

# Introduction to a Simple Demo NDN Application

**NDN Tutorial – ACM ICN 2016**

September 26, 2016, Kyoto, Japan

Alex Afanasyev and Davide Pesavento

<https://named-data.net/icn2016-tutorial>

# Goals for Today

Learn how to start coding NDN applications

- code templates
- security tools

Learn and appreciate new qualities of NDN applications

- power of NDN network to get “closest” data and data from caches
- data-centric security with schema-based trust

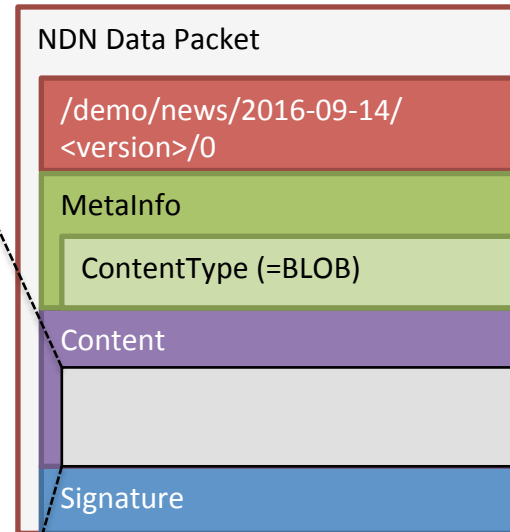


# “Web” Page

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Mega News</title>
  </head>
  <body>
    <h1>Mega News for September 14, 2016</h1>

    <ul>
      <li>Demo News</li>
    </ul>

    <div id="weather" src="ndn:/demo/weather/
current">
    </div>
    ...
  </body>
</html>
```

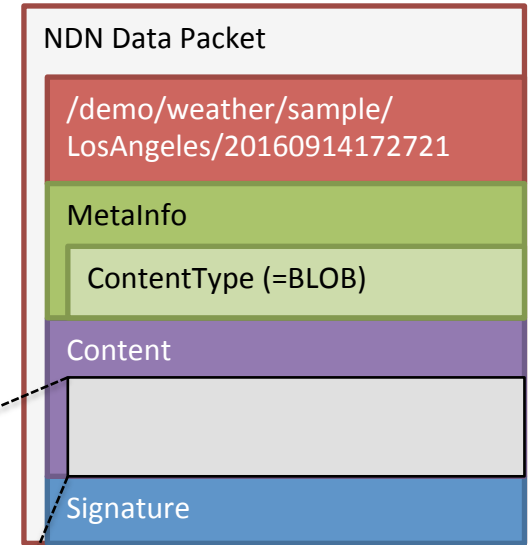


HTML to be requested  
by NDN-enabled  
browser (or a stand-  
alone NDN client)

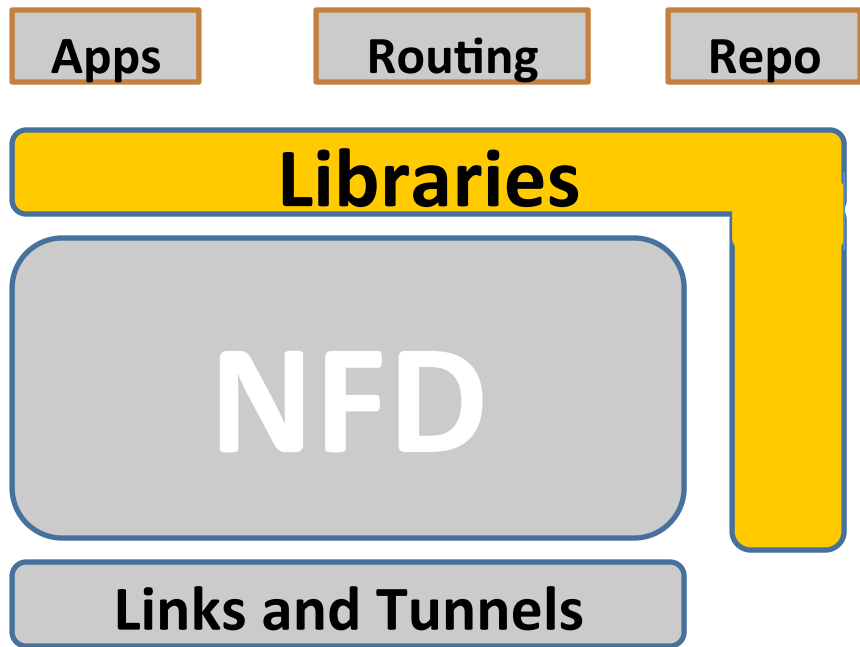
# Sensor Data

JSON Object (will be requested by JavaScript using NDN-JS)

