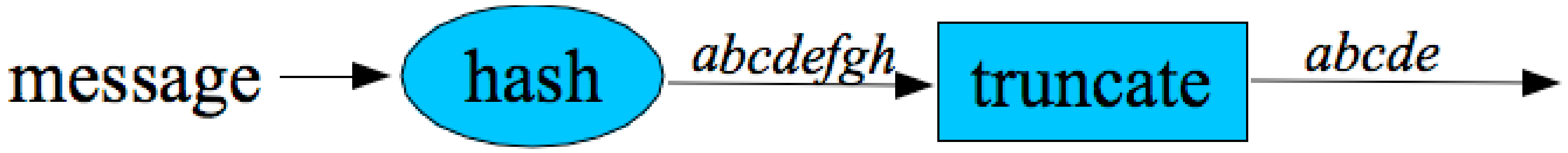
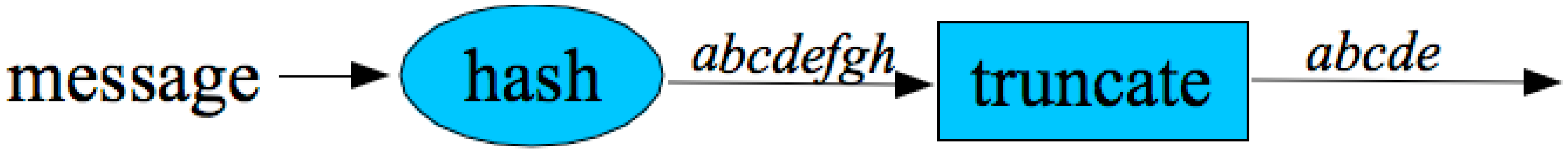
SHA-160: A Truncation Mode for SHA256 (and most other hashes)

John Kelsey, NIST

Halloween Hash Bash 2005



What's a Truncation Mode?



- Rule for chopping bits off a hash output
- We have a big hash fn we trust,

  Like SHA256
- We need a smaller hash output

  Like 160 bits
- We need to specify how this is done
  - Interoperability and security reasons

Why Do We Need One?

- Need drop in replacement for SHA1 (MD5?)
- Have unbroken hashes of wrong size
  - ECDSA/DSA key sizes
  - File and protocol formats
- Obvious approach:
  Truncate SHA256/SHA512
- This has been done before:
  Snefru, Tiger, SHA384, SHA224

Our Proposal in a Nutshell

$H(X,M)$ = hash M from initial value X

- Start with different IV for each truncation length n:
  - *n* has fixed-length representation
  - $IV^T_n = H(IV \text{ xor } 0xccc\ldots c, n)$
- Run bigger hash normally
  - $H^T_n(m) = truncate(\mathbf{H(IV^T_n, m)}, n)$
- Generic: Any n, many big hashes
  - (Rivest comment to SHA224)

Intuition: Why should this be okay?

- If hash “good”, seems like truncation should be good, too.
  - Fits our intuition about hash functions
  - Easy proof in Random Oracle Model
  - Prior art suggests other people agree
- So, is intuition correct here?

Security Considerations

- Issue #1: Related hash outputs
  - $H^T_n(X) \approx H(X')$
- Issue #2: Can we safely truncate?
  - No reduction proof
  - “Near collisions”

