Handling Trust Enforcement

Christopher Wood, PARC
Marc Mosko, PARC
Ersin Uzun, PARC
Agenda

1. Quick refresher on security in CCN
2. Authenticity and trust models
3. Transport-layer enforcement
4. Q&A
Security in CCN

CCN is all about transferring named data

All data is named, secured (to some degree), and then transferred

Communication security is defined with respect to either (both) of the following items:

**Authenticity**: ensuring and trusting the legitimacy of content

**Confidentiality**: controlling who has access to the data
Authenticity and Trust

Why is authenticity important?

- Ensuring consumers get legitimately correct content!
- Prevent network-layer attacks, e.g., *Content (Cache) Poisoning*
Content Poisoning Attacks

A scenario where a (malicious) entity injects fake content into the cache which serves as:

- Invalid responses to consumer interests
- A means to flush legitimate data from the content stores
Authenticity in the Network Layer

**Question:** How do we protect against attacks at the network layer without knowing what content to trust?
Authenticity in the Network Layer

**Question:** How to protect against attacks at the network layer without knowing what content to trust?

**Answer [1]:** Follow the *(Modified) Interest Key Binding* rule:

> An Interest message must either reflect the public key of the producer or must be a self-certifying entity.

Implication: include the **hash** of desired content or the **identity of the public verification key**

Authentication in the Application Layer

**Question:** How do we determine what content is authentic and trusted?
Question: How to determine what content is authentic and trusted?

Consider the following simple application-layer trust model:

(A) lci:/netflix/key [signed by trusted CA]

(B) lci:/netflix/tv/key [signed by (A)]

(C) lci:/netflix/tv/TheOffice [signed by (B)]

(D) lci:/netflix/tv/ParksAndRecreation [signed by (B)]

Perform signature verification, certificate chain traversal, and check key names against the policy.
Fundamental Trust Model Templates

**Question:** Can we autonomously support any arbitrary trust model?

1. Hierarchical PKI-based model
2. Decentralized and distributed (web of trust)
3. "Flat" and inflexible model
**Goal:** Enable application-specific trust model instantiation with fundamental trust enforcement machinery at the network layer
Bridging the Gap (Cont’d)

Application Layer (Trust Models)

Verification Stack Component

Network Layer (Enforcement Mechanics)
The **Verifier** component is the heart of the trust enforcement machinery responsible for:

1. Verifying ingress content objects
2. Autonomously resolving security information (e.g., certificates)
3. Accepting whitelist and blacklist trust sources
4. …
Internals: Trust Circuit

Upper Stack Component
InputQueueUp

Verifier
Component
InputQueueUp

Lower Stack Component
InputQueueUp

Upper Stack Component

Trust Circuit
Key Namespace Gate
defer
accept
reject
Trust Quantification Gate
defer
accept
reject
Tail

reject

Control Plane Notification
Consider a trust model where **certificates bind keys to name prefixes**

- A content object with name \texttt{lci:/parc/cs1/file} has a verification key with the name \texttt{lci:/xerox/parc/key}

- The content object references the certificate for \texttt{lci:/xerox/parc/key}

- The certificate for \texttt{lci:/xerox/parc/key} says it can sign content objects with the prefix \texttt{lci:/parc/}

![Diagram showing the relationship between CO and CERT with name and cert-name labels.]
**Enforcement**: create a single trust circuit check that checks the certificate-specified prefix against the content object name:

```java
if (contentPrefixMatchesKeyNamespace(content, keySpace)) {
    // process the packet
}
```
Schematized Binding Trust Model

What if the certificate specified more than just a name prefix?

What if the certificate specified a schema from which content names are derived?

Content: \texttt{lci:/parc/users/cwood/presentations/trust}

Key: \texttt{lci:/parc/csl/nds/publickey}

Root key: \texttt{lci:/parc/key}

Schema: \texttt{[lci:/parc] [/] [users] [/] [*] [/] [presentations | code | ...]}
Producer Implications

Producers sign content according to the expected trust model.

Hierarchical and flat model: use keys issued by trusted CAs or their own trusted parties.

e.g., Mozilla signs Browser plugin content objects with Mozilla’s key, which is installed in every user’s version of Firefox.

Web of trust: use their own keys (since they are the "introducer")
Consumer Implications

Consumers verify content according to their specified trust models.

The Verifier configuration determines what trust model is to be enforced:

**Flat**: instantiate the Verifier component with trusted (whitelisted) roots and disable certificate chain traversal

**Hierarchical**: instantiate the Verifier component with trusted (whitelisted) roots and enable certificate chain traversal

...
Questions?...
Thank you