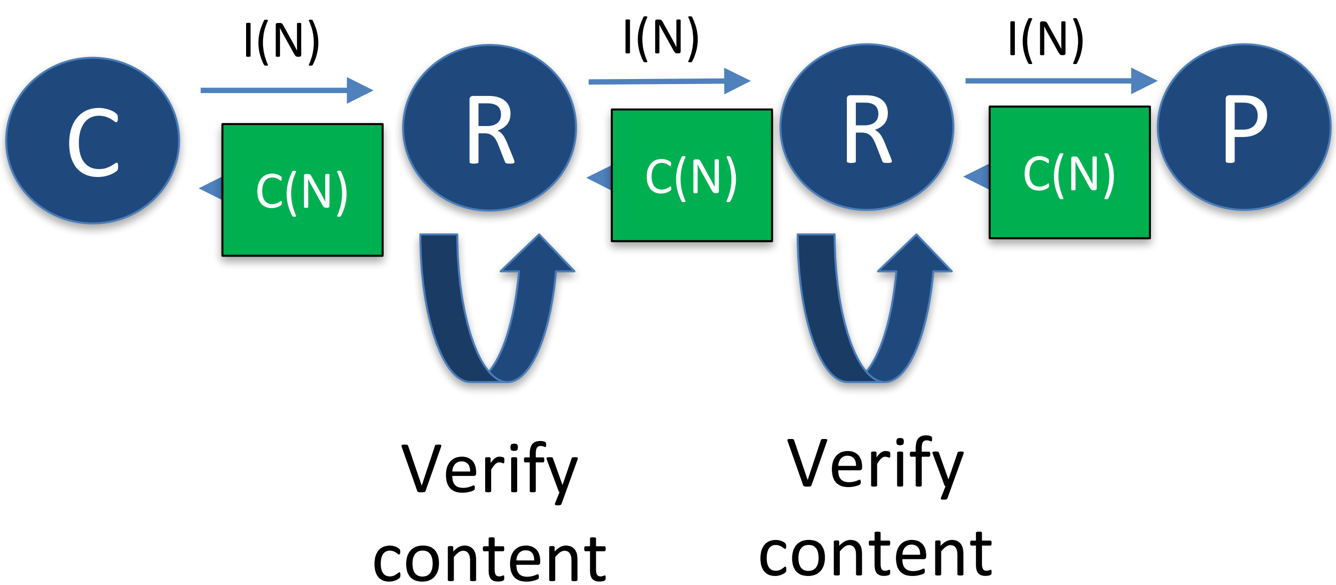
Mitigating On-Path Adversaries in Content-Centric Networks

Cesar Ghali, Gene Tsudik and Christopher Wood
University of California, Irvine
{cghali, gene.tsudik, woodc1}@uci.edu

