Trust in Information-Centric Networking

From Theory to Practice

Christian Tschudin
University of Basel, Switzerland
christian.tschudin@unibas.ch

Ersin Uzun
Palo Alto Research Center
ersin.uzin@parc.com

Christopher A. Wood
Palo Alto Research Center
christopher.wood@parc.com
Agenda

• ICN and CCN overview
• Content-Based Security and Trust
• Core trust logics for verification/signing
• CCNx Trust Engine Design
• Conclusion
Information Centric Networks

• Aims to evolve away from a host-centric paradigm to a network architecture in which the focal point is “named information”.

• **Mobility** and **multi access** are the norm and **anycast, multicast, broadcast** are usually natively supported.

• **Data is independent from location, application, storage, and means of transportation**, enabling in-network caching and replication.
CCN Overview

• CCN (and its sister architecture NDN) is one well known example of ICN
  – Data is obtained via an explicit request for the name with an interest
  – Consumers issue interests that are routed towards the data producer (using the name)
  – A content object carries the data back to the consumer
Example

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

$Cr_B$

/youtube/videos/presidentspeech
Example

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

$Cr_B$

Interest

/youtube/videos/presidentspeech
Example

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

$Cr_B$

Interest

/youtube/videos/presidentspeech
Example

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

$Cr_B$  $R_4$

Interest

/youtube/videos/presidentspeech
Example

\[ C_r_A \rightarrow R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow P \]

\[ C_r_B \rightarrow R_4 \]

Interest

\[ /youtube/videos/presidentspeech \]
Example

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

$Cr_B$  $R_4$

/youtube/videos/presidentspeech
Example

$C_{r_A}$ $R_1$ $R_2$ $R_3$ $P$

$C_{r_B}$

/youtube/videos/presidentspeech
Example

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

$Cr_B$  $R_4$

/youtube/videos/presidentspeech
Example

\[ C_r_A \rightarrow R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow P \]

\[ C_r_B \rightarrow R_4 \]

/youtube/videos/presidentspeech
Example

\( C_{r_A} \quad R_1 \quad R_2 \quad R_3 \quad P \)

\( C_{r_B} \quad R_4 \)

/youtube/videos/presidentspeech
Content-Based Security
Connection-Based Security

Today’s internet secures *connections*, not *content*:

https://online.wellsfargo.com/policies.html
Content-Based Security

Secure the *content*, wherever it travels…
…get it from anyone who has a copy.

online.wellsfargo.com/policies.html
Securing Content in CCN

Content Packet = \langle name, data, signature \rangle

In theory, any consumer can verify:

- Integrity: is data intact and complete?
- Origin: who asserts this data is an answer?
- Correctness: is this an answer to my question?
Trust in Application-Layer

- How does a consumer application determine which content is trusted?
  - A valid digital signatures doesn’t mean content is authentic or trustworthy
  - Trust decisions can only be made within a particular and potentially complex trust context (e.g., given set of trust anchors, rules and exceptions).
Trust in Network-Layer

• How network-layer machinery can enforce trust context of applications?
  – How do routers determine what content they should/can use to respond to requests
  – How the network stack can request/deliver content that the application would trust
Sample Trust Models

• Pre-shared keys
  – Massage Authentication Codes

• PKI
  – Traditional
  – Constrained
    • e.g., Yu et. al., Schematizing trust in NDN

• Web-of-Trust
  – PGP
Theory to Practice

• **Architectural design** that enables efficient representation and enforcement of trust preferences at the network-layer
  – CCNx requests can have either of content hash or publisher Key ID restrictions

• **A design/implementation of a machinery** that can translate any application-layer trust semantics to network-layer mechanics and enforce them during content publishing/consumption.

