NDN-IoT: a readily usable package for experimentation with IoT over Named Data Network

ZHIYI ZHANG, YANBIAO LI, EDWARD LU, TIANYUAN YU, ALEX AFANASYEV, LIXIA ZHANG

NDNCOMM

SEPT 2018

Objectives

- ◊ Audience: people interested in NDN but don’t know where to start
  - Or just want an easy start
- ◊ Make a “all-in-one” IoT demo package based on NDN-RIOT
  - a integrate and modularized open-source library
  - well-documented APIs
  - Some pre-defined naming convention for different services to cooperate
- ◊ Users may
  - Just to play around
  - develop new apps
  - Further extend the package (along all software/hardware dimensions)
- ◊ Non-goals
  - Wide platform availability
  - heterogenous network technologies supporting

Documentation

- ◊ Introductory whitepaper
- ◊ user guide
  - ○ Compatible hardware
  - ○ how to download, install, and turn on
  - ○ Make a How-to YouTube video
- ◊ App developer’s guide
- ◊ System developer’s guide
- ◊ Visualization of what is going on to demonstrate NDN functionality

Developing a community

- ◊ First and foremost: autoconfiguration, usability, resiliency
  - ○ **Jeff:** no one would bother to try if you don't have resilient operation
- ◊ Set up a mailing list
- ◊ Strongly encouragement on comments and feedbacks
  - ○ Some token awards or recognitions?
- ◊ Visualize system reactions actions
- ◊ Inviting attacks?

Goals of NDNoT Library

Providing integrated and lightweight NDN support in IoT scenario:

◊ Basic NDN protocol stack and communication features

◊ NDN running over link layer

◊ Security bootstrapping

◊ Service discovery

◊ Schematized Trust

◊ Usable Access Control for constrained devices

◊ NDN Sync support

Hardware

IOT devices

◊ Atmel Xpro (RIOT OS): 802.15.4

◊ ESP32: WiFi, BLE, Bluetooth

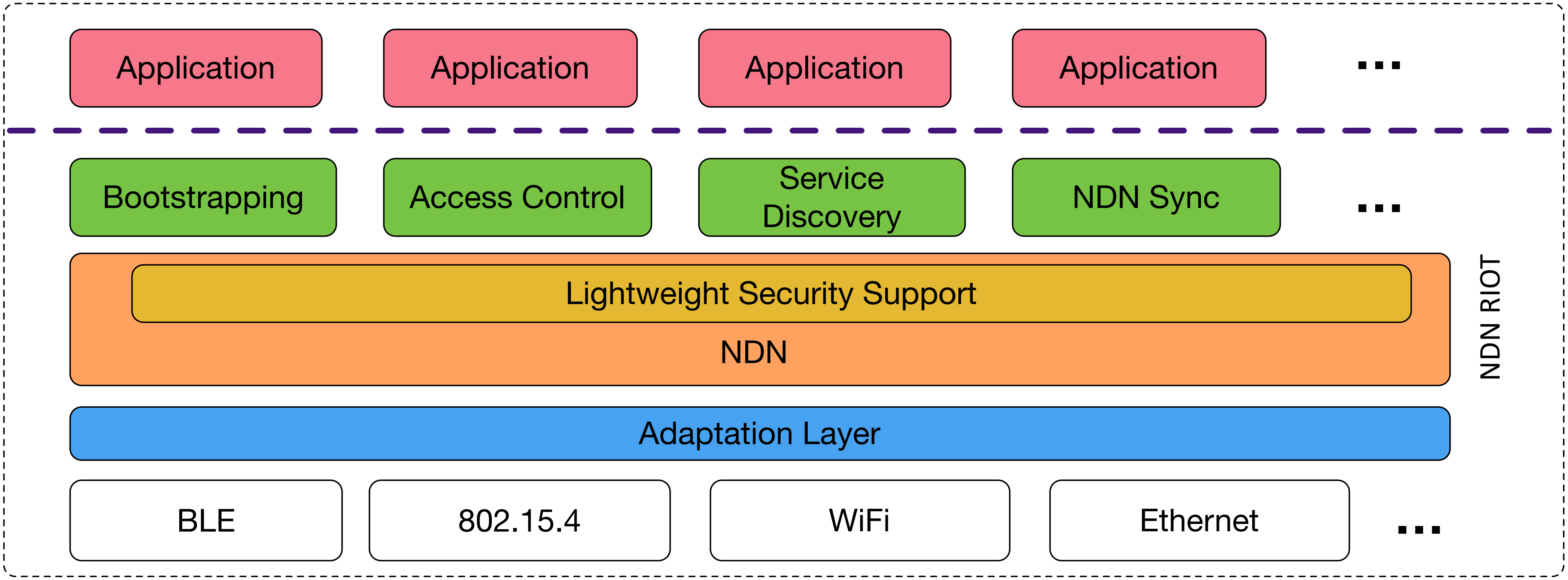
Controller

◊ Raspberry Pi

◊ Android Phone

◊ Linux/MacOS

IoT Device Software Framework



A simple story

- One buys a smart home temperature sensor with a IoT board that only has 32k RAM and 48MHz
- What's next?