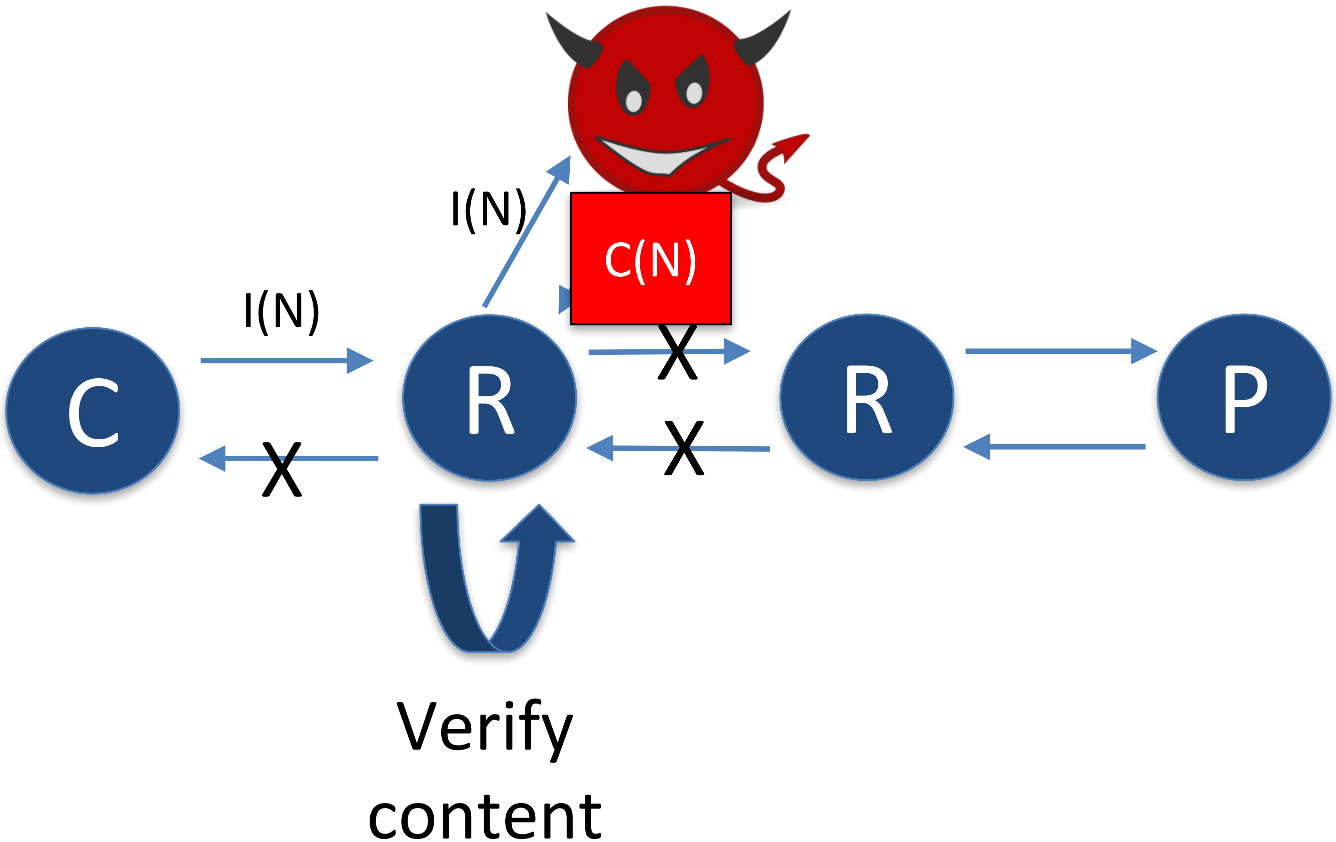
Outline

- Content poisoning
- On-path attack variations
- Adversary leap frog and fast path integrity checks
- Experimental analysis
- Conclusion and future work

Content Poisoning



Content Poisoning



Content Verification

Two mechanisms to verify content authenticity:

1. Digital signature
2. Content hash

What keys and hashes are trusted?

Verification Restrictions

- **KeyID**: hash of public verification key
  - Trusted public key obtained out-of-band
- **ContentID**: hash of content
  - Trusted hash obtained via manifest

On-path attacks are only applicable to interests without ContentIDs

Content Processing

1. Lookup matching PIT entry
2. Forward content to downstream interface(s)
3. Attempt to verify content and, if valid, insert into cache