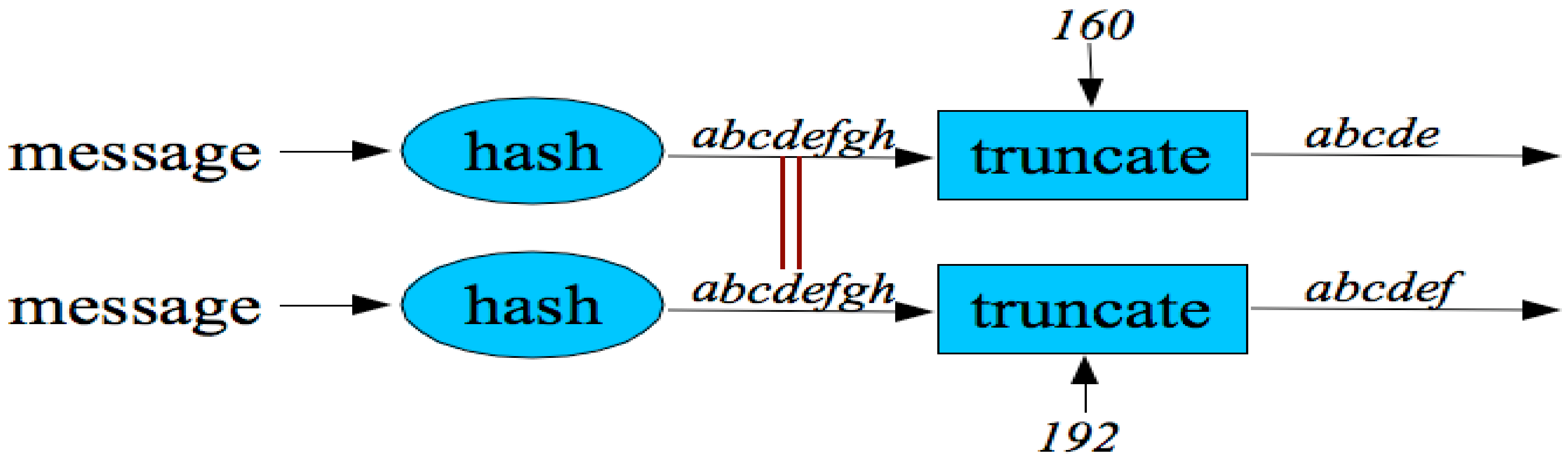
Issue #1: Related Outputs *Why we need $IV^T_n \neq IV$*

What if IV^T_n =IV?

Then we get *collision before truncation*:

$H^T_{160}(M)$ = ABCDE

$H^T_{192}(M)$ = ABCDEF



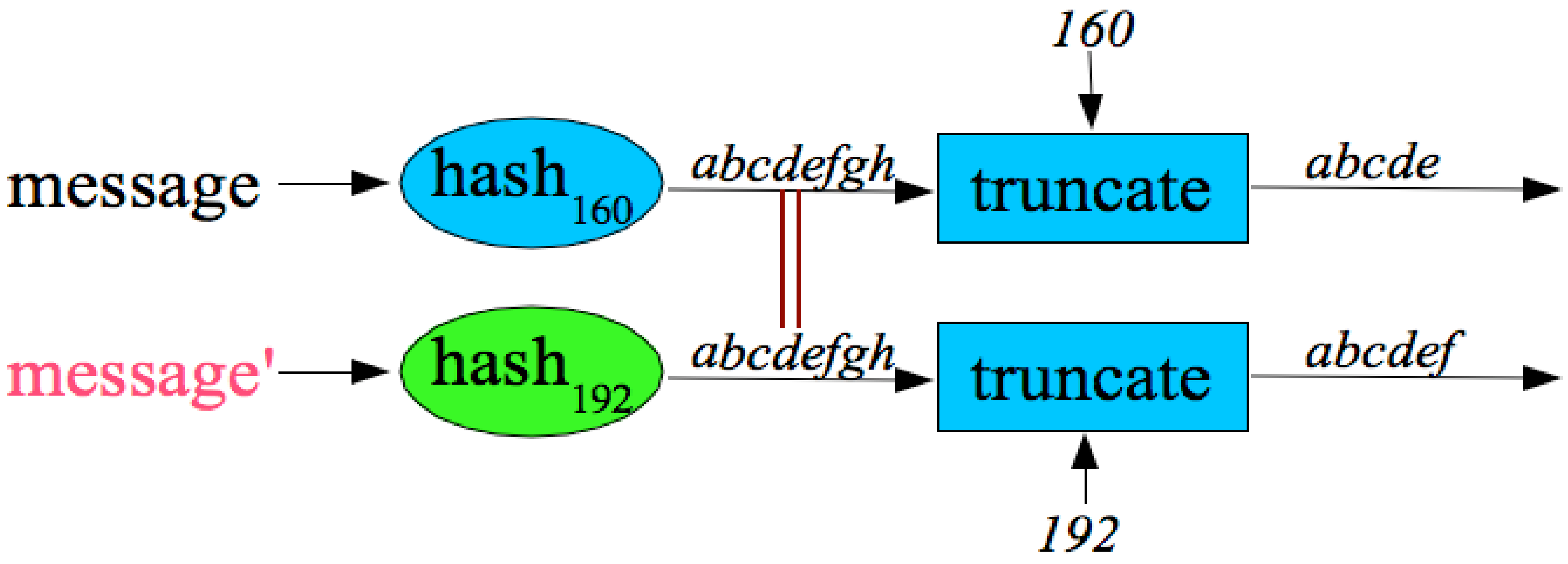
Does This Matter? A Common KDF

- KDF(S,P,n):
  - T = “”
  - for j = 1 to n:
    T = T || hash(S||P||j)
- Two people use different truncations:
  - Result: Two closely related keys
    “AAAABBBBCCCCDDDD”
    “AAABBBCCCDDD”
- Very unintuitive property!
  - Related key attack? Protocol problem?

Broader Issue: Related Outputs

$H^T{}_n(m) = truncate(\mathbf{H(IV^T{}_n, m)},n)$

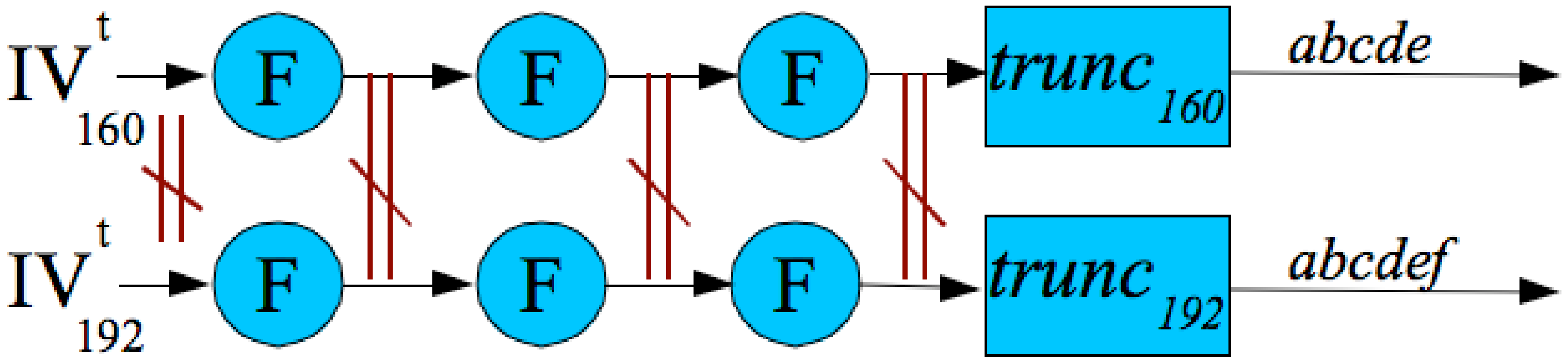
- What if $H^T{}_n(m) \approx H(m')$?
  - Broader property: can't prove it won't happen
  - Narrow property: collision before truncation



Does Our Scheme Prevent This?

We prevent collision before truncation:

- Can’t choose M’ -> IV^T_n (preimage)
- Can’t cause intermediate collision
  (DM constr. proof, collision-resistance)
- Can do DM-like proof here



Issue #2: Can We Safely Truncate?

- No reduction proof
- Example attack techniques to break truncated hashes
  - Biham/Chen: Near Collisions
  - Knudsen: Truncated Differentials
- But it still *looks* okay….

