Trust Schemas and ICN: Key to Secure Home IoT

Kathleen Nichols
Pollere, Inc.
Montara, CA, USA

ABSTRACT

Home and business internet of things (IoT) presents security challenges that can be addressed using information-centric networking (ICN) to secure information rather than channels. In particular, we leverage ICN's per-packet signing, combined with recent innovations in trust schemas, to construct a strong *trust zone*. This architecture creates domains governed by a secured trust schema provided to every device during its enrollment together with the device's attribute-based signing cert chain(s). Applications don't need to be rewritten to gain security; a run-time library with an MQTT-like publish/subscribe API uses the provisioned trust schema and certs to construct, sign and ship outgoing publications and to both cryptographically and structurally validate a subscriber's incoming publications. This unique application of trust schemas (Versec) is explained and an example home IoT framework is described where trust schemas express straightforward, homeowner-specific policies that an open-source library enforces at run-time on behalf of security-agnostic applications. Along with the specific innovation in trust management, the platform exploits current and emergent IoT best practices. Utility programs, libraries, and examples are available as an open-source Data-Centric Toolkit.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; **Domain-specific security and privacy architectures**; • **Networks** → **Home networks**; **Network protocols**.

KEYWORDS

Information-Centric Networking, Internet of Things, Trust management, Secure IoT

ACM Reference Format:
Kathleen Nichols. 2021. Trust Schemas and ICN: Key to Secure Home IoT. In *8th ACM Conference on Information-Centric Networking (ICN '21), September 22–24, 2021, Paris, France.* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/3460417.3482972

1 INTRODUCTION

Securing IoT is difficult: successful attacks continue despite the ongoing search for solutions (see for example [4, 54, 63, 64, 69]). A great deal of trust has been put in encryption at the MAC layer or in the IP protocol stack (TLS/TCP or DTLS/UDP). IoT platforms from companies like Amazon and Google require connection to their cloud-based servers and have focused on securing that connection and providing privacy to the data sent between endpoints over the channel. This fails to appreciate that commands can be sent *privately*, but may not be *authentic*. Although commissioning (provisioning or on-boarding) of new devices in home IoT networks has been evolving to take advantage of trusted platform modules (TPMs[36]) and become more secure, the operational security generally relies on users changing default passwords and performing sometimes difficult configuration yet does not prevent the hijacking of low-value devices (e.g., lightbulbs) to gain control of high-value networked devices like door locks and cameras [23].

Home IoT applications have a very different communication model than the endpoint-oriented one embodied in prevalent Internet protocols. In IoT many devices issue messages and many devices may be interested in those messages. Thus, publish/subscribe (pub-/sub) communication APIs are ubiquitous with nearly all platforms supporting the application-layer protocol MQTT (mqtt.org), e.g., HomeAssistant [26] and Samsung's IoT ARTIK Cloud [65]. Yet network layer implementations typically use multiple endpoint transport connections (to a central broker/server) and channel-based security protocols (e.g., TLS/TCP) that have repeatedly been found insufficient to secure IoT [4, 32, 54, 69]. In addition to the existential evidence of widespread attacks on such systems, Alex Halderman et al. [21, 39] have shown how the connection-based security of the Internet is a *mismatch* to content- (or information-) based systems and note the issues could be remediated by moving from perimeter- to content-based security. The Zero Trust cybersecurity conceptual framework [52, 70, 71] aims to shift security focus to preventing the unauthorized access of data and services rather than relying on securing channels with perimeter constructs. Zero Trust requires a strong identity for both devices and users, an area where significant progress has been made [9, 49, 50, 57]. It also requires granular access control enforcement that makes use of those identities and here progress has been lacking. Our contribution is focused on providing distributed granular access control, securing the means of control as well as the application data.

Though research on ICN protocols for Internet use[1] is not yet Internet-ready, ICN (specifically, NDN) network layer primitives can be combined with trust schemas [75] in a new approach [41], called Versec, to create a certificate-based, trust zoned architecture to solve critical problems for edge networks today. A *trust zone*, or domain, is a Zero Trust network governed by a single trust anchor and trust schema and enforced at run-time by a library using a signed binary copy of the schema at each entity. Trust schemas specify fine-grained rules governing names and signing chains to go beyond simple signer validation to enforce "who can do what to which." Devices in a trust zone are commissioned with an *identity bundle* (§ 3) of trust anchor, signed trust schema, and the signing

[1] e.g., https://named-data.net/, https://wiki.fd.io/view/Cicn, https://irtf.org/icnrg

ICN '21, September 22–24, 2021, Paris, France

ACM ISBN 978-1-4503-8460-5/21/09…$15.00
https://doi.org/10.1145/3460417.3482972

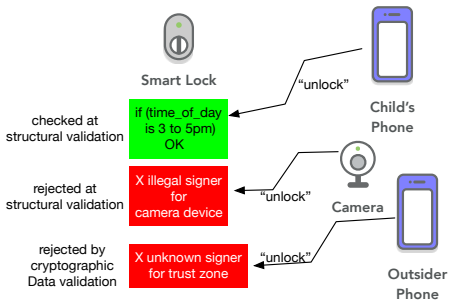

Figure 1: Devices invoke domain rules at validation

certs associated with a particular identity; nothing is accepted without validation. This paper describes the application of this approach to pub/sub communications in a home IoT network where its trust zone can prevent attacks that allow *authorized* devices on a network to issue *inauthentic* high-value commands. Communications from outsider devices have unknown signatures and can be discarded with little use of resources. Publications (actionable directives) can be restricted by roles ("owner", "child", "guest"), capabilities (e.g., ability to issue particular commands), and locations ("living room lights", "ceiling fan", "front door lock") where each entity must have (via its identity bundle) the certificates specified by the trust schema in order to build a particular publication. Figure 1 illustrates a trust schema at the Smart Lock being used to reject command publications from a camera (lacks role certificate to issue commands), to test a publication signed by a user with a restricted role (time-of-day), and to reject Data from a non-member of the trust zone. The outsider phone may have (legitimate) access to the local wifi network and still be excluded from the trust zone.

Trust zoned applications are implemented using our open source Data-Centric Toolkit (DCT) [59]. DCT includes a declarative language for expressing the rules of the trust schema, a compiler that turns the schema into binary, and a utility to package the binary into a certificate. Other utilities create the certificates of the signing chains and identity bundles (§ 3). Library modules employ the trust schema at run-time for Data construction, signing, and validation. Additional run-time modules distribute signing chains for provenance validation and group keys for symmetric encryption throughout the trust zone. Since security is specified in the trust schema, application code does not change as the security approach evolves. While application binary is unchanged, the security model can change from one that merely specifies that the signer must be known to one that adds fine-grained capability control and can add encryption. In addition to this paper's trust zoned home IoT, *dctIoT*, DCT is being used in a secure transport for a legacy intrusion detection system [42].

Section 2 covers the evolution of trust schemas and presents the Versec approach used by DCT. Section 3 defines the composition of identity bundles. Section 4 presents naming and trust schema for *dctIoT*. Section 5 describes DCT and *dctIoT* library modules. Section 6 notes related work and section 7 summarizes and discusses future work.

2 EVOLUTION OF TRUST SCHEMAS

2.1 Background

ICN fundamentals include uniquely named data packets that reflect content and the requirement that every data packet be cryptographically signed to ensure integrity and provenance. Early in the evolution of CCN, Smetters [67] proposed a novel solution of evidentiary trust and secure binding of names to content that does not require a central authority to disperse names. That work is partially based on the seminal SDSI [62] approach to create user-friendly namespaces creating transitive trust through a certificate (cert) chain that validates locally controlled and managed keys, rather than requiring a global Public Key Infrastructure (PKI). Certificates are created that have a particular context in which they should be utilized and trusted rather than conferring total authority. In a 1996 paper, Blaze et. al. [22] defined the term *trust management* for the study of security policies, security credentials, and trust relationships. A trust management layer concept in which a system decides whether to allow some potentially dangerous action was proposed and implemented as an engine (which the authors likened to a data base query engine) where requests, certificates, and policy descriptions (rules) could be submitted for approval/disapproval. Li et. al. [44] later refines some trust management concepts arguing that the expressive language for the rules should be *declarative* (as opposed to the original work) and focusing on trust management in quite general, decentralized environments where trust across boundaries was a critical element. This body of work influences that of trust schemas, which allow for specifying security rules based on application name structures. Applying the criteria of [44] that a trust management approach answers "Does a set of credentials prove that a request complies with a policy?" suggests that trust schemas implement a specialized trust management.

