Trust Management Meeting Minutes

First NDN trust management meeting (UCLA, November 1, 2012)

Gene Tsudik Ersin Uzun Zhenkai Zhu

1 Introduction

Trust management is very important in NDN since all content packets are signed and digital signatures are only meaningful if uniquely associated with a trusted (public) key.

After gaining experience in building several applications over NDN during the first two years of the project, this meeting provided us with a timely opportunity to explore commonalities and differences on how trust is, and can be, handled accross a variety of NDN applications.

2 Certificates and Certification

Currently, NDN supports x509 certificates in certain library functions. After a lengthy discussion, the following conlusions were reached:

- x509.v3 certificate format is the *de facto* standard which is worth supporting, since its syntax is sufficiently rich, i.e., capable of carrying lots of various information. Also, it offers some flexibility via options fields.
- However, x509v3 also comes with lots of baggage: (1) designed with hierarchical PKI in mind, (2) has some unnecessary/extra fields, and, (3) if we will have to heavily customize it with options fields and write a custom code to parse/verify them, it might be worthwhile to consider our own certificate format.

3 Naming and Trust management

Currently, all trust management is handled at the application layer. Naming is a powerful tool and it can facilitate trust management by embodying policy in naming convention in order to minimize configuration overhead. If such a convention (that would play well with all applications) can be agreed upon, some common trust management functionality can be added to, and handled in, NDN libraries. After discussing a range of applications (routing,

lighting, audio conferencing, etc.) and looking for an appropriate naming convention, we concluded the following:

- Key name should be specific enough to limit usage of the key (i.e., similar to a capability).
- There should be a naming convention that binds a content object with a key signing it. The one we agreed during the meeting was:

  > A content named C can must be signed with a key named: "$X\|key''$, where X is a proper prefix of C and "key" is a reserved last component (distinguished with a marker) that explicitly indicates that it is referring to a key.

- This convention does not apply if the signed content is itself a key (of type Key). This is because a chain of "trust" can be traversed and trusted as long as it ends with some trust anchor.
- The schema for content name following the matching key prefix (e.g., number of name components) and the maximum lenght for an acceptable trust chain are application-dependent and should be expressed by the application to NDN libraries that perform verification.
- Applications should also be able to enforce custom verification policies (e.g., a given app might not care about certificate expiration and/or revocation checking). This can either be done by a callback to the app or by requiring the app to communicate its policy ahead of time.
- It is better to keep both (1) key object type and (1) key marker in the name, in order to prevent unintentional classifications due to careless naming.

Although the proposed convention seems to apply to most applications, it might not be a good natural fit for industrial control, since each device has multiple controllers (e.g., fire department, building maintanance, control application etc.) each of which would likely "live" in a different namespace. Thus, the choices are: (1) the device should know all controllers, (2) controllers should know all devices; or, (3) devices could live in more than one namespace. Jeff and his team will think more about this and let us know if the above convention would be too limiting/taxing.