```
{  
  "city": "Los Angeles",  
  "time": "2016-09-14 17:27:21 PST",  
  "temperature": "74"  
}
```



# Available Tools for NDN Developers



- ndn-cxx
  - C++11 library
  - <http://named-data.net/doc/ndn-cxx/>
- NDN-CCL: NDN-JS, jNDN, PyNDN, ndn-cpp
  - JavaScript
  - Java
  - Python
  - C++
  - <https://named-data.net/codebase/platform/ndn-ccl/>
- ndnSIM
  - Simulate network and run simplified versions of apps
- Mini-NDN
  - Emulate small network and run unmodified code

# What we will learn

How to start

How to write basic functionality

- How to send interests using different APIs
- How to respond to interests
- How to create content object

How to use some advanced functionality

- How to generate keys
- How to sign data packets
- How to verify data packets and keys

# How to Start

- Install and run NFD
  - <http://named-data.net/doc/NFD/current/>
- Pick library to use
- Start writing application, e.g., by using an app skeleton
  - <https://github.com/cawka/ndn-skeleton-apps>



# Overall Picture for NDN app

Create a Face instance

- Kind of equivalent to “socket”

Register prefix(es) to receive Interests (for producer apps)

- Define callbacks to be called when an Interest is received

Express Interests (for client apps)

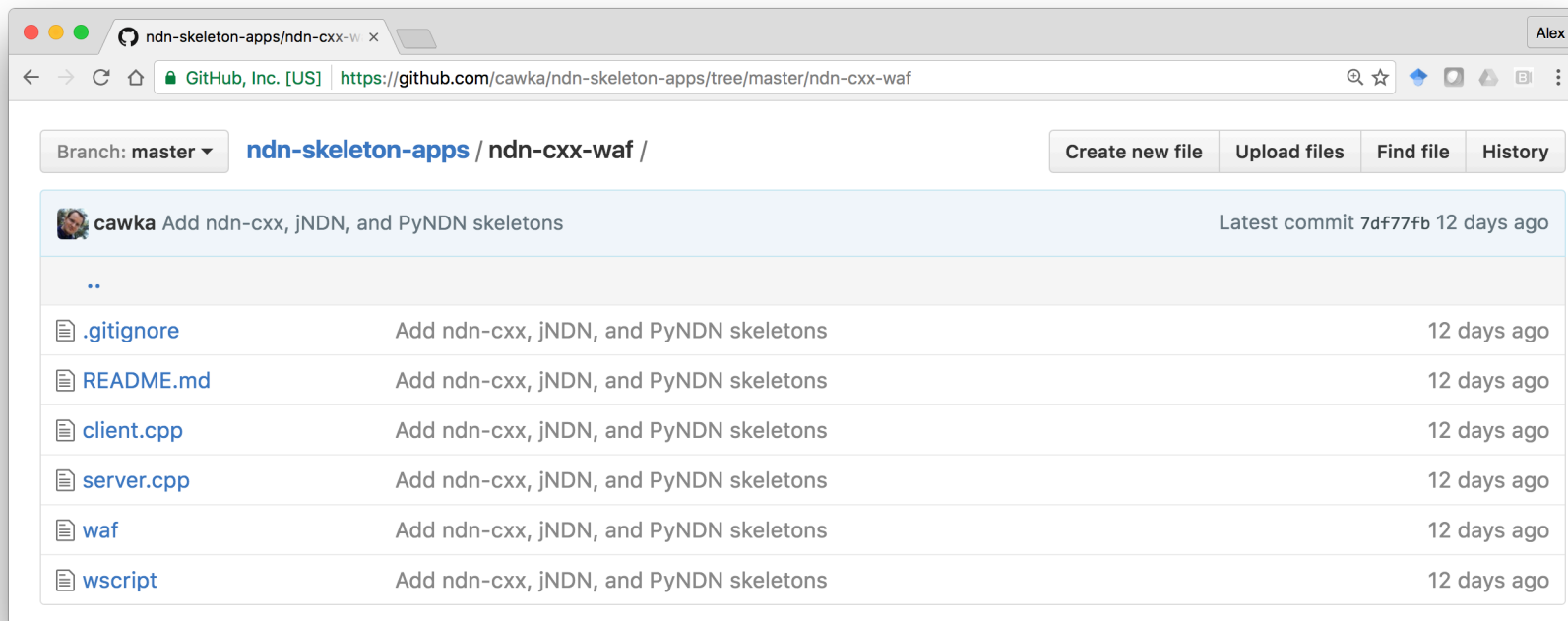
- Define callbacks to be called when:
  - Interest is satisfied
  - Interest times out, or
  - NACK is received

Start the processing loop

Asynchronous application  
model

# Let's Explore the Skeleton Apps

<https://github.com/cawka/ndn-skeleton-apps/tree/master/ndn-cxx-waf>