Truncation can make a strong hash weak!

- Let:
  G(x) = 256 bit hash
  F(x) = Truncate(G(x),160).
- Question: Can we reduce
  “G is a good 256-bit hash function”
  to
  “F is a good 160-bit hash function”?
- Answer: NO!
  We can show a counterexample!

How to break F without breaking G

- Suppose can find “near-collisions” for G:

  G(X)=**ABCDE**FGH

  G(Y)=**ABCDE**XYZ

  – ALWAYS collide in first 160 bits

  – NEVER collide in last 96 bits
- We **do** break F(X) = Truncate(G(X),160)
- We **don’t** break G(X)

Result: No reduction from G() to F()

Near Collisions

- Is this just a proof problem?

NO!

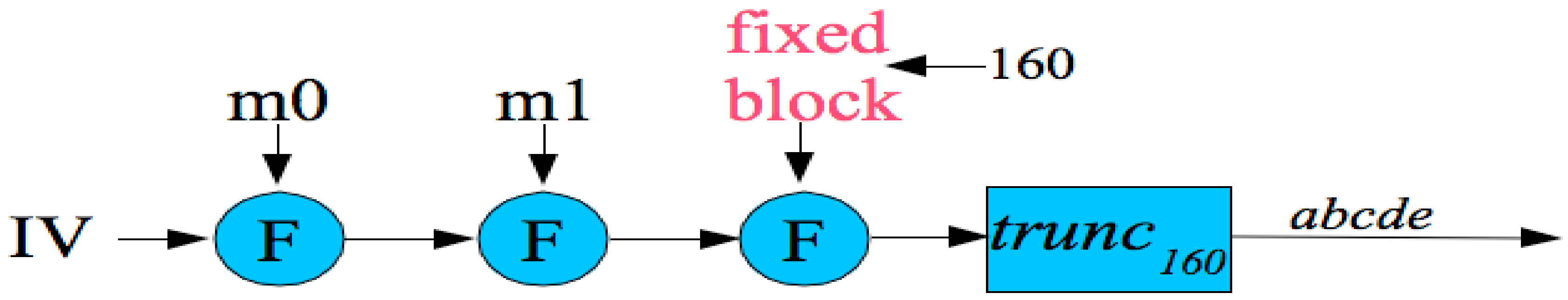
- Biham/Chen Crypto 2004 paper on *near-collisions* for SHA0.
- General Techniques:
  - truncated differentials, impossible diffs.

    abcdefgh --> 00000xyz

But is it okay?

- This is major question: Is there a threat to truncating SHA256?
- If we found near-collision in SHA256, this would call SHA256 into question….
  - No obvious attacks so far
- Note: Truncating to 160 bits, we only care about attacks $< 2^{80}$ work!

Alternative: Block Near Collisions



- We *could* postprocess hash output before truncation
  - Need extra assumption to solve proof prob.
- Gideon Yuval suggestion:
  - Process one or more fixed blocks after end
  - Doesn't solve proof problem
  - *May* block attack

Conclusions

- We have a proposal for a truncated hash mode
- We can’t do a reduction proof to the original hash directly
  - We need extra assumption about strength of truncated output
- We could make changes to address threat of near-collisions
- We need feedback: Will SHA256 support this?

Truncation Mode for SHA

John Kelsey
National Institute of Standards and Technology
john.kelsey(at)nist.gov

ABSTRACT

Recent cryptanalytic results have made the use of SHA1 potentially dangerous. The obvious near-term strategy is a move to SHA256. However, there are many cryptosystems (such as DSA and ECDSA) which cannot handle changes in hash function output, and in many, it is hard to justify rolling out new key sizes to deal with a cryptanalyzed hash function. For these reasons, we propose a general truncation mode for X9-approved hash functions built (broadly) on the Damgard-Merkle construction. The mode specifies a different initial chaining value for each choice of hash function to be truncated and truncation length, generalizing the approach taken for SHA384 and SHA224.