Trust schemas employ the NDN protocol's primitives in a tight specification of application security rules. In 2015, Yu et al [75] described an NDN trust schema as "an overall trust model of an application, i.e., what is (are) legitimate key(s) for each data packet that the application produces or consumes" and gave a general description of how trust schema rules might be used by an authenticating interpreter finite state machine to apply the rules to packets. Currently, the NDN testbed's NLSR routing protocol adds a regular expression trust schema specification to a configuration file that gets loaded with NFD [73] and reads certificates from text files.

2.2 Versatile Security (Versec)

Jacobson [41] proposed a new approach to both the trust schema language and the integration of trust schemas with applications. He likened a trust schema to the plans for constructing a building and noted that one set of construction plans serves multiple purposes:

(1) Allows permitting authorities to check that the design meets applicable codes
(2) Shows construction workers what to build
(3) Lets building inspectors validate that as-permitted matches as-built

Construction plans get this flexibility from being *declarative*: they describe "what", not "how". As noted in [44, p.4], a declarative trust

management specification based on a formal foundation guarantees *all* parties to a communication have the *same* notion of what constitutes compliance. This can allow one schema to provide the same protection as dozens of manually configured, per-node ACL rules.

Prior trust schema work addresses only validation. Versec adds a declarative schema specification language with a compiler that checks the formal soundness of a specification (case 1 above) then converts it to a signed, compact, binary form. The binary form is used by applications via DCT library routines to build (case 2) or validate (case 3) publications. "Publications" are objects with names, content and signatures such as NDN Data packets or MQTT messages.

An early implementation of the Versec approach was described in [55]. An updated version is integrated with DCT and briefly described below. The DCT GitHub repository contains a detailed language description[19] and annotated examples[12].

2.3 Versec and IoT

Several IoT aspects make it an attractive use case for Versec and vice-versa. Foremost is IoT's administrative context. While most distributed systems are loose federations, a home IoT system is a closed community of entities acting in close cooperation to perform specific functions as delegated by a single administrative authority. Versec provides fully distributed policy enforcement and policing for this closed community without relying on a secured-perimeter physical network and/or extensive per-device configuration.

Another aspect is highlighted in RFC 8520[43], a standard providing "things" a means to describe who they need to talk to and what they need to say. This would not be useful in general but, as [43]'s introduction notes, IoT is a special case:

> "These devices, which this memo refers to as Things, have a specific purpose. By definition, therefore, all other uses are not intended. If a small number of communication patterns follows from those small number of uses, the combination of these two statements can be restated as a Manufacturer Usage Description (MUD) that can be applied at various points within a network."

Incorporating a MUD's communication constraints into a schema extends the protection they offer beyond routers, firewalls and access points to the sending and receiving devices.

A related aspect derives from IoT's economic context: most IoT devices are low cost, commodity items with fixed roles like light bulbs, switches, power plugs, etc. To be competitive, devices both must interoperate with each other and be compatible with common "hubs" like SmartThings, Home Assistant, Google Home, Amazon Alexa, etc. Thus most devices use IoT-specific communication standards like Zigbee, Zwave or BTMesh that cover not just layers 1-4 but also include detailed, per-role "profiles" to ensure interoperability. These standards allow a schema to constrain not only who talks to who but also what they talk about, using a run-time binding model that requires minimal or no changes to existing IoT device applications. A trust schema behaves much like a enrolled device's identity cert: it is compact, local configuration, interpreted at runtime, not compile or link time, so it is compatible with fixed-firmware devices and normal device firmware distribution practice.

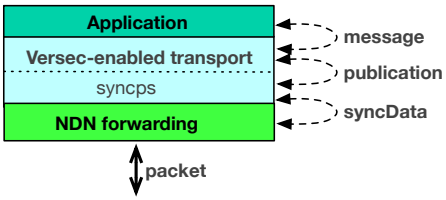

Figure 2: Using Versec-enabled transport

fig. 2 illustrates use of a Versec-enabled transport (or *bespoke transport* in [55]) where application messages are exchanged with a Versec-enabled transport whose API can be made similar or identical to current application-level message protocols. The transport translates between messages and publications that its Sync protocol, *syncps*, packages into Data for ICN forwarding (more detail in § 5). Since publication construction and signing are done *after* an application constructs its message and validation and deconstruction are done *before* the message is delivered, both are invisible to the application.

2.4 The Versec language

The Versec Domain Specific Language is a declarative description of a trust schema's rules. The *schemaCompile* compiler validates this description and translates it to a binary representation used by the run-time library to construct and validate DCT publications. The language provides a general framework for constructing self-consistent, validatable names, describing constraints on both the layout and components of Names and on the structural relationships between all components of a signing chain. It also follows LangSec [14] principles to minimize misconfiguration and attack surface. For example, the language can only produce entities (publications, certificates, signing chains, etc.) with a fixed layout so the run-time validation is loop- and recursion-free. All alternatives must be explicitly enumerated so validation and construction are take constant time and space. The language is intended to secure relatively local, mission-focused security domains (like a home or small business IoT network) where policy validity can be adjudicated via a common, local trust anchor. This means the set of signing chains for any schema must form a DAG, a property verified at compile time and exploited to make the run-time testing for signing-loops stateless and O(1).

Previous work on declarative trust specification languages [20, 44, 46] used logic programming formalisms like *Datalog*. These provide the expressiveness needed to deal with complex scenarios like constraining inter-domain delegation but at the expense of forcing system developers to program in an unfamiliar language and style. Versec focuses on expressing *constraints* within a single trust zone. This leads to a *strictly compositional constraint system*, modeled as a directed graph of the partial order induced by the constraints which, when expressed as an *order-theoretic lattice* [28], provides the same decidability, unification and subsumption capabilities as logic programming but with familiar Javascript/JSON-like structs for the programming model.[2]

[2] See [13] for a brief description and [6, 8] for background.

Versec uses a subset of the CUE language (cuelang.org) and the CUE tutorials [7] provide a useful introduction. Versec's subset is tailored to describing IoT constraints. Things like optional struct fields, default values, list and field comprehensions, regular expressions, conditional fields and null coalescing have been deliberately omitted because they introduce ambiguity that impacts schema security, understandability and runtime performance[24, 53]. Other CUE features like complete "types as values" support and the "." operator for struct field access are useful for giving Versec better visibility into application supplied arguments and are high on the "todo" list.

Language basics. A trust schema primarily deals with the meaning and relationship of individual components of the filename-like names of publications and certs. There are two equivalent representations: the linearized name with component values separated by slashes and a structure enclosed in curly braces ({...}) with the fields given in 'tag: value' form. The correspondence between these two forms is established by *definitions* like:

```
#pub: /_net/_trgt/_typ/_loc/arg/_ts & { _ts: timestamp() }
```

which defines #pub as a six component publication name. The strings between the slashes are the *tags* used to reference each component in the structure form and in the run-time API. An example of this usage is the *component constraint* following the '&'. This says two things: the type of the `_ts` component in every #pub is "timestamp" (64-bit unix timepoint in microseconds) and, when a #pub is constructed, the current time will be put in this component.[3]

The pub is specialized for particular purposes by creating derivatives with additional constraints. For example, rules to describe the relationship between lights and the switches that control them could be:

```
slp: #pub & { _trgt: "light", arg: "on"|"off" }

swp: slp & { _typ: "cmd" } <= swCert
lip: slp & { _typ: "sts" } <= liCert

swCert: devCert & { _role: "switch" }
liCert: devCert & { _role: "light" }

devCert: /_net/_role/_roleId/_loc <= netCert
```

This creates `slp` (switch+light pub), an instance of #pub which is constrained to target "lights" and accepts an argument of either "on" or "off" from an application at run-time. The "switch" variant of slp, `swp`, can publish an *on* or *off* command and requires the publishing entity to have a "switch cert", swCert, which confirms it is authorized to act as a switch. Similarly, an entity can use the "light" pub, `lip`, to report its on/off status (typically after a state change) if it possesses a "light cert". (The operator '<=' is read '*is signed-by*'.) The remaining three lines define cert formats. Note the `_loc` tag is used in both the #pub and `devCert` definitions. Tag names starting with a "_" are constrained to have the same value everywhere they are used in a signing chain. This means the pub builder uses the devCert's `_loc` to set a pub's `_loc` and the validator will always check that an incoming pub's `_loc` matches the signing `devCert`'s. This mechanism supports full Attribute-Based Access Control [25] using signing chains with suitably hardened cert linkage (§ 3.1) as a Policy Information Point.