The screenshot shows a web browser window displaying the GitHub repository page for `ndn-skeleton-apps / ndn-cxx-waf`. The browser's address bar shows the URL `https://github.com/cawka/ndn-skeleton-apps/tree/master/ndn-cxx-waf`. The repository page includes a navigation bar with buttons for "Create new file", "Upload files", "Find file", and "History". Below the navigation bar, a commit message is displayed: "cawka Add ndn-cxx, jNDN, and PyNDN skeletons" with the latest commit hash `7df77fb` and a timestamp of "12 days ago". A list of files is shown, each with a document icon, a filename, a commit message, and a timestamp:

File	Commit Message	Timestamp
..		
<code>.gitignore</code>	Add ndn-cxx, jNDN, and PyNDN skeletons	12 days ago
<code>README.md</code>	Add ndn-cxx, jNDN, and PyNDN skeletons	12 days ago
<code>client.cpp</code>	Add ndn-cxx, jNDN, and PyNDN skeletons	12 days ago
<code>server.cpp</code>	Add ndn-cxx, jNDN, and PyNDN skeletons	12 days ago
<code>waf</code>	Add ndn-cxx, jNDN, and PyNDN skeletons	12 days ago
<code>wscript</code>	Add ndn-cxx, jNDN, and PyNDN skeletons	12 days ago

# App Details

- News service (“web” server)
  - C++ (ndn-cxx), publishing HTML
- Weather service
  - C++ (ndn-cxx), publishing JSON from “sensors”
- News client
  - JavaScript (NDN-JS)
  - Initial page served from a local webserver
  - In-page NDN communication over WebSocket to the local NFD or to a remote hub
- Will also use NDN tools to test website and webservice without “real” client

# A “Little Bit” About Security

- NDN secures data
- Publishing apps need to sign data packets properly
  - Based on the selected trust model
- What could be the trust model in our demo?