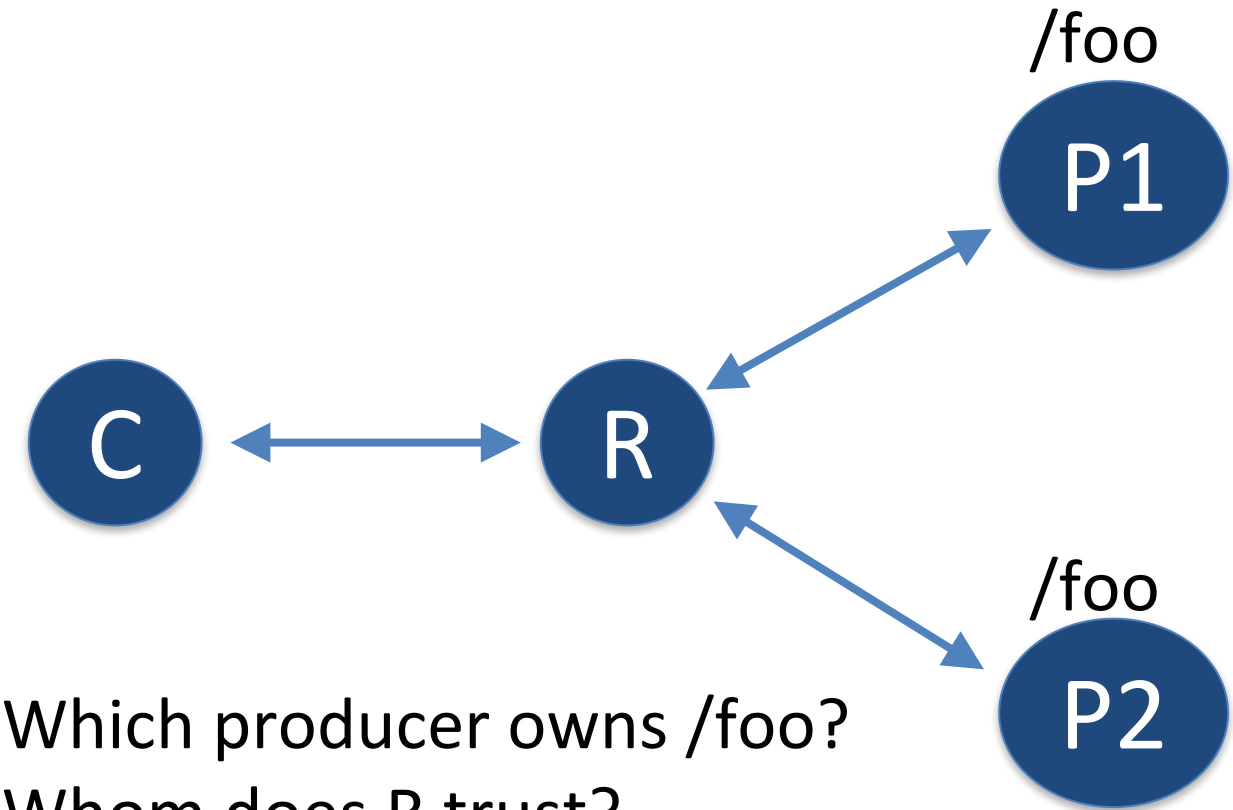
Content **must** be forwarded before verified. Otherwise, content is blocked at each hop.

On-Path Attacks

Without mandatory verification before forwarding, how do we prevent or deter on-path attackers?

First: reduce the problem to inline integrity checks.