4 Transcribed Meeting Minutes

Attendees: Jeff, Alex H, Naveen, Ilya, Yingdi, Zhenkai, Wentao, Van, Michael, Alessandro, Alex A, Beichuan, Gene, Ersin, Some Folk on Skype
Gene: let's discuss the agenda of today.
Van: I'd like to see how to use naming scheme to facilitate trust management. Example: routing, lighting control. - embody policy in naming convention to minimize the configuration overhead
Van: I wish to see the trust management functions into the ccnx library.
Alex H: want to discuss trust for specific apps
Van: there are broken pieces in the library: put key in key locator, and also access control implementation is broken
Van: PARC prototyped in Java, and the security framework there is not correct. It's not in the right place.
Jeff: want easy integrate things like HMac with CCNx code
Van: need to figure out the right place for the trust management upcall in the ccnx library
Gene: how would cert be represented in ccnx
Van: can be represent with content object and meta data
Gene: nobody loves x509v3, but it's de facto format. I think the current code supports only x509 format. how would library cooperate with cert that is not x509v3 format
Gene: today wants to see what kind of certs we support. there were past efforts to propose own cert formats by other projects, but all failed
Gene: are we going to adopt x509
Van: we got it already and didn't have time to remove it
Gene: x509 contains a superset of info of what we need
People: x509 has a lot of info, but a lot of info is not exposed in the C library (e.g. no simple way to get expiration date)
Van: want to know application need, for example, how to treat expired cert vs invalid cert, what level of details in the certs does application need?
Jeff: there is a need to check expiration date of a key in the app, I want to grant temporary access to some application, e.g. 1 day or 10 days, etc. (time-limited access)
Gene: in security area, expired cert == invalid cert. clock sync is needed to check the expiration date, but it's separate issue
Van: we care how expired cert manifests itself to the app, i.e. what kind of fields is need to be exposed to the app (e.g. just return valid or invalid, or provide more info, e.g. actual expire time...)
Gene: various info (e.g. reasons of invalid key) should be valuable to application and should be exposed to application for them to decide what to do next
Ersin: app could be able to specify what it cares or does not care by setting some parameters (passing policy config) when calling the trust management function
Van: validity checks in openssl lib has to do with syntactic, semantics and PKI model; in

NDN, we'll produce syntactically valid certs, but we can not ask openssl lib to only check the syntatics and do not perform other checks
Gene: we need to check the expire date
Van and Jeff: some app does not care about time, e.g. lighting only cares of sequences, but not time. some stuff may have different notion of time; some may even not have a notion of time at all.
Gene: time check is a MUST for certs. period. Seveal people disagree. Meanwhile, Van and Gene repeat themselves. Van keep saying some devices do not have clock, and Gene asks for example applications of such scenarios, and Van didn't give examples.
Gene: library must check time, but application can choose to proceed regardless of the time check results (e.g. a device which has no clock can return failed time-check result, but the app can ignore the failure and proceed)
Van: I thinks Gene is mixing two things: 1) in normal environment people could and should check all fields in the certs including time 2) in devices with no clock, no time could be obtained, there is no meaning in checking time
Gene: Why time-check is so hard? Why can't you just check the time even if you don't have a clock. You can obtain clock from remote or simply say the time is zero.
Van: the disagreement is here: GMT timestamps may be meaningless for a lot devices; how to check time is relevant to the local environment, not related to GMT times. so checking time with GMT should not be put into library
Gene: have you heard of an application that has no notion of time?
Jeff: GMT time is sometimes not important to lighting, for example. local time, or sequences, is more important.
Gene: devices' capability is improving as a result of Moore's Law. we should not be designing for yesterday's devices. Sequences without time only protects re-ordering and replay attack.
Gene: if the requirement is so simple, the discussion so far (about certificates) is irrelevant to these kind of applications.
Gene: So here is what I got: Van's point is the time-check is needed, but it's up to application to define what is time-check and how to perform time-check (using firearm safety as analog)
Van: depending on global time increases the vulnerability of the certs (list examples, somebody could have exploit the NTP or changed the time using other means)
Gene: if you can compromise the server and change the time, then the certificate is irrelevant (i, again as zhenkai, guess Gene means it's harder to compromise server then crack cert)
Ersin: can we have our own x509 format that supports different notions of time, including local time.
Gene: whatever; do whatever you want; I've advised you, and proceed on your own risk.
Ersin: the point for today's discussion should be "1. is x509 the best thing for NDN? " "2. should we come up with our own format of cert "

Gene: cert format is just format, it is not related to trust model
Ersin: yes, cert format is related to trust model
Gene: that needs another day's of meeting just to decide on the format of certs; and designing a new format of cert is not useful
Van: fine. we'll check what's in the x509 (including expiration date), let's move to lunch.
Ersin: maybe we should focus on trust model and then decide what fields we need to add in the cert after we agree on trust model
Alex H: we'd like to add new fields in x509
Gene: x509 is extensible, you can always add new options into x509