*/demo/news/page/<date>/<version>/<segment>*


*/demo/weather/sample/Kyoto/<time>*

*/demo/weather/sample/LosAngeles/<time>*

# Data-Centric Security (News Page)

<code>/demo/news/page/2016-09-25/%FD%00%00%01W_%84%A7%3B/%00%00</code>
HTML page
Signature <code>/demo/news/KEY/ ksk-1474377249714/ID-CERT</code>

`/demo/news/page/<date>/... MUST BE signed by /demo/news/KEY/...`

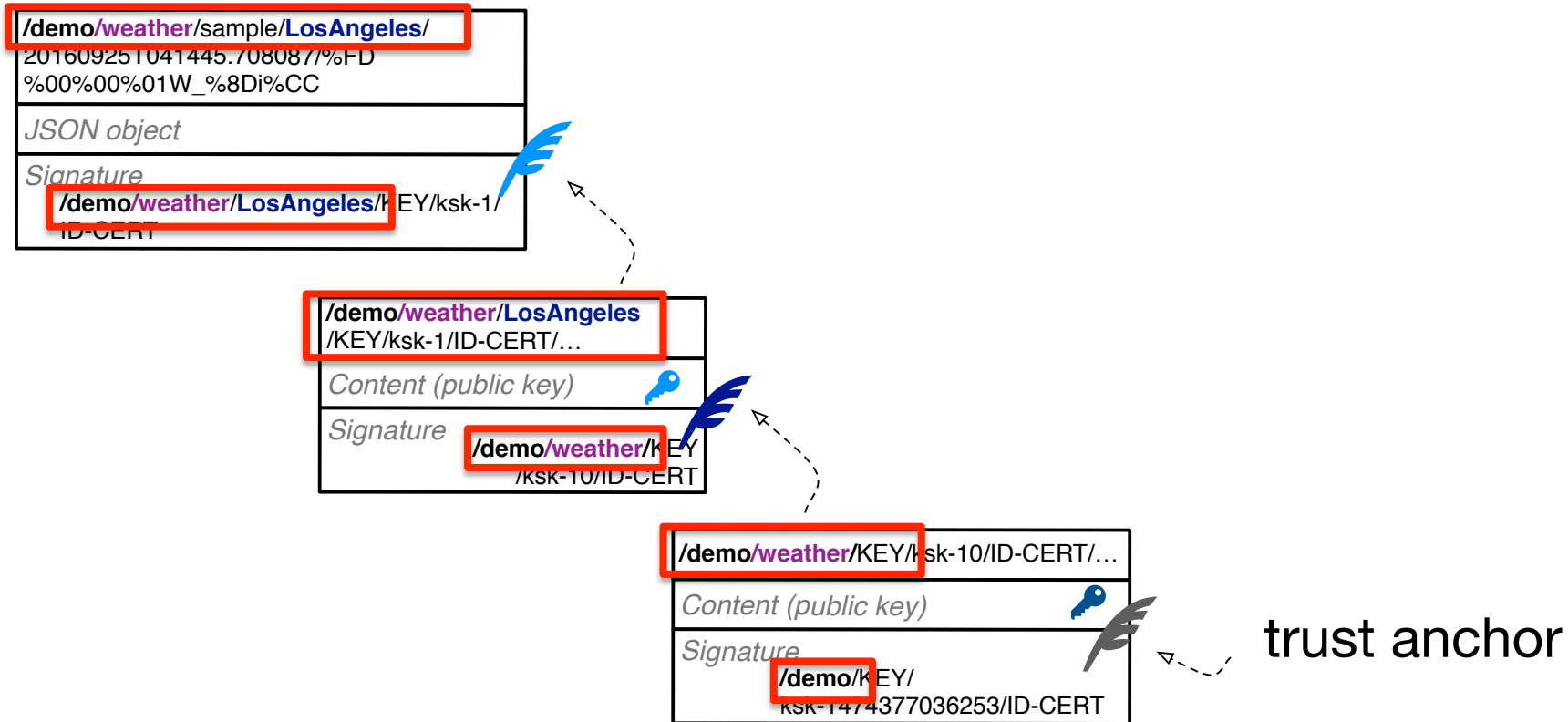
<code>/demo/news/KEY/ ksk-1474377249714/ID-CERT/...</code>
Content (public key) 
Signature <code>/demo/KEY/ ksk-1474377036253/ID-CERT</code>

`/demo/news/KEY... MUST BE signed by /demo/KEY/...`

<code>/demo/KEY/ksk-1474377036253/ ID-CERT/...</code>
Content (public key) 
Signature 
...

trust anchor

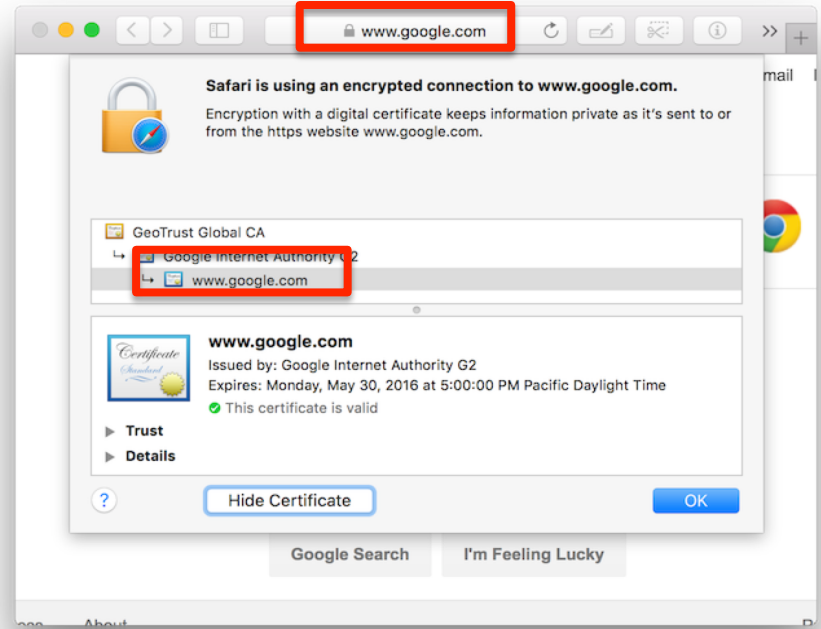
# Data-Centric Security (Weather)



# How is It Different from Today's HTTPs?

Domain name has to match name in the issued HTTPS certificate. BUT

- security is associated with **individual and immutable data packets**, not the transient connections
- different packets can be signed with **different keys**
  - “privilege separation”
- can work when consumer/producer **are not online at the same time**



# Summary for NDN Key Management

- Key generation
  - `ndnsec-keygen`
- Create signing request
  - `ndnsec-sign-req`
- Signing key (generation of a certificate)
  - `ndnsec-cert-gen`
- Publishing certificates
  - For example, using bulk insert protocol of `repo-ng`
    - `ndnsec-cert-dump -r <cert-name>`
    - `sudo repo-ng-ls #` to check contents of the `repo-ng`



# News Key Generation

- On news host: generate key and create signing request