Namespace Conflicts

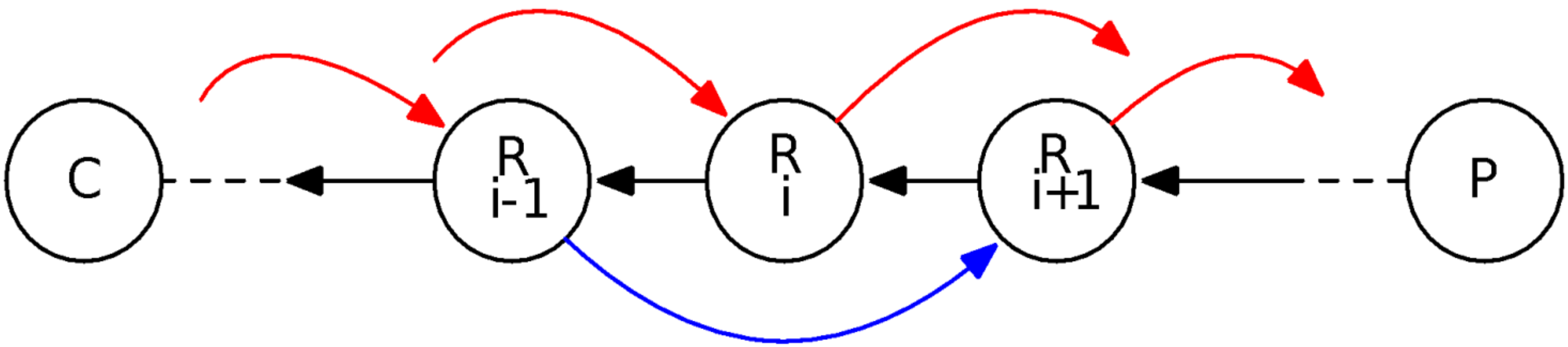


1) Which producer owns /foo?
2) Whom does R trust?

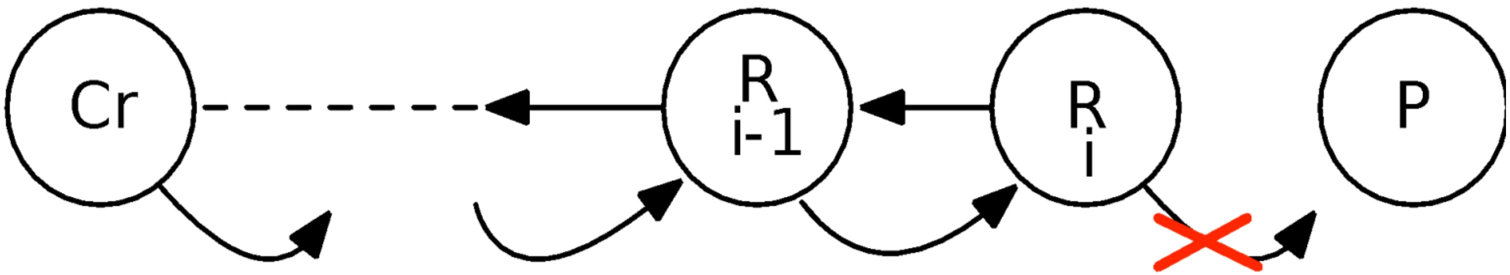
Namespace Arbitration

- There must exist an entity that manages namespace ownership.
- Routers must be able to verify ownership of namespaces according to this arbiter.