To understand how the schema works, say a kitchen sink light on Alice's IoT network was enrolled with devCert

```
/alice/light/42/sink
```

and a switch near it with

```
/alice/switch/11/sink
```

The schema tells the light's application to subscribe to publications with prefix

```
/alice/light/cmd/sink
```

When the switch "on" button is pressed, its application calls

```
publish("arg","on")
```

resulting in the multicasted publication

```
/alice/light/cmd/sink/on/0x0123456789abcdef
```

which matches the light's subscription and turns on the light.

More complex signing chains are discussed in § 4 for *dctIoT*.

In addition to the specifications for the structure of publications and their signing certs, Versec trust schemas include specifications for the namespace prefix and cryptographic signing/validation of both the publications and the syncData that carry them on the "wire" (used to denote media packets), e.g.:

```
#pubPrefix: _domain
#pubValidator: "EdDSA"
#wirePrefix: _ndnprefix/_domain & { _ndnprefix: "localnet" }
#wireValidator: "AEAD"
```

These lines specify a cryptographic signing and validation algorithm from DCT's run-time library (§ 5.1). The EdDSA algorithm is specified for publications, requiring signing by a private key and validation with its public key. Publications are contained within a syncData (an NDN Data) using AEAD [1] to provide confidentiality via AES-128 symmetric encryption. The run-time library automatically handles cert distribution and, when confidentially is specified, automatic generation, rotation and secure distribution of the symmetric group key (see § 5.1 ***distributors***). Versec explicitly prohibits applications from use of integrity-only signing as this does not enforce the trust zone.

3 IDENTITY BUNDLES

To participate in a trust zone, an entity needs a verifiable copy of the zone's trust anchor, signed trust schema and an identity signing cert (public key cert that includes private key) complete with its signing chain (public certs only), which always has the zone's trust anchor at the root. *Only* an entity's own signing cert is included as the public signing certs of others are obtained and validated (using the common trust schema and anchor) at run-time through use of a *syncps*-based cert collection (§ 5). As well-formed certificates and identity bundle deployment are critical elements of the architecture, this section describes certificate requirements and the construction and installation of an identity bundle. DCT includes helper functions to create certs and bundles. Current and

[3] The *syncps* transport (§ 5.1) uses this as part of its replay protection machinery.

emergent industry best practices provide a range of approaches for secure installation and update of private keys; as much as possible our deployment plans exploit those (§ 3.2 and § 5.3).

3.1 DCT Certificates (Certs)

Employing a SDSI-like architecture with a single, local, trust root cert (trust anchor) simplifies trust management and avoids the well-known certificate authority (CA) federation and delegation issues (there are no CAs; just the local trust anchor). This and other weaknesses of the X.509 architecture are summarized at [10] from original references including [30, 37]. DCT certs are generated and signed locally (for the house network) so there is no reason to aggregate a plethora of unrelated claims into one cert (avoiding the Aggregation problem[10]). DCT cert's one and only Subject Name is the name of the NDN Data object that contains the cert as its content and neither are allowed to contain *any* optional information or extensions. All certificates are created with a lifetime; local production means cert lifetimes can be just as long as necessary (as recommended in [29]) so there's no need for the code burden and increased attack surface associated with certificate revocation lists (CRLs) or use of on-line certificate status protocol (OSCP). Key roles that require longer lifetimes, like device keys, get new certs before the current ones expire and distribute those through DCT. If there is a need to exclude previously authorized entities from the trust zone, there are a variety of options. The most expedient is via use of the AEAD wireValidator by ensuring that the group key maker(s) for a trust zone (§ 5.1) exclude that entity from subsequent symmetric key distributions. It is also possible to distribute a new trust schema and signing identities (without changing the trust anchor) using remote attestation via the TPM.

Signing certs are accompanied by their full signing chains, both when installed in identity bundles and when an entity publishes its signing identity (without the private key, of course) to the cert collection. Every signing chain has the same trust anchor at its root. Without the entire chain, a signer's right to issue publications cannot be validated. DCT cert validation behaves as if the rights and properties of every cert in the signing chain were concatenated for each publication the chain signs. For this model to be well founded, each cert's "key locator" must *uniquely* identify the cert that actually signed it. This property ensures that each locator resolves to one and only one signing chain. The literature shows that name-based locators like X.509's "Authority Key Identifier" and "Issuer" are open to substitution attacks [11, 32, 47]. DCT's key locator is a *thumbprint*, a SHA256 hash of the *entire* Signer's NDN Data packet (name, content, key locator, and signature), ensuring that each locator resolves to one and only one cert and signing chain.

3.2 Making and installing identity bundles

The DCT toolkit provides utilities that assist in creation of identity bundles, including one that creates a trust anchor [61]. A trust schema that passes *schemaCompile* checking can be output in binary format, then embedded in a certificate signed by the trust anchor. The *make_cert* utility can create all the required certs and *make_bundle* will make a bundle from given certificates. Bundles contain, in order, the trust anchor, the trust schema, and the signing

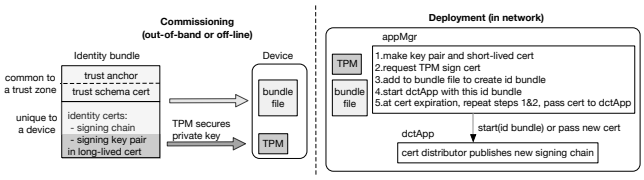

Figure 3: Deploying identity bundles

chain for the entity (private signing keys are included *only* for a signing identity). As discussed earlier in this section, bundles do not contain signing certs of other devices; new trust zone members are added dynamically. The private key(s) of the bundle will be secured using the Trusted Platform Module, the best current practice in IoT [18, 34–36, 40, 68, 72, 74]. Identity bundles are intended to be installed securely when a device is first commissioned (out-of-band) for a network: 1) the bundle, with the exception of private signing key(s) is installed as a file and 2) private key(s) are installed via the TPM. fig. 3 illustrates this on the left hand side. An authorized configurer (e.g., a homeowner for home IoT) adding a new device uses TPM tools to secure the private signing key(s) and installs the rest of the bundle file in known location before deploying the device in the network.

For any actual deployment, good key hygiene using best current practices must be followed e.g., [45, 58]. For *dctIoT* deployment, a small application manager (*appMgr*) with will be programmed for two specific purposes. First, it is registered with a supervisor [17] (or similar process control) for its own (re)start to serve as a bootstrap for *dctApp*. Second, it has access to the TPM functions and the ability to create "short-lived" (~hours to several days) public/private key pair(s) that will be signed by the private key of the installed identity cert using the TPM, which will happen at (re)start and at the periodicity of the cert lifetime. Since the signing happens via requests to the TPM, the key cannot be exfiltrated. At (re)start, the signing cert is added to the stored bundle file (the entire chain should be rechecked for validity) and passed to *dctApp* as it is invoked. For periodic signing cert updates, only the new cert needs to be passed to the already running *dctApp* as the rest of the bundle does not change. *dctApp* uses the library's cert distributor to publish its new certs. (See right-hand side of fig. 3.)

In our development environment, an identity bundle is given directly to an application via the command line and the TPM is not used. Sample applications in the repository [60] are written for the development environment. All DCT certs have a freshness period and an validity period which currently default to one hour and one year, respectively. Trust anchors, trust schema, and the TPM-secured signing identity are "long-lived" certs where that lifetime should be application- and deployment-specific, but is expected to be ~year(s). Another key pair is made just before a certificate expires so the new certificates can be distributed. Discussion of updating trust schemas and other certificates over the deployed network is deferred to § 5.3. Changing the trust anchor is considered a re-commissioning and is *not* done over the network.