In this paper, we show the design logic and an instance implementation of such a machinery in CCNx.
Core Validation Logic

```
isValidPkt(Packet, TrustContextIn, TrustContextOut) :-
Packet = pkt(DataName, _, KeyInfo, PktHash, PktSignature),
getTrustedKey(DataName, KeyInfo, TrustContextIn, TrustContextOut),
KeyInfo = key(_, _, KeyBits),
isValidSignature(PktHash, PktSignature, KeyBits),
```

• System tries to satisfy the isValidPkt() predicate by getting a trusted key and validating the packet’s signature.
• A packet has a name, the information regarding which key was used to sign, the hash value (and the signature value.
• KeyInfo usually has key’s name, ID, and always the key value

• Underscore is used to leave some fields optional
Core Validation Logic

isValidPkt(Packet, TrustContextIn, TrustContextOut) :-
    Packet = pkt(DataName, _, KeyInfo, PktHash, PktSignature),
    getTrustedKey(DataName, KeyInfo, TrustContextIn, TrustContextOut),
    KeyInfo = key(_, _, KeyBits),
    isValidSignature(PktHash, PktSignature, KeyBits),

getTrustedKey(_, KeyInfo, TrustContext, TrustContext) :-
    TrustContext = trustCtx(_, TrustedKeyList, _),
    member(KeyInfo, TrustedKeyList).

getTrustedKey(DataName, KeyInfo, TrustContextIn, TrustContextOut) :-
    fetchTrustedKey(DataName, KeyInfo, TrustContextIn, TrustContextOut).

All differences among the trust models are now isolated to the getTrustedKey() predicate
Model-Specific Variations: MAC

```prolog
fetchTrustedKey(_, _, Context, Context) :-
    Context = trustCtx('preshared', _, _), fail.
```

- In the simplest trust model, symmetric session keys are pre-shared
- Consequently, the `fetchTrustedKey()` is a failing action if the key is not already known
Model-Specific Variations: Hierarchical/Schematized

- **Basic hierarchical model**: fetch certificate chain until a trusted certificate is found...
- **Schematized model**: make sure additional constraints on data and key names and explicit authorizations are satisfied.

```prolog
defetchTrustedKey(DataName, KeyHint, TrustContextIn, TrustContextOut) :-
    KeyHint = key(KeyLocator, _, KeyBits),
    TrustContextIn = trustCtx(Model, _, Aux),
    ( Model = 'hierarchical'
        ;
        Model = 'schematized', % Aux has the list of schemas
            member(schema(KeyLocator, DataName), Aux)
    ),
    ccnFetchCert(KeyLocator, CertPkt),
    CertPkt = pkt(KeyLocator, KeyBits, _, _, _),
    isValidPkt(CertPkt, TrustContextIn, TrustContextTmp),
    TrustContextTmp = trustCtx(Model, KeyList, Aux),
    TrustContextOut = trustCtx(Model, [KeyHint | KeyList], Aux).
```
Signing Logic

• Relatively straightforward as applications usually know their identity and key
• Example logic below consists of find a suitable schema, picking a viable certification path, and signing using the corresponding key

```prolog
getSigningName(NameToBeSigned, TrustContext, [SignerName|Tail]) :-
  TrustContext = trustCtx('schematized', KeyList, Schema),
  member(schema(SignerName, NameToBeSigned, Schema),
  (member(key(SignerName, _, _), KeyList), Tail= []
  ;
  getSigningName(SignerName, TrustContext, Tail)
).
```
Theory to Practice:
The CCNx Trust Engine
Implementation

The trust engine is composed of three functions:

- **InspectPacket**: pull out packet info
- **FetchTrustedKey**: obtain the trusted verification key (and update the trust context).
- **VerifySignature**: verify the signature using the trusted key
CCN Trust Engine Overview

Trust Engine

Verification FSM

(1) process

Whitelist and Blacklist Trust Check

Schematized Trust $\text{FetchTrustedKey}$

(2) accept

Output Queue

(3) reject

Interrupt or exception

(4) request

Request the required key and go to next packet in queue

Key Store

Fetch Trusted Key

Inspect Packet

Verify Sig.

Is key in the key store?

Is key rule in schema?

Lookup key certificate and register listener

return key

reject

defer

no

yes

defer

no

yes
Conclusion

• In ICNs, network needs to deliver content that consumer applications would trust – otherwise it is non-functional!

• This paper demonstrates how to design and implement a machinery that
  – translates trust context/model of applications to network-layer mechanics that can enforce them
  – can handle variety of potentially complex trust models with simple unified logics for easy understanding/implementation
  – provides easy checks for potential pitfalls such as verification loops and weak certification links
  – is instantiated by a full working implementation on CCNx codebase.
Thanks!

Any question?

You can contact christopher.wood@parc.com for all prolog predicates, TR version of the paper and the CCNx implementation