Modification Attack



Generation Attack

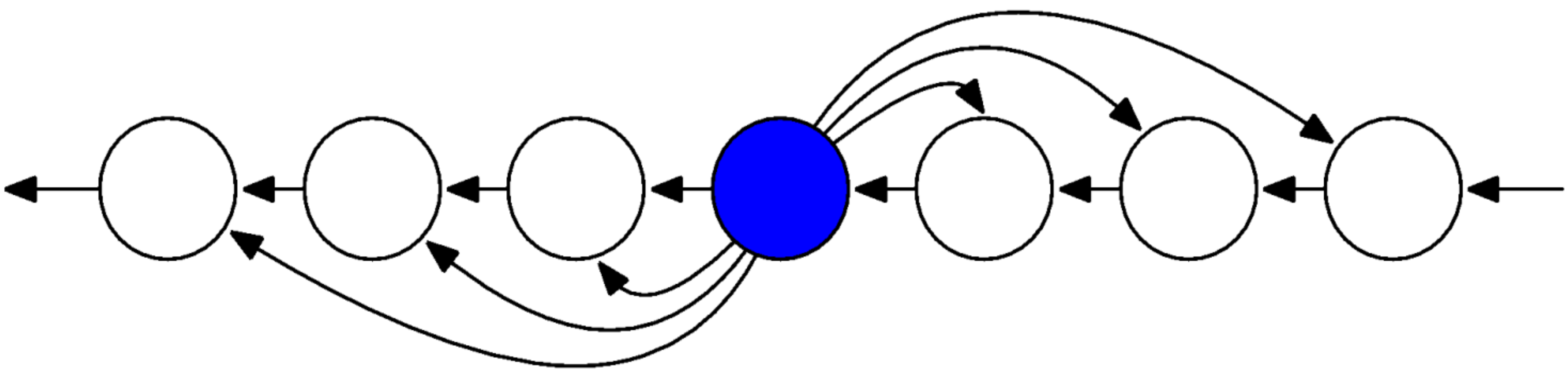


Integrity Checks

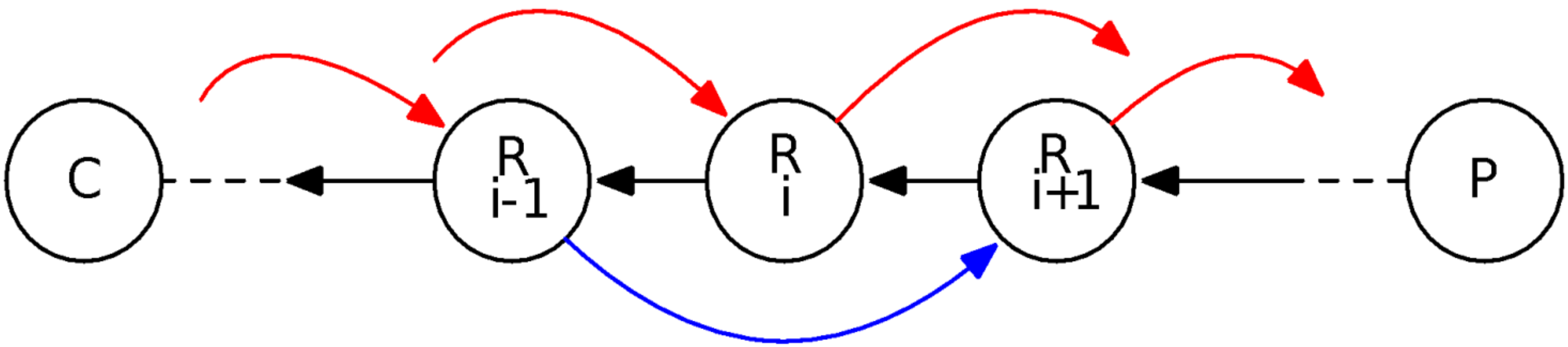
- Problem: signatures are too expensive
- Approach: use MACs

Fast Integrity Checks

- Routers share pairwise k^2 keys with k^2 neighboring routers k > 1 hops away



Adversary Leap Frog



MAC Generation

- Upon interest:
  - Append local router ID.
  - Forward as normal.
- Upon content:
  - Verify k upstream MACs. Drop and avoid immediate upstream router if invalid.
  - Append k downstream MACs using keys shared with downstream IDs.

MAC Compression

- Packets carry O(k^2) MACs
- Failure of a *single* MAC means the immediate upstream router is malicious
- Compress k individual MACs into one MAC via XOR

Packet Headers

- Without compression, header contains:
  - –MACs to check validity of of previous *k* hops
  - –MACs for i-th downstream router to check validity of *k*-i hops
  - –Total <= k(k + 1)/2
- With compression, header contains:
  - –List of *k* upstream routers IDs
  - –List of *k* aggregate MACs