4 A HOME IOT TRUST SCHEMA

All the home IoT platforms we surveyed[4] use a structured abstract profile to represent devices and their *capabilities*. Devices can have multiple capabilities, e.g., a lamp has a switch and a light. Common devices like motion-sensing lamp fixtures, doorbell cameras, and electronic door handlesets might include capabilities like: light, sensor, switch, lock. Capabilities have *attributes* which represent state information and commands which are the ways in which the attributes can be controlled. Attributes are as simple as "on" or "off" or might be a range of values, as for a temperature sensor. Commands can toggle an attribute or set an attribute. *Events* let a capability report a state transition, e.g., from off to on or into a particular temperature range. Most IoT platforms represent their capabilities and devices in standard device profiles, well-defined structured objects. Individual IoT platforms have a library of standard device profiles that specify common capabilities (e.g., switches, cameras, locks) and provide a template for devices with custom capabilities. Although these concepts are named differently across IoT platforms, the functionality is quite similar and there are implementations that translate, or map, between platform representations (particularly in Home Assistant). The unification of these disparate but similar data models is an active area of standards work, e.g., the JSON representations of One Data Model, W3C IoT Community Group, or Matter [3, 15, 16]. Though mappings exist, a single standard library would be particularly helpful for (semi)automated trust schema. In the meantime, *dctIoT* publications are constructed to accommodate functionality of the major platforms.

Though IoT devices have well-specified capabilities *there is no enforcement mechanism focused on restricting devices to only act within those capabilities*. When security relies on a network perimeter's access control or on encrypting the channel between two communicating devices, the approach can fail if a single internal device is compromised, leaving networks open to exploits based on hijacking a single lightbulb, camera, or other device and issuing commands that should never come from such a device. It is generally difficult or impossible to constrain an authorized user to functions relevant to their role, e.g., homeowner, child or guest. Descriptive models of devices and capabilities couple with desired policy rules can be used in a Versec trust schema, *dctIoT.trust*, and deployed in DCT-based applications to address these issues.

4.1 Specifying dctIoT publications

Following [55], *dctIoT* Names are organized hierarchically, starting with a component that identifies the particular network (e.g., *houseNet*) and end with three components that are used for replay prevention, segmentation/reassembly, and identification. *msgID* is a unique ID for the application level message, *sCnt* indicates whether this is a single publication message and if not gives its k out of n segment value, and *msgTS* is the timestamp when the message was created. The basic Name format is:

<networkID><...><msgID><sCnt><msgTS>

[4]SmartThings, Web of Things, Zigbee Clusters, Zwave, BTmesh, and HomeAssistant. These generally use different names for the same concepts. The description here is most similar to SmartThings [2] but we attempt a general format that can map to all major approaches.

Object description	Components

Device name/ID: {
(opt: component, entity, endpoint ID)
Capability name:
attributes:{
[.]
}
commands: {
[.]
}
events: {
[.]
}
}
}
	<location> or <issuer>
	<location>
	<capability>
	<topic>
	<arguments>
	
	<topic>
	<arguments>
	
	<topic>
	<arguments>

<capability><topic><location><arguments><issuer>	

Figure 4: Map IoT object structure to Name components

where the components that make up <...> are specific to this application. Figure 4 shows a structured IoT object description of what IoT applications need to communicate about. A device can report on the value of its capability attributes via arguments, a particular capability can accept commands with particular attributes, and a capability can report on such events as a temperature exceeding a threshold. Publication name components make these capabilities, actions, and roles visible to ensure that devices only report on attributes that they can observe (normally, their own capability attributes) and that commands only come from devices and roles or capabilities with permission (e.g., the homeowner can use a phone to send a command to a lock but a doorbell camera can never publish a command). Publication names both *communicate* application information and *restrict* communications to trust schema permitted roles and capabilities so that individual devices reject illegal publications.

The application-specific portion of *dctIoT* Names is organized to reflect the data-centric pub/sub approach where synchronized collections contain sets of Data of the same topic (facilitating a publisher-agnostic many-to-many model) rather than a set of Data from the same publisher (a one-to-many model). Devices need to subscribe to and publish in collections that are distinguished by their capabilities, so *capability* is the first component of this application-specific portion. The next component is *topic* which indicates what aspect of the capability definition is addressed in this publication (**attribute**, **command**, or **event**). The next component is used to distinguish which capability and topic the publication concerns, denoted by *location*. Location has a broad definition; it can indicate a room or floor in a building, a particular device or class of devices, a network, or a designator **local** to keep publications from leaving the device. The *arguments* component is used for additional information that may be required for this topic.[5] Finally, a component for the *issuer* of the publication provides a field that can contain the role identifier of the originating entity, the device where the publication originated, or a combination of the two. The right hand column of fig. 4 lists publication components opposite the object elements that appear or may appear in them. The portion of the publication Name (<...> above) specific to the dctIoT is at the bottom.

[5]Future versions of the trust schema language and compiler will permit structures in components.

4.2 dctIoT trust schema

Once defined, a publication format is used in a Versec trust schema, e.g., the *dctIoT* schema of fig. 5. Three distinct publication formats are defined (denoted with a leading "#"): Report, tagCommand, and prgCommand. Components given *without* a leading "_" are parameters supplied when the run-time library is called to construct a publication. Components *with* a leading "_" must have the same value everywhere in the signing chain so are used to propagate information from certs to publications such as the _cap (capability) field in the Report; i.e., its _cap may be obtained from a cert anywhere in the signing cert and _cap must be the same throughout the signing chain. The origin component specifies a function, sysId(), that will be called to set the component when a particular publication is constructed.

In the example, there are two capabilities, *light* and *switch,* and each has two basic states: "on" or "off." Publications derived from Report can give the state for light or switch (*lsState*, requiring signature by a cert with capability *light* or *switch*) or report the event of a light changing state, either off to on, or on to off (*lightEvent*, requiring a a cert with capability *light*). The specification for Report ensures that a capability can only publish events and attributes about itself by making use of its unique devId from its device cert. The devType and devTag are not unique; devType can be used to specify a type of device ("deskLamp" or "ledStrip") while devTag can be used to set up designators for grouping devices that are not necessarily similar, e.g., "livingRoom" or "sink".

This trust schema limits lights to signing publications with topic *event* or *attribute* which means that a *command* from a light is illegal. The rules say that a switch can publish the current value of its single attribute (on or off) and can publish a command for the light capability with location indicator that is taken from the devTag field in its capability certificate, allowing it to be paired with a light that has been configured with the same devTag, e.g., set devTag to be a single location like "sink" or a room like "kitchen." A prgCommand can have any location component and is used to create lightOwnCmd that lets an ownerCert sign any (properly formatted) publication with the light capability. This is a role certificate that an owner might have on a mobile device to remotely control lights. A more restrictive role certificate could limit the capabilities or locations in the format of legal publications. The wireValidator and pubValidator lines specify that publications cryptographically signed and validated using an EdDSA algorithm. syncData packets are AEAD signed and encrypted for confidentiality.

Each device on-boarded to *houseNet* has innate capabilities and is configured with certificates for those capabilities, signed by the device's cert, authorized for use in this network.

schemaCompile is used to check dctIoT.trust for missing certs and syntax issues and to make a binary version of the trust schema if it passes checks. The diagnostic output (with a verbose option) is shown annotated in fig. 6. Example schemas for dctIoT [60] have binary schema output sizes of 300-600 bytes and schema certificates of fewer than 900 bytes. Identity bundles are ~1400 bytes.

```
_net: "houseNet"
// Publication definition
#Report: _net/_cap/_devId/topic/args/_origin/mID/sCnt/mts & { _origin: sysId() }
switchCert: capabilityCert & { _cap: "switch" }
lightCert: capabilityCert & { _cap: "light" }
lsState: #Report & { topic: "attribute", args: "on"|"off" } <= switchCert | lightCert
lightEvent: #Report & { topic: "event", args: "on2off"|"off2on" } <= lightCert
// For a paired by _devTag (e.g., "sink") switch and light
#tagCommand: _net/cap/_devTag/topic/args/_origin/mID/sCnt/mts & {
    topic: "command", _origin: sysId()
}
lightTagCmd: #tagCommand & { cap: "light", args: "on"|"off" } <= switchCert
// A programatic command can go to any 'loc'
#prgCommand: _net/cap/loc/topic/args/_origin/mID/sCnt/mts & {
    topic: "command", _origin: sysId()
}
lightOwnCmd: #prgCommand & { cap: "light", args: "on"|"off"|"report" } <= ownerCert
capabilityCert: _net/_devTag/_cap/_keyinfo <= deviceCert
deviceCert: _net/_devType/_devId/_keyinfo <= configCert
ownerCert: roleCert & { _role: "owner" }
roleCert: _net/_role/_roleId/_keyinfo <= netCert
configCert: _net/"config"/confId/_keyinfo <= netCert
netCert: _net/_keyinfo
// Publication prefix and validator type
#pubPrefix: _net
#pubValidator: "EdDSA"
// Prefix used for syncData (NDN Interest/Data)
#wirePrefix: _ndnprefix/_net & { _ndnprefix: "localnet" }
#wireValidator: "AEAD"
// components are KEY, keyID, issuerID, and version
_keyinfo: "KEY"/_/"dct"/_
```