===== Lunch break =====

Ersin: in NDN, one can sign links to content object, and the signature of these objects need not to be checked. You trust these content objects not because we trust their signature, but because of the links I gave you, which is secure. This is new.
Gene: no, it's not. When you gave me the links (signed by you), you implicitly signed those objects, because the links contains the exact names of the objects including there hash.
Beichuan: describe routing scenario. each operator of a routing domain configures a root CA; when adding routers, the CA puts a cert in the router; router signs daemon (process) keys on the router; daemon keys signs actual routing packets.
Gene: Here is a scenario: a router decides to run RIP, but the domain does not allow RIP; do we want to prevent that?
Beichuan: a router running RIP would affect others, who'll just ignore the RIP updates.
Gene: several compromised routers could respond to the RIP updates
Beichuan: we have nothing to do if all routers are compromised
Gene: do we need policy specifying which protocols are allowed
Michael: and what if the RIP data is named after ospf data
Van: let me show you a picture of a strawman design for the routing
Van: shows a picture: BigCo/NetOps/SFpop/OSPF/rtr731/pid345/LSP#678 (the name in one link state packet generated by the SFpop IGP routing process rtr731) signed by BigCo/NetOps/SFpop/OSPF/rtr731/pid345 (the name of the routing process cert (given to the process when the router creates it). This name must match the bold part of each packet's name) signed by BigCo/NetOps/SFpop/RTR731 (the name of the router cert (given to the router when it's configured)) signed by BigCo/NetOps/SFpop/config/employee975 (The name of the cert of the employee who last configured the router(This key signs the entire router config as well as router's signing key.) signed by BigCo/NetOps/SFpop/config (The name of the cert that is the root of trust for SFpop router configuration)
Van: we want the namespace to be rich enough to minimize configuration error; the name of the key should be specific enough to limit the usage of the key (e.g. the key can only be used to sign routing updates)
Jeff: some irrelevant question: different type of content object processed differently by ccnx library (e.g. DATA, KEY, LINK, etc.)?

Michael: yes.
Gene: is it on purpose that 'RTR731' line is not a prefix of the previous two?
Van: yes. The top two uses a convention: a content name signed by a key must have the key's name as a prefix. and start from the third one, the convention no longer holds.
Gene: why OSPF/rtr731 appears in the up two names
Van: the name until OSPF is the name your listen to and the suffix is for individual process
Gene: when we go to interdomain, somebody compromise intradomain router, and perhaps can do some bad stuff to interdomain routing
Van: inter and intra are different; intra domain has a well-defined root; inter does not have a well-defined root. We'll work on intra domain first
Gene: would this model benefit from limiting keys to protocols (e.g. an employee has different keys for different protocols, not a single key that can do anything)
Van: yes, you can add terms to limit what a key can sign into the name.
Van: ISP have different teams; intra-team communications are different from inter-team communication. E.g. the above employee is a pop team member, it can not speak on behalf of other teams. so it's not a single key that can do anything; what it can do it defined by the organization's nature.
Gene: so it doesn't make sense to make any general rules on the keys are named, because different organizations may have different rules
Van: This is an example of using richness in the name to solve the trust problem.
Van: whenever you get the routing update, you always have the root of trust (BigCo/NetOps/SFpop/config)
Ersin: should we specify what prefix a key is responsible for, i.e. what names a key can sign
Van: key used to sign content objects can only sign the name that has exactly one more component of the key name
Gene: only the above two names obey this rule.
Van: yes. when you go to the key verification chain, you don't need naming convention, you just follow the trust chain. So the convention only applies to normal content objects and first level key, not to the keys thereafter.
Beichuan: why doing this?
Van: when doing key verification, there is explicit linkage when you follow the trust chain
Gene: no doubt about the linkage, but how do you specify what a cert owner can do, i.e. there capabilities
Van: you can do whatever in the namespace, e.g. in the above, the employee is allowed to do config (because it's a 'config' namespace), and it is ok for him to do whatever configuration necessary
Van: the pattern at the top two (the name of the key tells what the key can be used for) is the common case
Ersin: does it make sense to identify the common binding between a key and the names it can signs