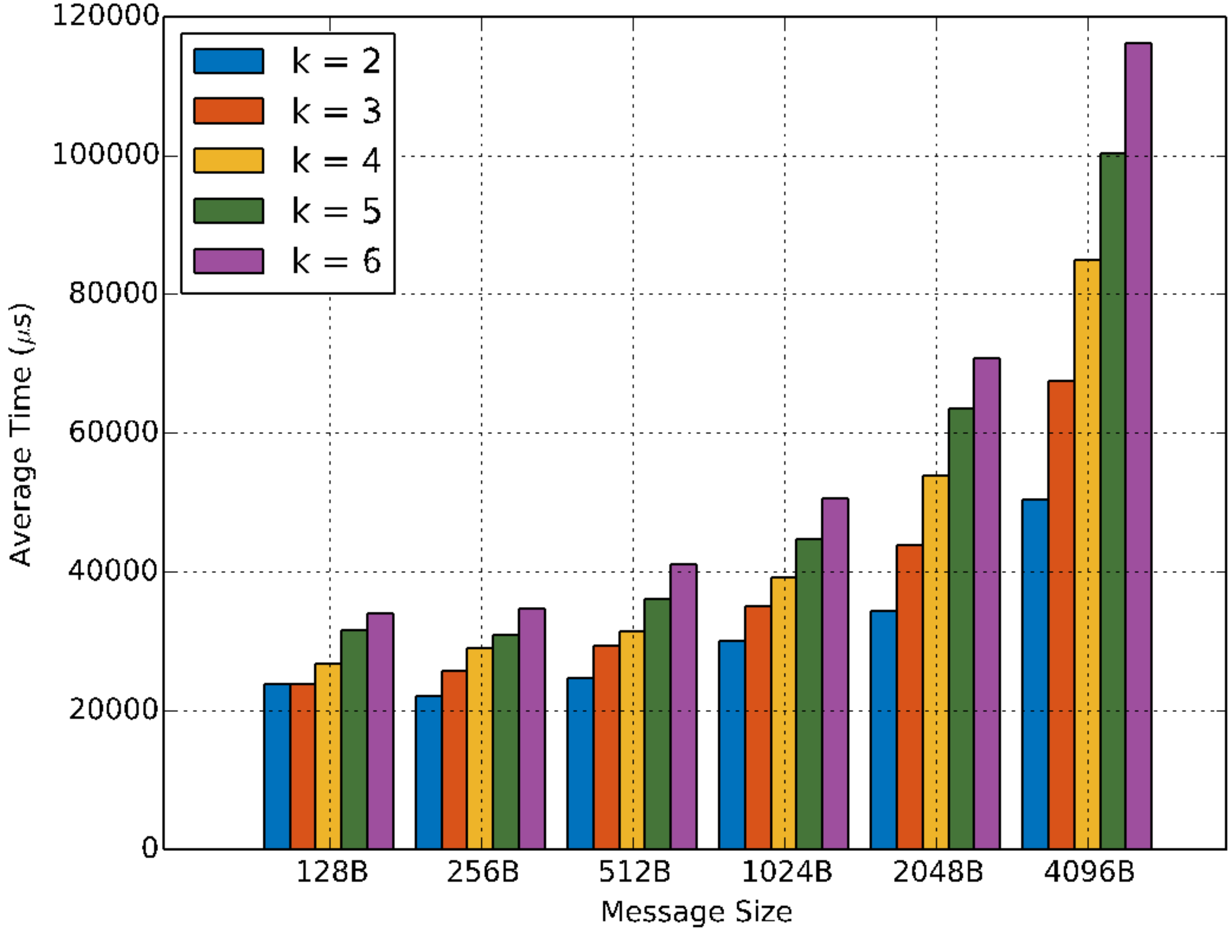
Experimental Analysis

- Assess overhead of MACs operations
- Max of 2*k* operations:
  - *k* MACs verification
  - *k* MACs generation
- Network topology has no impact on per-packet overhead