Figure 5: dctIoT.trust example

5 IMPLEMENTATION

DCT provides the library used to implement *dctIoT*'s Versec-enabled transport (see fig. 2). An application-specific shim hides details of the transport and exposes only necessary information to the application, an approach we first employed in [55, 56] and have also used in [42]. *dctIoT*'s shim implements a message-based pub/sub (*mbps*) API loosely based on the widely-used (in IoT) MQTT application protocol. This provides a familiar interface for developers and for adapting applications as well as being well-suited to ICN communications. In *mbps*, as in MQTT, applications communicate by publishing and subscribing to hierarchically structured topics where publications carry commands, status and signal events. Publishers do not have specific knowledge of subscribers and publications are made available asynchronously. fig. 7 shows the distinct modules used to implement *dctIoT*, its *mbps* shim (the only custom module) and DCT library modules which *mbps* uses to package and unpackage application messages into publications, to ensure that invalid messages are not passed to applications and no invalid publications are formed. Publications are added to or received from a collection via the Sync protocol *syncps* [55] which packages them into (and retrieves them from) syncData for the NDN forwarder (see [60]).

5.1 DCT library

Modules of DCT, grouped by category, with relevant information:

syncps in its original version [55] implemented a Sync collection providing unacknowledged delivery and any-to-any communications. Each *syncps* announces the publications it currently has in its collection by sending an Interest containing a Difference Digest [31, 51]. Difference Digests solve the multi-party set-difference

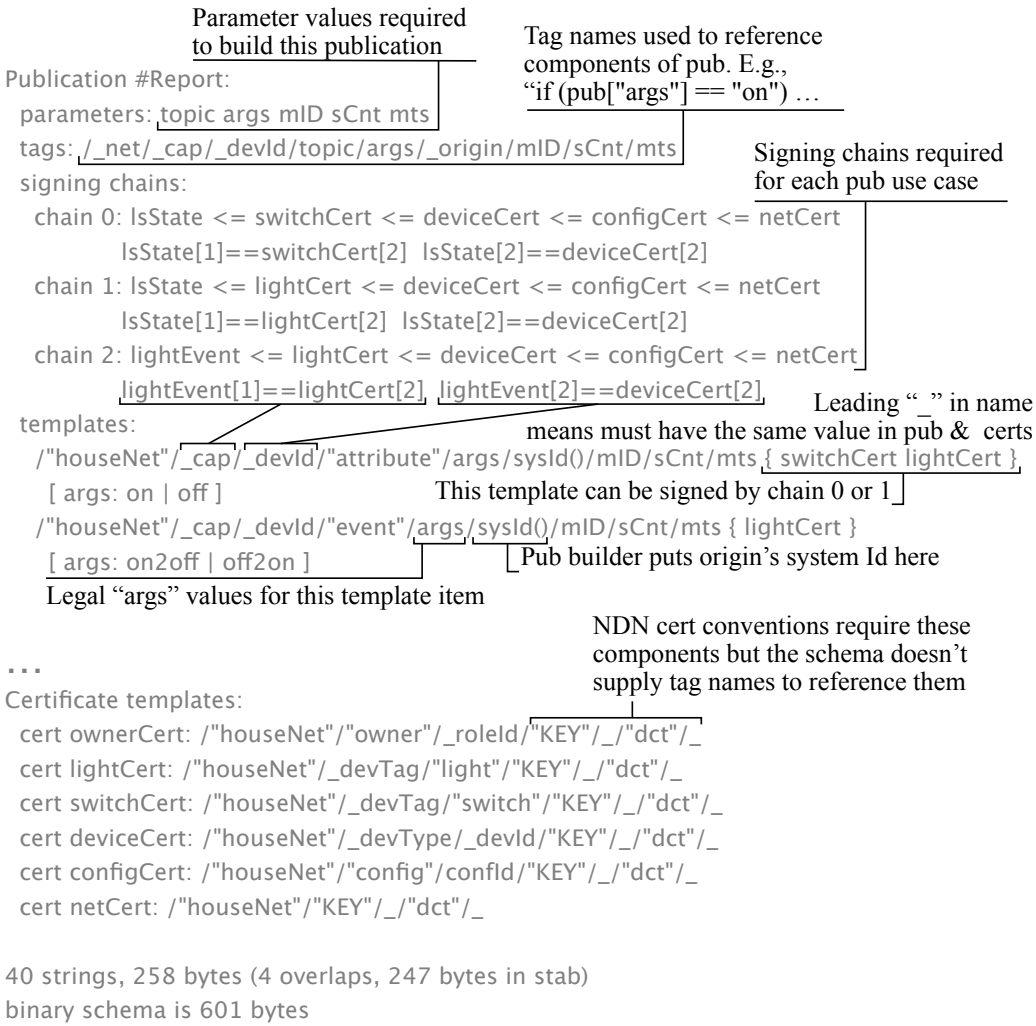

Figure 6: Annotated verbose output of schemaCompile for dctIoT.trust

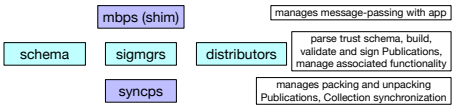

Figure 7: Versec-enabled transport for *dctIoT*

problem efficiently without the use of prior context and with communication proportional to the size of the difference between the sets being compared. *syncps* has to manage active and inactive publications to know when and if to communicate them to peers and subscribers, but it knows nothing about the format or semantics of publications. Upcalls from *syncps* to other modules provide validation and expiration information for publications as well as validation and signing of syncData. DCT's version of *syncps* has some improvements and bug fixes and adds the ability to acknowledge when publications have reached their collection if an optional callback handler is provided. A single instance of a syncps manages a single collection.

schema modules read and parse a binary trust schema, provide certificate management to validate and locally store required certificates, and build and validate publications based on the trust schema. Validation and signing makes use of *sigmgr* modules and certificate management uses *distributors* modules.

signature managers (sigmgrs) implement any required signing and validation, utilizing libsodium [5] for cryptographic functions. *sigmgrs* implement integrity signing (based on both RFC7693 and SHA256), key-based signing (EdDSA), and symmetric key-based encryption (AEAD, additional encrypted authenticated data, only used on syncData packets). Additional approaches to certifying trust can be added by implementing a new *sigmgr*.

distributors maintain synchronized collections of certificates and keys for a trust zone (thus each distributor has its own *syncps*).

One type of distributor maintains the collection of signing certificate chains, where entities publish when they join the trust zone or are given new certs. The signing certs of all trust zone members are located in the collection, obviating the need for external on-line certificate repositories. The trust anchor and signing chains in the trust schema are used to validate all received certificates. When an application starts, it publishes its signing chain and waits for confirmation that at least one other entity has acquired it (done through syncps) before the application can start communicating.

Another distributor type periodically makes and securely distributes symmetric keys for all the signing identities (certs) found in the signing cert collection and is only invoked when AEAD is used to enforce data privacy. When used, it starts after signing cert publication is confirmed and before application communications begins. Certain entities (as few as one or as many as all) are designated as having the potential to make these keys and, if more than one, initially contend to become the group key maker[6] after which that entity periodically makes a new key and packages it into a publication where the key is encrypted multiple times, using the public key from all the valid signing certs, i.e., there is an encrypted version of the symmetric key for each member entity. Members of the trust zone decrypt their version and use with the AEAD sigmgr.