Jeff: it's pretty common that some cases the up two names work (which obey the convention), and sometimes we need the button three cases (out of hierarchy in key names)
Ersin: do you think it's good to specify what key name a key can sign in the 'key name' itself
Van: it's a wrong thing. For example, a router has a lot of semantics about different protocols, you can't put all these info in the key name
Gene: how can we confine the employee to only configure OSPF
Van: you can put more limitations in the name; it's applications specific pattern, but there is no general pattern
Gene: do we want to spend more time to find the general pattern, or let it be as is
Beichuan: do you have any suggestions
Gene: routing is special. i would love to see the protocol in the root, i.e. you have root for OSPF, for example, and different root for RIP
Ersin: an employee now has to have a lot of keys for different protocols in order to configure the router
Gene: there aren't many protocols
Van: configuring routers cover a lot of aspects other than protocols
Gene: is it needed to separate the roles of different protocols.
Beichuan: we talked different keys for different interfaces, do we still need that? is it our goal to operate like what you showed in the specific SFpop organization example or more general
Van: you want your system to embody the real operational scenario, but hopefully you can do that in general pattern. i.e. you don't have to make a rule for each different organization; we can give people a cook book of how to handling trust management; don't want to specify a lot of details that are not relevant to operation or make operation difficult. (gave example that people in DoD often turn off security when deploying something because it's preventing communications)
Gene: the middle name (the route key) does not say anything about the protocol Alex A: we should also specify what a key can do in addition to just signing the key
Van: past experience proved that capability-based security (specifying what you can do in names, etc.) is wrong; whenever people put types into the certs, it's very hard to capture the fine-grain control of type fields;
Gene: maybe we ought to stick more info into the names, this does not lead to the explosion of keys, but only the explosion of certs (a key can be used in multiple certs)
Jeff: in the library, we could have some way to see the capability of the key;
Gene: yes. using naming convention (Van mentioned above) is just a convenient way of combining the capability and the key. This is the same as today. If UCSD publishes something about UCLA, you don't trust them although they have a valid cert from, say, Verisign, because you can see the different from the cert name and the content name.
Alex A: but there is only one trust chain to follow
Ersin: you can name the key with different names

Alex A: where would this key stay
Van: we'll sync the key. collect all the keys in the repo
Van: the first LSP a router send would have the keys included (keys are flooded among routers). (this is only for routing)
Jeff: can a key publish anything other than key; e.g. if i send /BigCo/NetOps/SFpop/OSPF/rtr731/pid345 interest, and i want to fetch content object with longer name but not the key?
Van: we can put literal 'key' component in the name
Gene: should we have a reserved term 'key' for the name of a key?
Van: anything you put in the name let you to ask for it explicitly.
Gene: so we should put 'key' in the name
Michael: aggree, but it doesn't make sense to get rid of the KEY type, you can use it and you can also add a 'key' term in the name. The type is just for extra check (it is your intention to publish this as a KEY, not as a normal DATA object)
Ersin: it's good to have a type KEY, what if people accidentally put a 'key' component in their data names.
Gene: is it good to have a way to issue interest just for key? i.e. put the type of the requested data in interest
Gene: you know the exact name of the key, because you want to verify something, and then you already know the name
Ersin: yes for most cases, but for self-signed cert, you don't have the exact key name because you don't know the hash of the content object until after your finish the signing
Gene: Ahh.. so this is the loop.. I see
Ersin: put a component 'key' makes sense
Gene: so we put a 'key' as a last component of the key name (Ersin agrees)
Van: in lighting it's different
Gene: in the cert, one could speicify how many components can be added to the key prefix that a key can sign
Van: it's too application specific (a key may be used by multiple applications, and application may have different requirements)
Michael: we should have a scheme to specify what kind of names a key can sign
Van: this can not be a convention that all application instances agree on
Van: it's a property of an application, the app should provide this scheme to the trust library People agree
Gene: each application does its own check
Jeff: but chasing down the hierarchy of trust should be done by the library
Gene: should we provide a routine, first compare names between content name and key name (before the checking of signature), tell the application the difference, and application decide what to do at this point, or application can specify a pattern when calling the routine and the routine automatically decide what to do after comparing the names
Jeff: we'd prefer latter.
Gene: how do we provide the mapping between a key name and the names it is allowed to