```
ndnsec-keygen /demo/news  
ndnsec-sign-req /demo/news > request.ndncert  
# send request to trust anchor site
```

- On trust anchor site: generate certificate and the certificate

```
ndnsec-cert-gen -N "News Server" -s /demo -p /demo/news  
request.ndncert > issued.ndncert
```

- On news host: install and publish the issued certificate

```
ndnsec-install-cert issued.ndncert
```

```
ndnsec-dump-certificate -r /demo/news/KEY/ksk-1474377249714/ID-CERT/  
%FD%00%00%01WG%C3%0A%96
```

# Trust Anchor Key Generation

- Generate key

```
ndnsec-keygen /demo
```

- Create and install self-signing certificate

```
ndnsec-sign-req /demo | ndnsec-install-cert -
```

- \*\*\* HOME environment variable controls where the key database is stored.  
For example, to keep trust anchor in “secret” folder

```
export HOME=/secret/folder  
ndnsec-ls-identity
```

```
HOME=/secret/folder ndnsec-ls-identity
```

# Other Hints

- To see installed certificates
  - `ndnsec-ls-identity` # list configured identities
  - `ndnsec-ls-identity -v` # list configured identities and generated keys
  - `ndnsec-ls-identity -vv` # list configured identities, generated keys, and installed certificates
  - `ndnsec-ls-identity -vvv` # same plus detailed information about each certificate
- Export / import private keys (e.g., when moving between machines/ accounts)
  - `ndnsec-export -p <identity>`
  - `ndnsec-import -p <file>`
- Show certificate info
  - `ndnsec-cert-dump [-h] [-p] [-d] [-r [-H repo-host] [-P repo-port] ] [-i|k|f] name`

# Sensor Key Generation

- LA sensor

```
ndnsec-key-gen /demo/weather/LosAngeles > weather.certreq  
# send request to weather service key management site
```

- On weather service key management site

```
ndnsec-cert-gen -p /demo/weather -N "LA Weather" -r weather.certreq  
> weather.cert  
ndnsec-dump-certificate -r /demo/weather/KEY/LosAngeles/  
ksk-1474778889387/ID-CERT/%FD%00%00%01W_%AD%07%5E
```

- LA sensor

```
ndnsec-install-cert weather.cert
```