5.2 mbps: message-based pub/sub shim

Pub/sub has over 30 years of history but has become more important in recent years as it is applicable to a range of modern problems from data center tasks to IoT; with this potential in mind, *mbps* was designed to be more general-purpose than our previous shims. In MQTT, assured delivery semantics[7] are outside the basic pub/sub primitive; specific styles of QoS, approaches to long-term storage, and other functionality are layered on top. *mbps* has similar goals, providing a simple API that hides network layer and security details and offers two levels of message QoS: a default unacknowledged delivery and a confirmation that the publication has reached at least one other member of the collection. Applications create an *mbps* instance and pass it the identity bundle filename. Applications can use the following *mbps* methods:

connect(successCB, <opt>failureCB): Performs set up (if any) necessary to allow communications to start. The method of MQTT that creates a connection to a Broker is "hijacked" here to connect to the collection used for communications (e.g., signing key distribution is carried out). Upon success, the work of the application starts.

publish(msg, args, <opt>confCB): Publishes the given message content and returns a unique message ID. If a confirmation callback is included, *mbps* invokes confCB with an indicator of success or failure of the specific message's publication.

subscribe(handler): subscribes to all the topics in this *mbps* collection (set by the pubPrefix in the designated trust schema). A received message's underlying publication(s) is validated before the handler is invoked.

run(): once application set up is finished, this turns over control to the transport.

The API simplicity is shown in this application snippet:

```
void msgPubr(mbps &cm, std::vector<uint8_t>& toSnd) {
        ... //prepare arguments
        cm.publish(toSnd, a); //load arguments in a
}
static void msgRecv(mbps &cm,
        std::vector<uint8_t>& msg, const msgArgs& a) {
        ... //do something with msg
}
int main(int argc, char* argv[])
{
    mbps cm(argv[optind]); //make cm, pass identity bundle
    cm.connect(
            [&cm]() { //successful connect
                cm.subscribe(msgRecv);
                                ... //prepare toSnd
                msgPubr(cm, toSnd); });
    cm.run();
}
```

5.3 Security information and implications

An important feature of this architecture is that the security used for a particular application can be changed by creating a new trust schema and installing a new identity bundle with *no* changes to application binary. The examples at [60] can be used to explore this feature through comparing its three trust schemas: *mbps0.trust* ensures publications and syncData are cryptographically signed with an entity's signing key (EdDSA), *mbps1.trust* adds the publication structural role-based requirements (as in fig. 5), and *mbps2.trust* adds AEAD encryption to the syncData (*mbps2.trust* differs by changing only the wireValidator of *mbps1.trust* to AEAD). This illustrates that Versec can initially be used solely to enforce signing and certificate distribution in a trust zone and later progresses to more fine-grained access control.

Every publication and syncData (i.e., *all* communications) must be signed by a cert that proves, according to the trust schema, the publishing entity is allowed to say what it said. The trust schema and every cert valid under the schema are signed by the trust anchor, directly or through a signing chain. Every entity is enrolled by giving it an appropriate signing cert (e.g., ownerCert or deviceCert and capabilityCert(s) in fig. 5 with private key) and its signing chain. Enrollment and signing chains are structured so as to prevent privilege escalation. For example, in fig. 5 devices are given the private keys of their capabilities but not the private key of their device cert since that would allow them to expand their capabilities. The signing chains are also designed so that while the ephemeral private keys that sign publications generally need to be kept in the publishing process's memory, higher value private keys that sign other keys can be kept in and used from a secure enclave like a TPM to prevent exfiltration.

The bootstrap identity bundle gives each entity the trust anchor, the schema, and the full signing chain of its signing identity cert(s) in a few kilobytes. These give every entity the means to prove to itself and any other member of its trust zone that it is validly configured. DCT's model requires that the original publisher of any object ensure that all certs and schemas needed to validate an

[6] Rudimentary, though functional; more sophisticated approaches are possible.
[7] beyond its "at most once," "at least once," and "exactly once" message QoS

object have been published *before* the object is published. This is enforced by requiring that received syncData or publications with any missing signing dependencies *must* be considered unverifiable and silently discarded. This is in contrast to security architectures that store data packets and query networks for their signing keys, a clear attack opportunity that grows out of the "be liberal in what you accept" philosophy of today's Internet. We adopt the Language-Based Security [14] field of work recommendation that protocols "be definite in what you accept" to ensure the integrity of trust zones. DCT's run-time library handles this automatically by putting the entity's signing cert and bootstrap configuration information into an application-local cert collection and using the cert distributor (§ 5.1) to sync that collection with all the peer entities: initial signing chain publication is completed at start up and thereafter the cert distributor keeps the collection in sync. When a new, properly configured device (whose type is defined in the trust schema) is attached to the network to join the trust zone, it is sufficient for it to join the the signing certificate collection.

Signing certificates used by applications should be updated periodically (as described in § 3.2), then published through the cert distributor without the need for application restart. Trust schemas *may* change with addition of new types of devices or users, so an over-the-air update will be explored. The long-term private signing cert whose key stored by the TPM should be updated using existing and emergent best practice [38, 40]. However acquired, a validated new trust schema must be updated in the identity bundle and the *appMgr* (§ 3.2) notified to restart the application.

5.4 Performance experience

Security architectures for IoT devices have typically been concerned with the lower resources of such devices when contrasted to general purpose computers. Today's home IoT devices are not the incapable elements (or "motes") of sensor networks. In particular, the processors in lightbulbs have been used to take over entire home networks [23] and thus, it is both reasonable and prudent to regard home IoT devices as capable computing elements.

Configuration data is relatively compact: The certificates created for [60] have a median size of 256 bytes (mean 254, SD 7.4, min 239, max 260) with bundle median and mean sizes of 1326 bytes (SD 8.4, min 1315, ma 1337) and 823-828 byte signed schemas. The substantially more complex schema of [42] has a median cert size of 247 bytes (mean 251, SD 8.5, min 235, max 262) and bundle median and mean of 2307 bytes (SD 1.2, min 2307, max 2310) and a 1253 byte schema. Measured performance (over 1000 runs) for *app2* of [60] using EdDSA signing of publications and AEAD encryption of syncData is summarized in table 1.

One-way delay IPv6 multicast UDP message delivery time, measured from publication creation time to a subscribing application's reception (i.e., full DCT stack plus network traversal).

Request-Response IPv6 multicast UDP message delivery time, measured from the Request publication creation time to the Response publication's reception by the requesting application (i.e., full DCT stack plus network traversal).

6 RELATED WORK

Two important building blocks of our architecture are a shared trust schema that is integrated into the communications model, allowing fine-grained role permissions, and a Sync protocol, *syncps*, that uses the NDN Interest and Data primitives to enable collection-based communications. These building blocks do not appear in other ICN security architectures, but certain elements of our approach are common to other work.

The Manufacturer Usage Description (MUD) work [33, 43] shows that IoT device characterization can be used to describe communications requirements which can in turn be used by network elements to constrain how they communicate. The implementation approach of [33] is focused on in-network access control of unchanged, legacy devices where enforcement is via access control lists in network elements like routers, switches and firewalls, a traditional perimeter-based defense. In contrast, our approach assumes devices that can deploy trust schema-based applications and creates a zero trust community on any physical network along with easily deployed encryption and the communication efficiency advantages of ICN.

The process of enrolling a device into a network by provisioning it with an initial secret and identity in the form of public-private key pair and using this information to securely onboard a device to a network has a long history, including in ICN [27, 66]. The approach we outlined in § 3.2 is just one of many ways to provision, making use of current best practice. Key differences from these earlier approaches is that a *dctIoT* identity constrains actions rather than simply conveying membership. In [27], onboarding is a one-to-one transaction between a device and a controller or a gateway, while *dctIoT* publishes and validates signing chains in a distributed collection to onboard, a many-to-many transaction. The practice of encrypting a group symmetric key with individual public keys is widely used in ICN, e.g., [27, 48, 66, 76]. Marxer [48]suggests publishing new group encryption keys in "contexts" where the new key is encrypted by public keys of the members which is similar to *dctIoT*'s group key publication, signed by the keymaker, that carries a list of the current group key encrypted by each member's public key.

The NDN-based building management system of [66] is also designed for an environment that does not rely on physical network segregation for security. At installation, devices get their specific namespace, a trusted public key (trust anchor) and an identity (key pair) and possibly symmetric key(s). Its trust model has some similarities but relies on a manager daemon separate from the applications to enforce access via lists of namespaces and permitted user identities and identities with permitted namespaces are external to the communication model of the applications. Lacking an integrated trust schema where validity (rather than permission) is expressed in capabilities and roles that can be extracted from certificates, these lists must be updated with any new identities and propagated to all enforcement points.