sign for the bottom three key names in the above picture? (where the naming convention no longer holds)
Van: we don't have a good method yet.
Gene: should run a experiment with routing to see if the solution discussed so far has usability problem
Jeff: is there a clean boundary between 'hierarchy' and 'non-hierarchy' parts? E.g. the first two names in the picture are 'hierachical' and the rest are not
Gene: the data content object should be in hierarchical domain, while the key objects could fall in 'non-hierarchical' domain Alex A: should this be the only way?
Ersin: in today's operation, people expect a 'hierarchical' matching between the key name and the content name; so this is the way to go, but not the only way
Van: agree
Van: in applications like lighting, there should be mechanisms to go from the key name to what a key can do (capability)
Gene: there is similarity between routing and lighting. both are control based and fall nicely in hierarchical model
Jeff: the naming of the lighting is according to the physical locations right now
Gene: I want to mention cross certification: One CA certifies another CA in the same level or CAs below; this may help in lighting case
Van: there are always (overlapping) multiple hierarchies in control systems or internet systems. cross certification leads to a lot of variances and problems
Gene: you'll need to describe what the keys can do then
Van: specify a set of 'verbs' a key can do; it's the only names you can use
Gene: who signs these 'verbs'
Alex H: installer signs Gene gave an example of 'cross certification', show it's indeed what we want (e.g. ucla authorizes the fire department in westwood to turn off the lights in ucla buildings)
Van: yes. this is what we need. I probabaly misunderstood the words 'cross certification'
Jeff's example: A /ucla.edu/prefix-to-fixture/command/signature B /ucla.edu/facility/fire/.../key
Van: we have two choices. 1) put ucla fire and safety in the namespace and authorize fire department in this namespace 2) put fire department key in lights (?) Example: fire department can turn off power in big facilities
Gene: lights should know who can control them
Van: then you have to enumerate all possible controllers. 1) either the lights knows all the controllers, or 2) controllers know the lights
Gene: you can make the root of trust of the controllers general enough, e.g. 'California first responders'. So any first responder in california can turn off the lights in emergency.
Jeff: we can give fire department keys in the namespace B (code the root of trust '/ucla.edu/facility/fire' in fixtures), and then the fire department can still address the fixtures using the names in namespace A Van talked about an example of allocating a subset of lights to a production team in theater; we can use a separate namespaces for these

lights and give the keys for that namespace to the specific production team Jeff agrees.
Jeff: notice some similarity between lighting and routing in trust management, can we generalize the pattern?
Van: how do we generalize for steps in the middle (of the routing example) is difficult Van and Jeff have conversations over the routing example and its relationship with lighting (a little lost as they are talking to each other).
Jeff: the employee in routing example is like the application in lighting
Van: I thought the OSPF in the routing example is more like the application People try to find similarity between lighting and routing
Van: do we really need to find similarity? we are doing two applications for a reason
Gene: yes, they are two different applications, but different apps may have the same security requirements
Gene: so the (command) interest carries something like a content object, because it carries a signature Van and
Jeff: yes.
Van: the difference between the two applications is: in routing, there is clear hierarchy; pop runs multiple routers, which runs multiple routing protocols. in lighting: many masters, no clear hierarchy. Hence the simple hierarchical trust model in routing does not fit in lighting application