Bootstrapping

Goal

- ◊ The IoT device (e.g., Temperature Sensor) **learns the trust anchor** of the system and **obtain an identity certificate** issued by the system controller (e.g., Android Phone)

Assumptions

- ◊ The IoT device and the home controller have **shared secret** through out-of-band means
  - e.g., the user uses his phone to scas the QR code on the sensor
- ◊ Use the shared secret as a crypto **public key** (BK), e.g., ECC/RSA public key

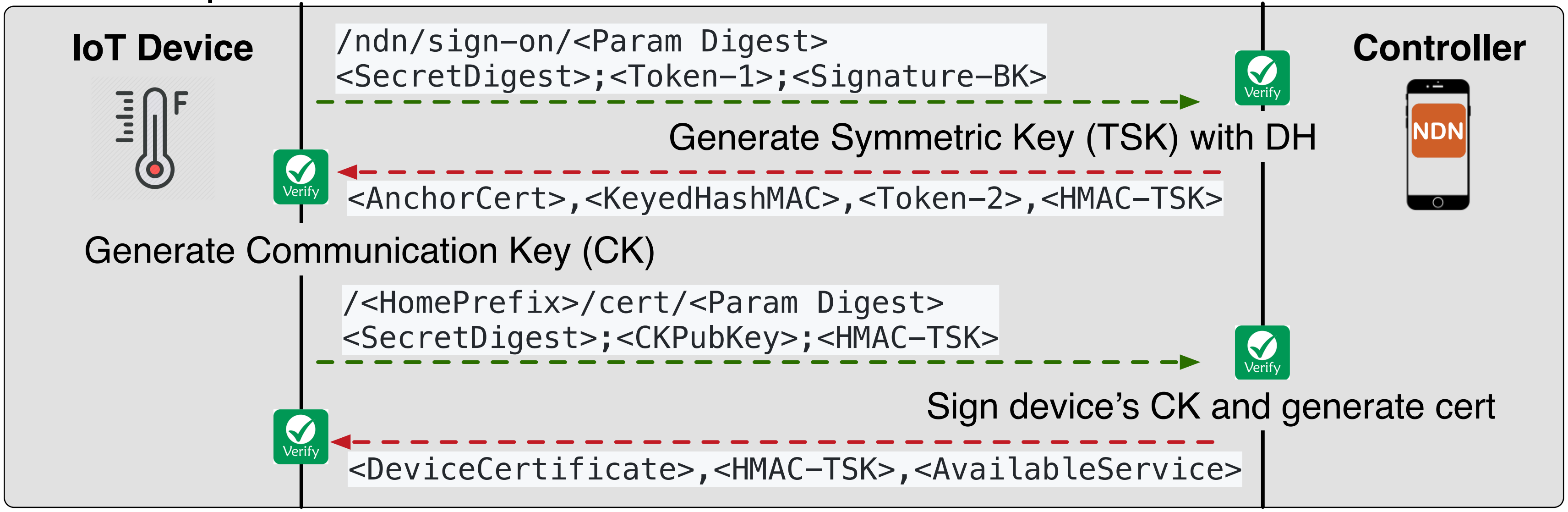
Bootstrapping

◊ Identify each other by verifying the possession of shared secret.

◊ Negotiate a symmetric key for better performance

◊ Utilize uniqueness to prevent replay attack

◊ Use Interest parameter to save bandwidth



/ndn/sign-on/<Param Digest>
<SecretDigest>;<Token-1>;<Signature-BK>

Generate Symmetric Key (TSK) with DH

<AnchorCert>,<KeyedHashMAC>,<Token-2>,<HMAC-TSK>

Generate Communication Key (CK)