A very different architectural use of schemas in ICN, Marxer's differential access to stored data [48], proposes using a schema in a capability document and capability lists defining what principles (or entities) can have access to certain data.

CPU	(all times in μs)					
	One-way delay			Request-Response		
	median	min	75%	median	min	75%
2015 2GHz Embedded Nehalem	850	249	871	1390	456	1430
2012 3.5GHz E5 Xeon	315	265	333	604	426	649
2014 4GHz Core-i7	280	229	287	585	372	606

Table 1: Performance on various (older) CPUs

7 SUMMARY AND FURTHER WORK

This paper presents an architecture and toolkit for creating ICN Zero Trust zones using Versec, a new approach to trust schemas and their usage. Trust schemas are distributed as certificates and used by an efficient run-time policy enforcement layer integrated with application communications transport, thus a rigid set of conventions about "who gets to do what to which" is enforced while making secure application writing easier. This approach is applied to home IoT to create a secure platform that presents a simple pub/sub message interface to applications. An initial version of the platform has been coded; refinements and additions are in progress, including more expressiveness for the trust schema language, optimizations to streamline communications, and approaches to connecting remote trust zones. Creating trust schema templates for a wide range of IoT devices and translating a large number of IoT object definitions into *dctIoT* names represents a large undertaking and events may lead to some consolidation in IoT data models, so we intend to start "narrow" with a fully functional prototype containing a few useful, representative elements.

Using ICN security features to solve real-world problems is in its infancy. This approach is a significant step toward making trust specifications more usable, straightforward and secure while making information-centric application creation easier. Though focused on IoT, at least some aspects of this approach should be applicable to areas such as vehicular communications, military and first responder communications, and a host of energy related applications.[8] The toolkit is open source precisely in order to aid the work of others. Open areas that could aid in widespread usability of this approach include automatic mapping of IoT (or other application) structured descriptions into templates for *mbps* and Versec and other aids to expressing trust rules in Versec. In particular, a template for home IoT (or another application) could be used to configure a local trust anchor and different roles and permissions as desired by the administrator. Our group key distributor has a very simple algorithm for electing a key maker among eligible entities of the trust zone; more sophisticated versions might include fault-tolerance. Novel methods for securely distributing and installing trust schema updates to devices may be possible. The *appMgr* and approach described in § 3.2 have not yet been implemented and there certainly may be more innovative approaches possible. Although *mbps* implements a general pub/sub API, other APIs for Versec-enable transports might be possible and useful for other communications models, e.g., streaming, conferencing, virtual reality.

[8] DCT is being used in a secure transport for ongoing research under a DOE funded project.

ACKNOWLEDGMENTS

Van Jacobson provided the Versec approach and numerous invaluable conversations. Randy King of Operant Networks has presented a host of interesting problems that stimulated portions of this work. The shepherds provided useful feedback on improving this paper.

REFERENCES

[1] [n.d.]. Authenticated Encryption. https://en.wikipedia.org/wiki/Authenticated_encryption
[2] [n.d.]. SmartThings API (v1.0-PREVIEW). https://smartthings.developer.samsung.com/docs/api-ref/st-api.html#operation/listCapabilities
[3] [n.d.]. *Welcome to iotschema.org.* http://iotschema.org/
[4] 2019. Cybersecurity. *The Bridge* 49 (2019).
[5] 2020. https://doc.libsodium.org/
[6] 2020. CUE history and principles. https://cuelang.org/docs/about/
[7] 2020. CUE Tutorials. https://cuelang.org/docs/tutorials/
[8] 2020. Implementing CUE. https://github.com/cue-lang/cue/blob/master/doc/ref/impl.md
[9] 2021. https://staceyoniot.com/you-can-learn-much-about-matter-from-the-project-chip-github-repo/
[10] 2021. https://en.wikipedia.org/wiki/X.509#Security
[11] 2021. Additional authenticated data guide. https://cloud.google.com/kms/docs/additional-authenticated-data#confused_deputy_attack_example
[12] 2021. Annotated Schema Example - DNMP. https://github.com/pollere/DCT/blob/main/versec/dnmpExample.md
[13] 2021. Graph Theory – Subsumption and unification. https://en.wikipedia.org/wiki/Graph_theory#Subsumption_and_unification
[14] 2021. LANGSEC: Language-theoretic Security "The View from the Tower of Babel". http://langsec.org
[15] 2021. Matter is the foundation for connected things. https://buildwithmatter.com/
[16] 2021. One Data Model. https://onedm.org/
[17] 2021. Supervisor: A Process Control System. http://supervisord.org/
[18] 2021. TPM attestation. https://docs.microsoft.com/en-us/azure/iot-dps/concepts-tpm-attestation
[19] 2021. The Versec Trust Schema Compiler. https://github.com/pollere/DCT/blob/main/versec/language.md
[20] Martín Abadi and Boon Thau Loo. 2007. Towards a Declarative Language and System for Secure Networking. In *Third International Workshop on Networking Meets Databases, NetDB'07, Cambridge, MA, USA, April 10, 2007*, Brian Cooper and Nick Feamster (Eds.). USENIX Association. https://www.usenix.org/conference/netdb-07/towards-declarative-language-and-system-secure-networking
[21] Alexander Afanasyev, J Alex Halderman, Scott Ruoti, Kent Seamons, Yingdi Yu, Daniel Zappala, and Lixia Zhang. 2016. Content-based security for the web. In *Proceedings of the 2016 New Security Paradigms Workshop*. 49–60.
[22] Matt Blaze, Joan Feigenbaum, and Jack Lacy. 1996. Decentralized Trust Management. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 164–173. https://doi.org/10.1109/SECPRI.1996.502679
[23] Check Point Software Technologies LTD. 2020. *The Dark Side of Smart Lighting: Check Point Research Shows How Business and Home Networks Can Be Hacked from a Lightbulb.* https://blog.checkpoint.com/2020/02/05/the-dark-side-of-smart-lighting-check-point-research-shows-how-business-and-home-networks-can-be-hacked-from-a-lightbulb/
[24] S. Chokhani. 1996. A Security Flaw in the X.509 Standard. https://csrc.nist.gov/csrc/media/publications/conference-paper/1996/10/22/proceedings-of-the-19th-nissc-1996/documents/paper075/paper.pdf
[25] Chung, David Ferraiolo, David Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. 2019. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. https://doi.org/10.6028/NIST.SP.800-162
[26] Home Assistant Community. 2018. Smarter SmartThings with MQTT and Home Assistant. https://community.home-assistant.io/t/smarter-smartthings-with-mqtt-and-home-assistant/42493

[27] A. Compangno, M. Conti, and R. Droms. 2016. OnboardICNg: a Secure Protocol for On-boarding IoT Devices in ICN. In *Proceedings of the 2106 ACM ICN 2016 Conference*. ACM, 166–175.

[28] Brian A. Davey and Hilary A. Priestley. 2002. *Introduction to Lattices and Order, Second Edition*. Cambridge University Press. https://doi.org/10.1017/CBO9780511809088

[29] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. 1999. SPKI Certificate Theory. *RFC* 2693 (1999).

[30] Carl Ellison and Bruce Schneier. 2000. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. In *Computer Security Journal*, Vol. 16. 1–7.

[31] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. 2011. What's the difference?: efficient set reconciliation without prior context. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*. 218–229.