Choice of MAC

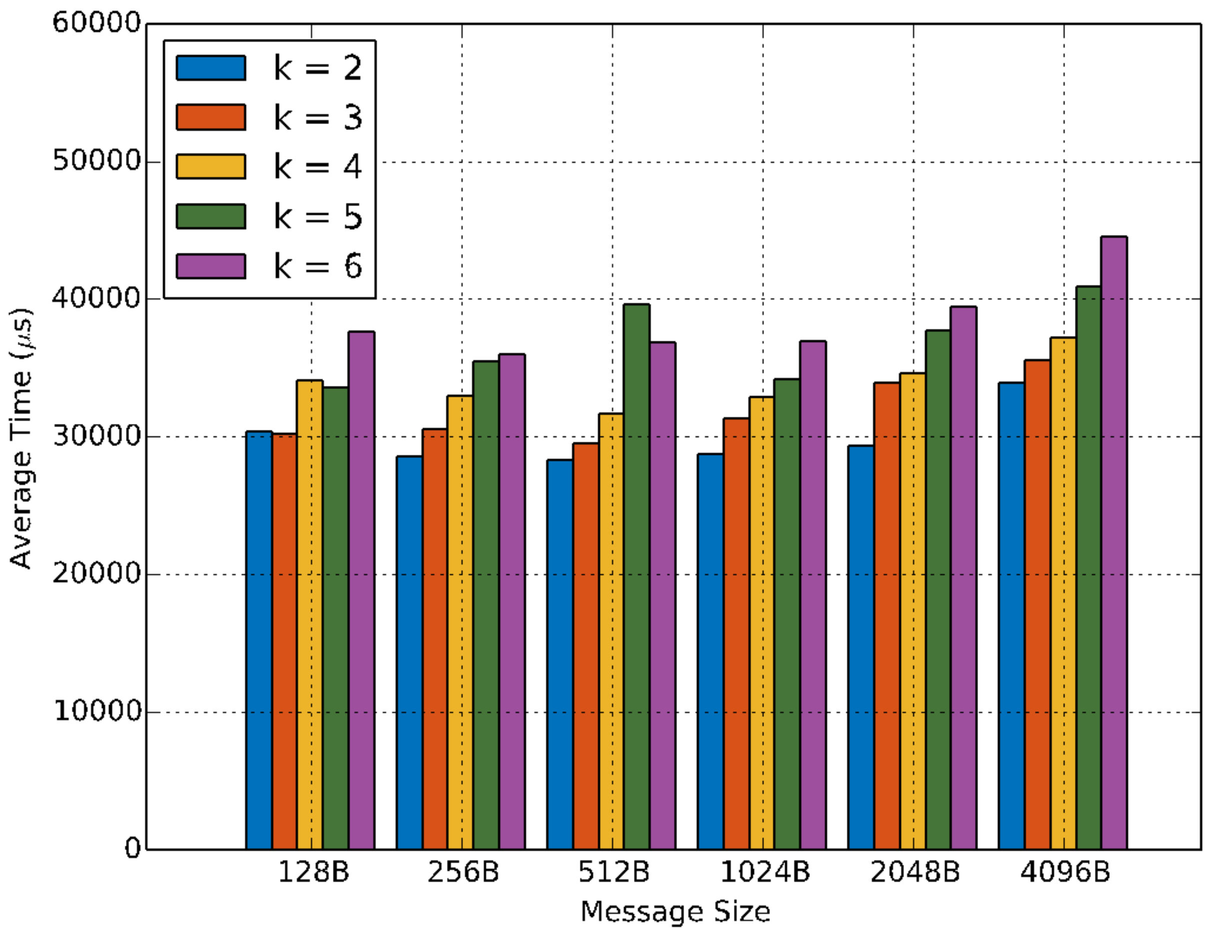
- Many variations
  - HMAC: Hash-based MAC
  - CMAC: Block-cipher-based MAC
  - PMAC: Parallel block-cipher-based MAC
- We chose HMAC given its widespread use in CCNx
- CMAC or PMAC would be more efficient given native CPU support

