Protecting the Long Tail
Transparent Packet Security in Content-Centric Networks

Christopher A. Wood
Department of Computer Science
University of California Irvine
woodc1@uci.edu

IFIP Networking 2017 — June 12, 2017
Agenda

• CCN overview
• Privacy parity and transparent encryption
• TRAPS design & features
• Experimental analysis
• Conclusion
CCN Highlights

• Architecture for transferring named data from producer to consumer upon request
• Names are cryptographically bound to data
• Requests (interests) are routed based on names rather than endpoint addresses
• Content can be opportunistically cached in the network
CCN Overview

$Cr_A$ $R_1$ $R_2$ $R_3$ $P$

$Cr_B$ $R_4$

/youtube/videos/presidentspeech
CCN Overview

Diagram showing nodes and connections:
- $Cr_A$
- $R_1$
- $R_2$
- $R_3$
- $P$
- $Cr_B$

Connections:
- Interest from $Cr_A$ to $R_1$
- $R_1$ to $R_2$
- $R_2$ to $R_3$
- $R_4$ to $R_2$
- $Cr_B$ to $R_4$

Additional node:
- "/youtube/videos/presidentspeech"
CCN Overview

\[ Cr_A \quad R_1 \quad R_2 \quad R_3 \quad P \]

\[ Cr_B \quad R_4 \]

Interest

/youtube/videos/presidentspeech
CCN Overview

$Cr_A$ -> $R_1$ -> $R_2$ -> $R_3$ -> $P$

Interest

$Cr_B$ -> $R_4$

/youtube/videos/presidentspeech
CCN Overview

$Cr_A$ $R_1$ $R_2$ $R_3$ $P$

$Cr_B$

/youtube/videos/presidentspeech
CCN Overview

$C r_A$

$R_1$

$R_2$

$R_3$

$P$

$C r_B$

$R_4$

/youtube/videos/presidentspeech
CCN Overview

$Cr_A$

$R_1$

$R_2$

$R_3$

$P$

$Cr_B$

/youtube/videos/presidentspeech
CCN Overview

$Cr_A$ $R_1$ $R_2$ $R_3$ $P$

$Cr_B$

Interest

/youtube/videos/presidentsspeech
Manifests

root FLIC node (named)
inner FLIC nodes (nameless)
data leaves
data

/a/b/c
Onto Privacy
IP Privacy

Turns this...

GET /a/b/c

RESPONSE: <data>
IP Privacy

Into this... (with IPSec or TLS)

What’s revealed?
• Source and destination addresses and port #
• Timing
• Packet sizes
CCN Privacy

Turns this...

C

Interest: /a/b/c

P

Content: <data>
CCN Privacy

Into this...

What’s revealed?
• Consumer and producer locations
• Timing
• Packet sizes
• Interest name
• Producer identity
• ...

Properties of the (application) data

encrypted content?
Privacy Parity?

CCN

- Application names are necessarily expressed in the clear
- Content is unencrypted unless consumer and producer agree on encryption mechanism and key(s)

IP

- Sessions are encrypted and all traffic is encapsulated
- Data is encrypted at the transport layer, beneath the application
Transparent Encryption Goals

Add “transport” privacy to CCN that is:

1. Application agnostic
2. Sound by default with tunable parameters
3. Free from key exchange protocols
4. Compatible with the CCN request-response pattern
Consumers and producers share one piece of knowledge: names
Consumers and producers share one piece of knowledge: names

Use names as a shared “secret”
Key Ingredients

1. Application to network name translation
2. Name-based content encryption
3. Hash-based content encryption
(1) Application to Network Names*

/foo/bar/baz

(1) Application to Network Names*

/\texttt{foo/bar/baz}\n
\text{Name translation function } F()\n
/F(\texttt{foo})/F(\texttt{foo/bar})/F(\texttt{foo/bar/baz})

(2) Name-Based Content Encryption

\[ r \leftarrow \{0, 1\}^\lambda \]

\[ k = \text{KDF}(N| r) \]

\[ D'(\bar{N}) = \text{Enc}_k(D(N)) \]
Named Content

\[ \bar{N} = F(N, s) \]

\[ N = \frac{a}{b/c} \]

\[ k = KDF(N || r) \]

\[ s_G(\text{now}()) \]

\[ D(N) \]

\[ \bar{N} = C(\bar{N}) \]

\[ \text{Enc}_k(D(N)) \]
TRAPS Flow

\[ N = /\text{foo}/\text{bar}/\text{baz} \]
\[ N = H(\text{/foo}) / H(\text{/foo}/\text{bar}) / H(\text{/foo}/\text{bar}/\text{baz}) \]
(3) Hash-Based Encryption*

\[ k = \text{KeyGen}(D(N)) \]
\[ D'(\bar{N}) = \text{Enc}_k(D(N)) \]

Static (Hash-Based) Content

\[ N = /a/b/c \]

\[ D(N) \]

\[ k = \text{KeyGen}(D(N)) \]

\[ f = \text{Enc}_k(D_i(N)) \]

\[ I_{D_{C_i(N)}} = H() \]

\[ \text{Manifest}(\bar{N}) \]

\[ D(N) \]
Hash (Key) Discovery

• TRAPS applies to all content — manifests included
• Use manifest to carry hash $D(N)$
• Use CCNxKE (presented later today) to transfer manifests to consumers
  • Content hashes are encrypted in transit
Limitations

- Does not work for **dynamically generated names** — data may be dynamic
- Knowledge of the **full** application name allows one to decrypt the content
  - For hash-based content, one must **also know the content hash**
Security Model

Goal is not TLS-grade security
Security Model

Goal is not TLS-grade security

• Knowledge of N means one can decrypt the content
  • Dictionary attacks are feasible for popular names
• Content in the “long tail” popularity distribution is less prone to attack
Dictionary Attack Hardening

- Add time into the network name generation
- Add a producer-generated salt into the network name generation
- Use memory-hard functions
Analysis

Measure the four steps of the protocol:

1. Name obfuscation
2. Name de-obfuscation
3. Content encryption
4. Content decryption

Use select hash functions: SHA2 and Argon2
Overhead (SHA256)

![Bar chart showing overhead times for different segments and steps.]
Overhead (Argon2)
Throughput Bounds

\[
\frac{PT_{obf}}{L} < 1
\]
Throughput Bounds

Packet bytes per second [B/s]  Worst-case processing time [s]

\[ \frac{PT_{\text{obf}}}{L} < 1 \]

MTU [B]
Maximal Throughput

The diagram illustrates the throughput capacity for different packet sizes and algorithms. The x-axis represents the packet size in bytes, ranging from 1500 to 9000. The y-axis represents the throughput capacity in packets per second, ranging from $10^4$ to $10^9$.

- **ARGON2(4-25)**: Red bars
- **SHA256**: Blue bars
- **ARGON2(4-21)**: Green bars

The graph shows that as packet size increases, the throughput capacity generally increases for all algorithms. However, the specific performance varies depending on the algorithm used.
Conclusion

• TRAPS brings opportunistic, application-agnostic encryption to CCN
• TRAPS does not offer the same security as TLS — a powerful enough attacker can break it
  • Protecting unpopular content or content with an unpredictable name (or hash!) is the goal
• Performance assessment shows TRAPS add little overhead in the fast path
• TRAPS obfuscation places a limit on the maximal consumer throughput
Questions?

Fire away!