[32] C. Brubaker et al. 2014. Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations. *IEEE Security and Privacy* (November 2014), 114–129. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4232952/

[33] D. Dodson et al. 2021. *Securing Small-Business and Home Internet of Things (IoT) Devices: Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD).* Technical Report NIST.SP.1800-15. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-15.pdf

[34] N. Donovan et. al. [n.d.]. Device Management Requirements to Secure Enterprise IoT Edge Infrastructure. https://www.wwt.com/white-paper/device-management-requirements-to-secure-enterprise-iot-edge-infrastructure/

[35] Krishnan Ganapathy. [n.d.]. Using a Trusted Platform Module for endpoint device security in AWS IoT Greengrass. UsingaTrustedPlatformModuleforendpointdevicesecurityinAWSIoTGreengrass

[36] Paul Griffiths. 2020. *TPM 2.0 and Certificate-Based IoT Device Authentication.* Whitepaper. Global Sign. https://www.globalsign.com/en/resources/white-papers-ebooks/white-paper-tpm-20-and-certificate-based-iot-device-authentication

[37] Peter Gutmann. 2002. Everything you Never Wanted to Know about PKI but were Forced to Find Out. https://www.cs.auckland.ac.nz/~pgut001/pubs/pkitutorial.pdf

[38] Subir Halder, Amrita Ghosal, and Mauro Conti. 2020. Secure Over-The-Air Software Updates in Connected Vehicles: A Survey. *Computer Networks* 178 (06 2020), 107343. https://doi.org/10.1016/j.comnet.2020.107343

[39] J. Alex Halderman. 2016. NDN: A Security Perspective. https://www.nist.gov/system/files/documents/itl/antd/Alex_Halderman.pdf

[40] Luke Hinds. 2019. Keylime - An Open Source TPM Project for Remote Trust. https://www.youtube.com/watch?v=YtPsruEqGeY

[41] Van Jacobson. 2019. Watching NDN's Waist: How Simplicity Creates Innovation and Opportunity. http://ice-ar.named-data.net/meetings/2019-ICE-WEN-Annual/0-ICNWEN-Van-Keynote.pdf

[42] Randy King. 2020. Improving Existing Software Applications with a Practical and Secure NDN Publish/Subscribe Transport. (September 2020). https://www.nist.gov/video/ndn-community-meeting-day-2-part-2 video of talk (at 1:33 into Day 2 Part 2) at NDN Community Meeting 2020.

[43] Eliot Lear and Ralph Droms. 2019. Manufacturer Usage Description Specification. *RFC* 8520 (2019), 1–60.

[44] Ninghui Li, Benjamin Grosof, and Joan Feigenbaum. 2003. Delegation logic. *ACM Transactions on Information and System Security* 6 (02 2003), 128–171. https://doi.org/10.1145/605434.605438

[45] Lars Lydersen. 2019. Commissioning Methods for IoT. https://www.silabs.com/documents/public/presentations/ew-2019-iot-security-commissioning-methods-for-iot.pdf

[46] William R. Marczak, David Zook, Wenchao Zhou, Molham Aref, and Boon Thau Loo. 2009. Declarative Reconfigurable Trust Management. In *Fourth Biennial Conference on Innovative Data Systems Research, CIDR 2009, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*. www.cidrdb.org. http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_11.pdf

[47] M. Marlinspike. [n.d.]. More Tricks for Defeating SSL in Practice. http://2015.hack.lu/archive/2009/moxie-marlinspike-some_tricks_for_defeating_ssl_in_practice.pdf

[48] Claudio Marxer and Christian Tschudin. 2017. Schematized Access Control for Data Cubes and Trees. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (Berlin, Germany) *(ICN '17)*. Association for Computing Machinery, 170,175. https://doi.org/10.1145/3125719.3125736

[49] Lucas Mearian. 2020. *Amid privacy and security failures, digital IDs advance.* https://computerworld.com/article/3512108/frustration-over-growing-privacy-and-security-failures-advancing-self-sovereign-identities.html

[50] Metadium. 2019. *Introduction to Self-Sovereign Identity and Its 10 Guiding Principles.* https://medium.com/metadium/introduction-to-self-sovereign-identity-and-its-10-guiding-principles-97c1ba603872

[51] Michael Mitzenmacher and Rasmus Pagh. 2018. Simple multi-party set reconciliation. *Distributed Computing* 31, 6 (2018), 441–453.

[52] Virag Mody. 2020. *From Zero to Zero Trust.* https://gravitational.com/blog/zero-to-zero-trust/

[53] Falcon Momot, Sergey Bratus, Sven M. Hallberg, and Meredith L. Patterson. 2016. The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them. In *IEEE Cybersecurity Development, SecDev 2016, Boston, MA, USA, November 3-4, 2016*. IEEE Computer Society, 45–52. https://doi.org/10.1109/SecDev.2016.019

[54] Lily Hay Newman. 2019. Why Ring Doorbells Perfectly Exemplify the IoT Security Crisis. https://www.wired.com/story/ring-hacks-exemplify-iot-security-crisis/

[55] K. Nichols. 2019. Lessons Learned Building a Secure Network Measurement Framework Using Basic NDN. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*. ACM, 112–122.

[56] K. Nichols. 2019. *Lessons Learned Building a Secure Network Measurement Framework Using Basic NDN (slides).* https://pollere.net/Pdfdocs/LessonsLearned.pdf

[57] Charlie Osborne. 2019. Google's OpenTitan: A new open source silicon root of trust project debuts. https://www.zdnet.com/article/googles-opentitan-a-new-open-source-silicon-root-of-trust-project-debuts/

[58] owasp.org/www-project sidekek/. 2020. SideKEK README. https://github.com/OWASP/SideKEK

[59] Inc. Pollere. 2020. Data-Centric Toolkit (version 3.0). https://github.com/pollere/DCT

[60] Inc. Pollere. 2021. Message-Based Publish/Subscribe (MBPS). https://github.com/pollere/DCT/tree/main/examples/mbps

[61] Inc. Pollere. 2021. Tools for setting up certs for DCT-enabled applications. https://github.com/pollere/DCT/tree/main/tools

[62] R.L. Rivest and B.W. Lampson. 1996. *SDSI - A Simple Distributed Security Infrastructure.* Technical Report. MIT.

[63] E. Ronen, C. O'Flynn, A. Shamir, and A-O. Weingarten. 2017. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. IEEE Symposium on Security and Privacy.

[64] E. Ronen, C O'Flynn, A. Shamir, and A-O Weingarten. 2017. *IoT Goes Nuclear: Creating a ZigBee Chain Reaction (slides).* https://eyalro.net/pdf/IoTSP17.pdf

[65] Samsung. 2016. Samsung Announces Commercially Available IoT Cloud Platform to Deliver Interoperability Between Devices and Applications. https://news.samsung.com/us/samsung-announces-commercially-available-iot-cloud-platform\protect\discretionary{\char\hyphenchar\font}{}{}deliver-interoperability-devices-applications/

[66] Wentao Shang, Qiuhan Ding, Alessandro Marianantoni, Jeff Burke, and Lixia Zhang. 2014. Securing building management systems using named data networking. *IEEE Network* 28, 3 (2014), 50–56.

[67] Diana K. Smetters and Van Jacobson. 2009. *Securing Network Content.* Technical Report. PARC. https://named-data.net/wp-content/uploads/securing-network-content-tr.pdf

[68] Tony Truong. [n.d.]. How to Use the TPM to Secure Your IoT/Device Data. https://tonytruong.net/how-to-use-the-tpm-to-secure-your-iot-device-data/

[69] W. Turton. 2021. Hackers Breach Thousands of Security Cameras, Exposing Tesla, Jails, Hospitals. https://www.bloomberg.com/news/articles/2021-03-09/hackers-expose-tesla-jails-in-breach-of-150-000-security-cams

[70] UK National Cyber Security Centre. 2019. Zero trust architecture design principles. https://www.ncsc.gov.uk/blog-post/zero-trust-architecture-design-principles

[71] UK National Cyber Security Centre. 2021. Zero trust architecture design principles. https://github.com/ukncsc/zero-trust-architecture

[72] K. Goldman W. Arthur, D. Challener. [n.d.]. Quick Tutorial on TPM 2.0. https://link.springer.com/chapter/10.1007/978-1-4302-6584-9_3

[73] Lan Wang, Vince Lehman, A. K. M. Mahmudul Hoque, Beichuan Zhang, Yingdi Yu, and Lixia Zhang. 2018. A Secure Link State Routing Protocol for NDN. *IEEE Access* 6 (2018), 10470–10482.

[74] Tom Yates. [n.d.]. Secure key handling using the TPM. https://lwn.net/Articles/768419/

[75] Yingdi Yu, Alexander Afanasyev, David D. Clark, kc claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking, ICN '15, San Francisco, California, USA, September 30 - October 2, 2015*. 177–186.

[76] Zhiyi Zhang, Yingdi Yu, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. 2017. NAC: name-based access control in named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking, ICN 2017, Berlin, Germany, September 26-28, 2017*. 186–187.