HMAC



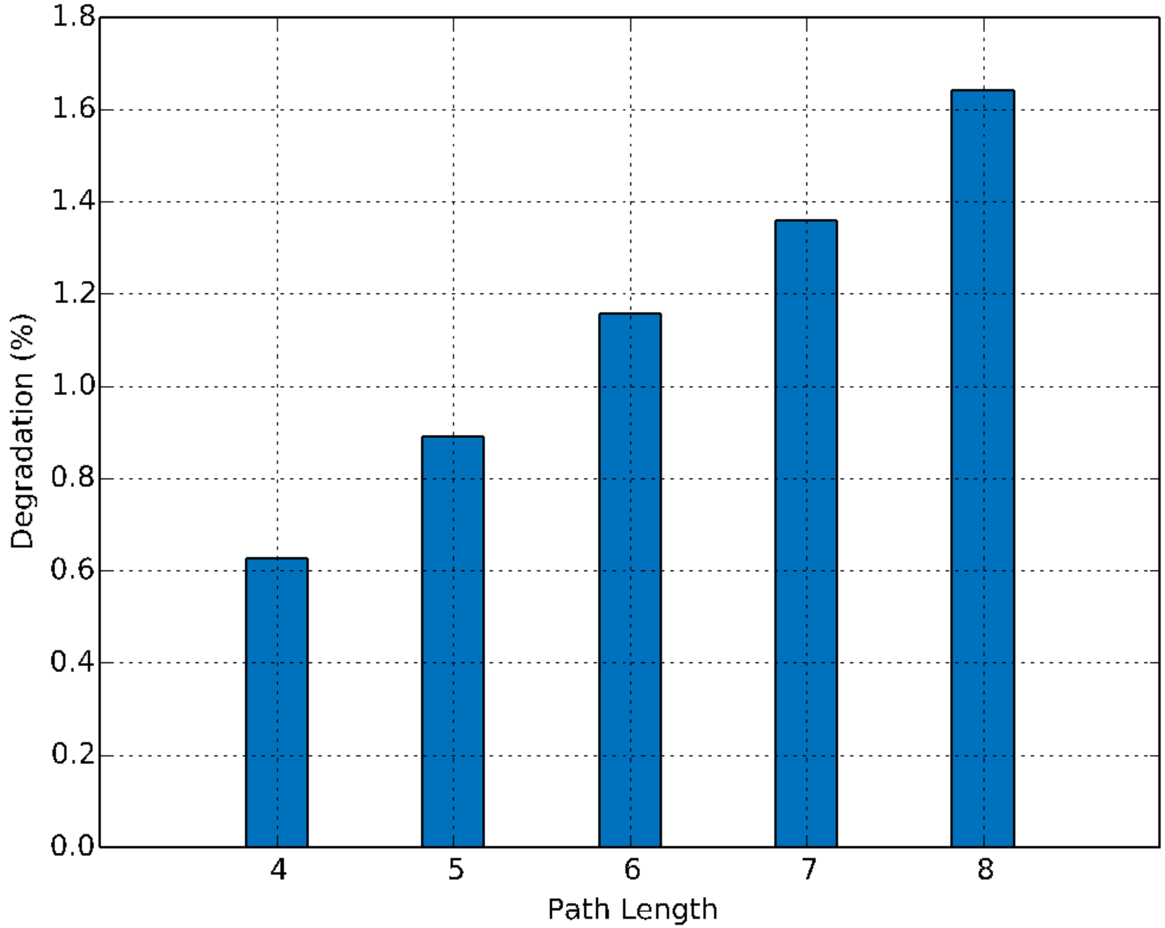
k = 2
k = 3
k = 4
k = 5
k = 6
120000
100000
80000
60000
40000
20000
0
Average Time (μs)
128B
256B
512B
1024B
2048B
4096B
Message Size

Hashed HMAC



60000
50000
40000
30000
20000
10000
0
Average Time (μs)
k = 2
k = 3
k = 4
k = 5
k = 6
128B
256B
512B
1024B
2048B
4096B
Message Size

End-to-End Latency[1]



(1) https://github.com/chris-wood/ccn-onpath-simulation-ccnsim

Scalability and Privacy

- Integrity zones do not scale well at the level of individual routers – work at the AS level
- Integrity zones cost in terms of privacy since path visibility is exposed

Conclusion

- Reduce on-path attacks to inline integrity zone checks
- Use pairwise MACs and adversary leap frog to detect modification and generation attacks

Future Work

- Design key distribution mechanism
- Analyze offline performance costs

/this/is/the/end/version=0x00/chunk=0x01/PID=0x02