Michael brought up something like multiple signatures, e.g. multiple signatures are required to authorize something (perhaps triggered by DoD example, you have to get authorization from multiple persons in order to access something). Van and Gene talked the possibility of doing this
Ersin: let's do not go into this branch now. At worst, it can be solved by having a link kind of thing that points to the second signature. People agree that this is not the topic for today's discussion.
Van: NDN allows us to use controlled flooding for safety command. e.g. prepend a topological name prefix to a command to be flooded. e.g. /ucla.edu/bolterhall/fire/blahblah, /ucla.edu/bolterhall is topological and /fire/blahblah can be flooded within bolterhall
Gene: how about other control systems, what is the most acute control system
Jeff: nuclear power plant control...
Gene: how about smoke detection, sprinkle system
Michael: it has own special wires for these kind of system
Jeff: it's not common to use ethernet control for these systems
Van: big facilities needs aggregated constraints, i.e. the constraints are no longer independent as in lighting (e.g. you can turn off any one cooling, but no more). the currently practice, there are middle proxy agents that enforce the aggregated constrains, but the middle proxy agent can be the single point of failure. And there is desire to move the distributed control system that still enforce the aggregated constraints.
Van: NDN loves broadcast, and in control system, broadcast is available in low levels (e.g. in a building). the current system is just not using it. By using broadcast, different

components in a system can know what's going on in other parts and help them make better decisions

===== break =====

People discuss the longest hops of trust chain to follow during the break.

Gene: the expiration date one keeps for the cached key object should be the nearest expiration found in the trust chain

Van: agree

Ersin wrote some conclusion on the whiteboard: content and first key 1. fixed, proper prefix 2. options expressed by applications: schema for content name after matching key prefix, largest hops to follow in trust chain

Audio

Van: audio conference has similar pattern as the routing example, the organize picks a name prefix for the conference data. e.g. /broadcast/room the next level, each application instance would have a key (correspondent to the routing daemon process) the next level, it is the user's institutional key, which could be verified by following the trust chain(i.e. user generates a key each time he joins the conference) (corresponding to the router's key)

Gene: So the keys used to sign the conference data are ephemeral keys. They are considered shorter

Van: Yes. we need get rid of the current keystore; develop a new keystore like the TPM module, so that the shorter keys are not exposed as often. Van's point: the audio conference scenario is similar to routing case

Gene: There is an alternative method: ask organizer to be the root of trust, generate and distribute the certs to participants People: but the conference can not continue if the organizer leave

Gene: true, but it is a common scenario for conferencing Me explaining what's the current implementation. Gene suggests that we should do it in his way (his suggestion of organizer as a trust root)

Video distribution

Jeff: we distribute encrypted video content without sharing the key. maybe we should consider the broadcast encryption

Gene: there are broadcast encryptions. they are popular in late 90s and early 2000s. The problem is revocation. We don't touch revocation today.

Game

Jeff: we want to do a pure p2p game

Beichuan: This is similar to audio case

Gene: there are people working in this area, we should not reinvent the wheel (GameSec)

Beichuan: we need a trust anchor anyway

Gene: we can use the institutional key for the players

Beichuan: why would others trust this 'organizer'

Gene: the key is the pseudo identity for the gamer Jeff mentions that they'll use linkedin connections as the boostrap for the trust model in game

Gene suggests we conclude our Consensus today: 1. Parsing of the cert. (nothing gets exposed to the application in the current library call except 'verified' or 'unverified'.)
Gene: we could 1) make the application tell what is the policy 2) pass a callback and let the application code process it
Gene: get the primary result plus the secondary return value of reasons
Jeff: app could get the result and issue another call for reasons if desired
Gene: how about the fetching
Michael: auto fetching key is kinda supported
Gene: what happens if there is problem
Michael: 'unverified', but the application can ask the library to verify it. If the key can not be fetched, he has no idea what's going to happen

Consensus: Verification should support policy (app cares which fields or does not care which fields)

2. Routing and Instrumented environment Consensus: 1. key name must be the prefix of the content name 2. we should have a scheme for application to specify the suffix of the content name (after the key name) 3. we should have method to specify how far to follow the trust chain (how many hops to follow, when to return, etc.), but this should be simple, otherwise it's hard to implement and hard to use.