/<HomePrefix>/cert/<Param Digest>
<SecretDigest>;<CKPubKey>;<HMAC-TSK>

Sign device’s CK and generate cert

<DeviceCertificate>,<HMAC-TSK>,<AvailableService>

Bootstrapping Assessment and Performance

Assessment

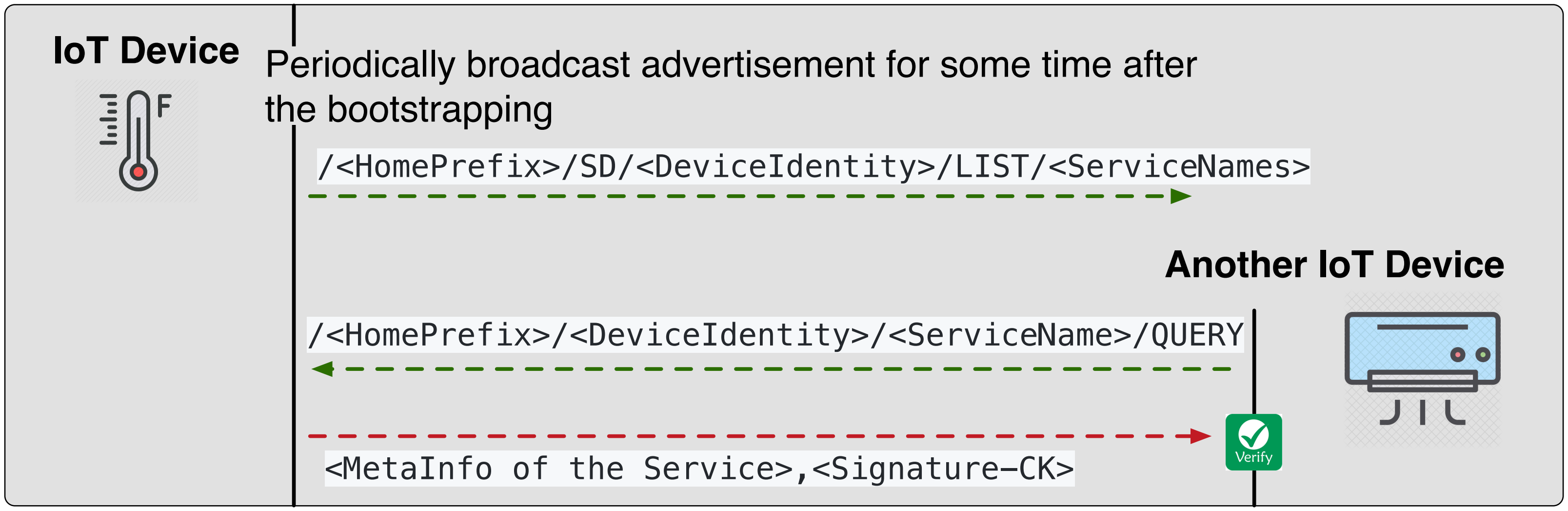
- ◊ One asymmetric signature signing and verification (I1)
- ◊ One Diffie Hellman Process
- ◊ Three HMAC signing and verification (D1, I2, D3)

Performance:

- ◊ Time Consumption: sec(s) (including network and system IO) for Xpro (with RIOT) board (32K RAM, 48MHz)
- ◊ Details: ECC key size 160 bits; DH key size 256 bits
- ◊ Bandwidth Consumption: around 300 bits less by utilizing Interest parameters

Service Discovery

- ◊ Learning existing services from the controller in the last step of bootstrapping
- ◊ Advertising services by broadcasting advertisements after bootstrapping
- ◊ Broadcasting again when services change or restart (soft state)
- ◊ Query meta data before using a service



