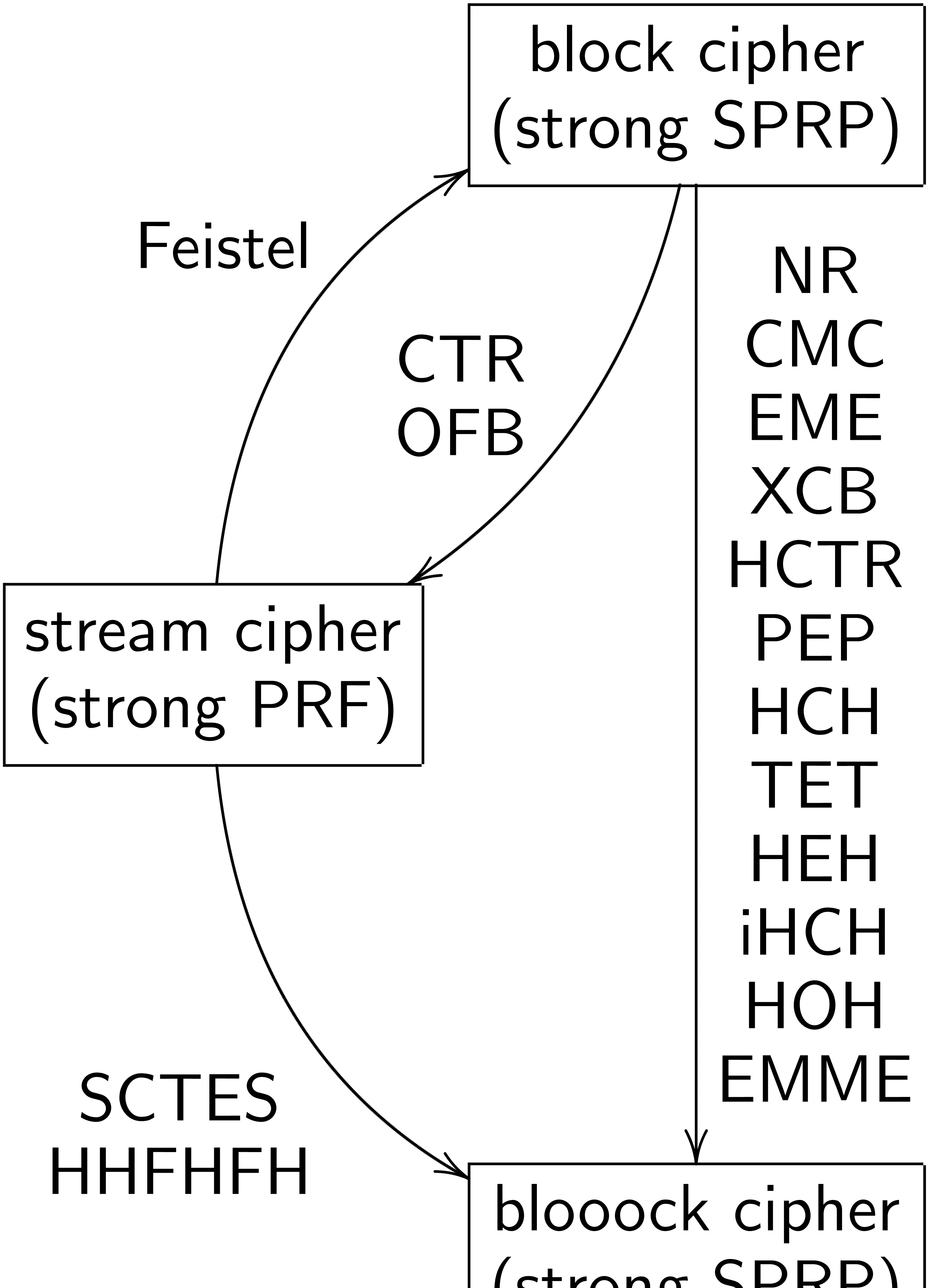
Improving Tor

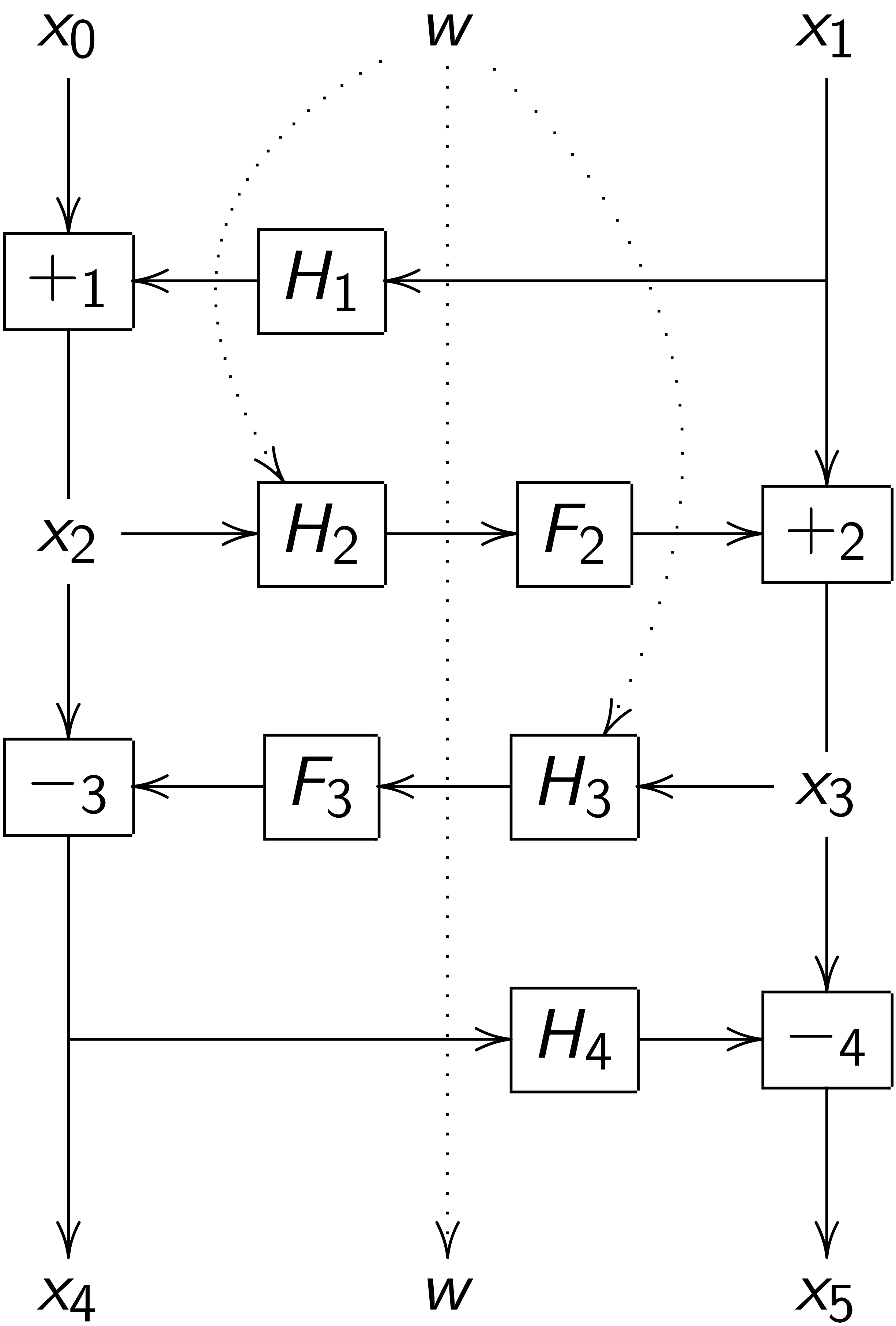
Tor wants "fast, proven, secure, easy-to-implement, non-patent-encumbered, side-channel-free" 509-byte blooock cipher. (But current cipher is a disaster, so can consider compromises.)

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.


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 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.

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"?

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.

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 $q^2/2^{256}$.

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.

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.

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.