IoT Device

Periodically broadcast advertisement for some time after the bootstrapping

`/<HomePrefix>/SD/<DeviceIdentity>/LIST/<ServiceNames>`

Another IoT Device

`/<HomePrefix>/<DeviceIdentity>/<ServiceName>/QUERY`

`<MetaInfo of the Service>,<Signature-CK>`

Schematized Trust

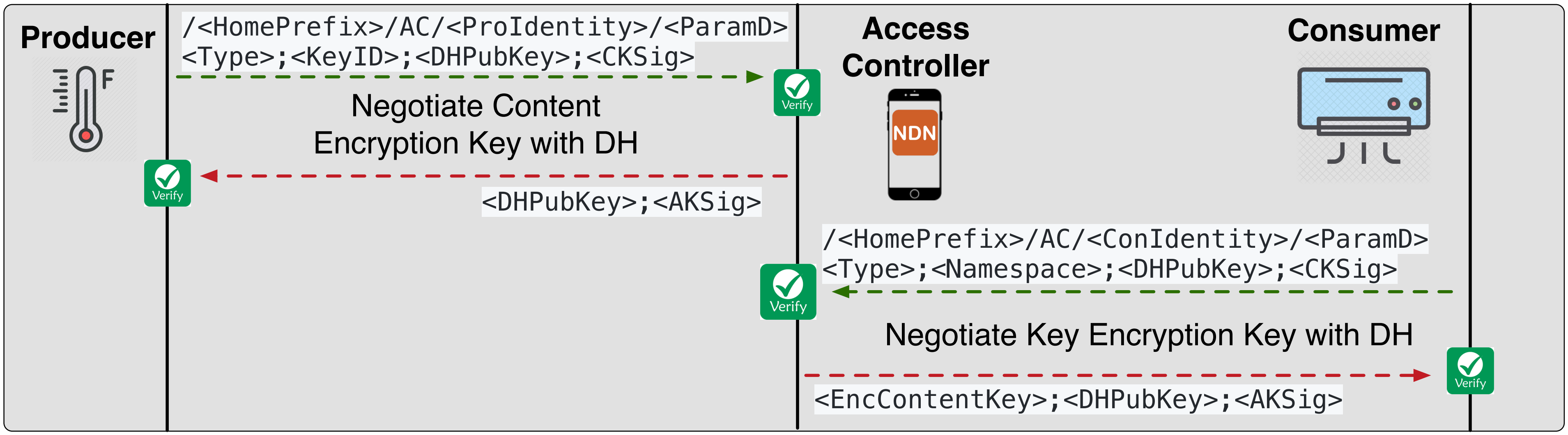
- Control IoT device’s trust relationship with other devices in different scenarios

Example:

- The AC (/home/living/AC) should only trust the temp data (/home/living/temp) under the same prefix
- The AC should only obey the command signed by the device with controller prefix (/home/control) or with specific format (/home/living/remote-<>)

Lightweight Access Control

- ◊ Existing implementation of NDN access control doesn't fit constrained devices
- ◊ Instead use all symmetric key encryption/decryption
- ◊ Use Interest parameter to save bandwidth



Producer

Access Controller

Consumer

```
/<HomePrefix>/AC/<ProIdentity>/<ParamD>
<Type>;<KeyID>;<DHPubKey>;<CKSig>
```

Negotiate Content Encryption Key with DH

```
<DHPubKey>;<AKSig>
```

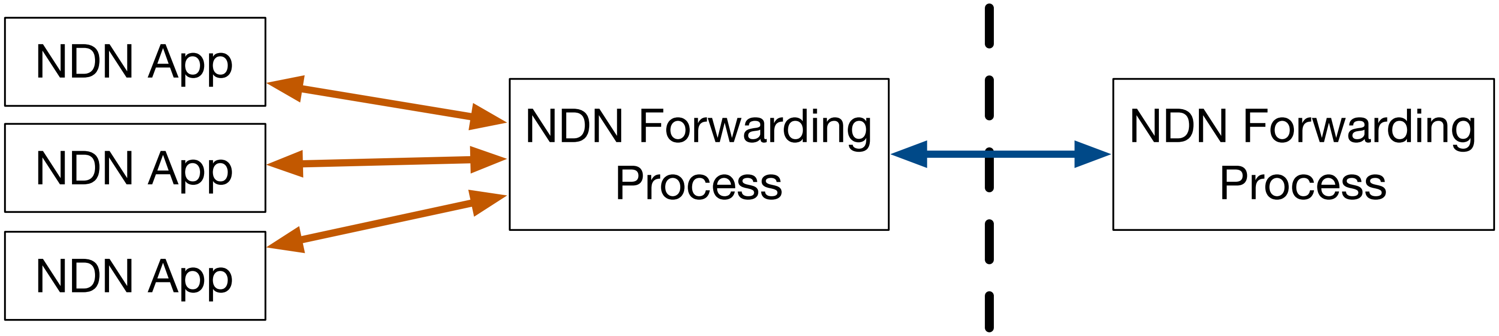
```
/<HomePrefix>/AC/<ConIdentity>/<ParamD>
<Type>;<Namespace>;<DHPubKey>;<CKSig>
```

Negotiate Key Encryption Key with DH

```
<EncContentKey>;<DHPubKey>;<AKSig>
```

Adaptation Layer

- The adaptation layer abstracts different link-layer protocols and wraps the NDN Interest and Data packets into link-layer frames.
- Name Prefix <-> Interface mapping
- A separate process and communicates with NDN applications using Inter-Process Communication (IPC) or other equivalent mechanism.



Current status and future plan

- Finished with unit tests:
  - NDNoT for RIOT: Bootstrapping
  - NDNoT for RIOT: Service Discovery
  - NDNoT for RIOT: Access Control

- In Progress
  - Adaptation Layer
  - Specification
  - Tutorial

- Next stage
  - NDNoT for RIOT: schematized trust
  - NDNoT for RIOT: sync
  - NDNoT for RIOT: integrate test
  - NDNoT for ESP32

Thank You!