PAPER

Implementing a Secure Autonomous Bootstrap Mechanism for Control Networks

Nobuo OKABE[†], *Member*, **Shoichi SAKANE**[†], **Kazunori MIYAZAWA**[†], **Ken'ichi KAMADA**[†], **Masahiro ISHIYAMA**[††], **Atsushi INOUE**[††], *Nonmembers*, *and* **Hiroshi ESAKI**[†††], *Member*

SUMMARY There are many kinds of control networks, which have been used in various non-IP network areas, such as BA (Building Automation), FA (Factory Automation) and PA (Process Automation). They are now introducing IP and face the issues of security and configuration complexity. The authors have proposed a model which intends to solve these issues while satisfying restrictions, i.e. small embedded devices, isolated networks and private naming system/name space, which are required when introducing new functionality into existing control networks. Secure bootstrap sequence and device-to-device communication using the chain of trust are the points of the model. This paper shows the practicability of the model through implementing the model experimentally.

key words: *Control networks, Security, Auto-configuration*

1. Introduction

Control networks are different from IP (Internet Protocol) with regard to their history, purposes and technology. There are numerous standards of the control networks, e.g. FOUNDATION fieldbus * [1], PROFIBUS ** [2], MODBUS *** [3] and BACnet **** [4] which have been used in various non-IP network areas, such as BA (Building Automation), FA (Factory Automation) and PA (Process Automation). Multiple standards coexist within a single system usually because the system's requirements are diverse. Many standards are introducing IP as a transport technology, e.g. FOUNDATION fieldbus HSE, PROFInet, MODBUS/IP and BACnet/IP.

There are issues when introducing IP into control networks [5]. First, current control networks have not sufficiently considered security [6]. Network security must be necessary when introducing IP into them. However, security is not easy for control networks because the small embedded devices commonly used in control networks have limited computational performance because of their restricted requirements of cost, physical size and power consumption. The other issue is the configuration complexity of the control networks. Devices are usually manually configured in the fields whereas their user interfaces are not so powerful, like PCs. However, the number of devices are increasing because measuring and controlling need to be more precise. For example, a BA system of a large building complex in Japan has 170,000 control points with 16,500 devices *****. This will present not only the cost of engineering but also the possibility of human errors in the future if a labor-saving mechanism is not introduced.

The following are restrictions when introducing new functionality into the control networks.

- Small embedded devices:
  The small embedded devices commonly used in the control networks have limited computational performance because of their restricted requirements of cost, physical size and power consumption. Some devices will have more powerful CPUs in the future. At the same time, low-power CPUs will survive because choice of CPU depends upon not only cost performance but also power consumption which has an impact against battery operation or bus width which has an impact on circuit size.
- Isolated network environments:
  The control networks do not always require connectivities to the Internet even though introducing IP. It is the user's choice whether to connect to the Internet. Hence, functions introduced into them have to work well under an isolated network environment.
- Private naming system and private name space:
  Information of the control networks, not only the traffic but also device's name, has to be confidential, because the information can help to indicate corporate activities, e.g. the capability of plants. Hence, the naming system should be closed to the public if operators desire. It is also important not

Manuscript received January 1, 2003.
Manuscript revised January 1, 2003.
Final manuscript received January 1, 2003.

[†]Yokogawa Electric Corporation
[††]Toshiba Corporation
[†††]The University of Tokyo

[††††]This research is supported/funded by the Ministry of Internal Affairs and Communications of Japan.

[*]FOUNDATION fieldbus is a registered trademark of the Fieldbus Foundation.
[**]PROFIBUS is a registered trademark of PROFIBUS International.
[***]Modbus is a registered trademark of Modicon, Inc.
[****]BACnet is a registered trademark of ASHRAE.

[*****]`http://www.echelon.com/about/press/2003/echelon_mori.htm`

to force a device's identity to be global unique if most of the devices should not be accessed from outside. For the above two reasons, DNS is not an appropriate naming system for them.

We have proposed a model [7], which intends to solves the above issues while satisfying the above restrictions when introducing IP into the control networks. It is important for them to inherit existing property when introducing new functions because they have large property, e.g. specification, operational knowledge and applications. This is the reason the model is a framework whose functions are addressed to either below the application layer or the middle-ware instead of inventing new control network protocols.

In this paper, we show the practicability of the model by implementing it experimentally. This paper shows an overview of the model in Section 2, details of the model in Section 3, the implemented system in Section 4, considerations through implementation in Section 5, related work in Section 6 and further study items in Section 7.

2. Proposed Model

2.1 Network Security

The authors have already studied a security mechanism [6] which can satisfy the restrictions described in Section 1. We will use this mechanism in our proposed model. The following are the features of the security mechanism.

- This mechanism provides end-to-end security. Most control networks rely on a firewall model which assumes specific network topology. However, end-to-end security will be necessary because wireless technology and nomadic devices break the firewall model.
- Communication is protected by IPsec [8] which provides IP packets with confidentiality, integrity and authentication with the other end. IPsec is useful because its enforcement is independent from applications and sharable among them. IPsec is applicable to small embedded devices due to not using public key cryptography.
- It is important for IPsec to share a secret, which is called IPsec SA (Security Association), between both ends. Key exchange protocols will be important if running IPsec on small embedded devices because these devices do not have a powerful user interface like a PC, which makes manual keying difficult. The security mechanism uses not IKE (the Internet Key Exchange) [9] but KINK (Kerberized Internet Negotiation of Keys) [10] for the key exchange protocol. IKE is the most popular key exchange protocol for IPsec. However, it is not suited to small embedded devices because the Diffie-Hellman key exchange is mandatory. KINK can work well on small embedded devices because KINK is based upon Kerberos † [11], where public key cryptography is not mandated.
- In the security mechanism, a node's identity is in the manner of Kerberos, i.e. a principal-id, which is a combination of a realm name and a principal name.

2.2 Auto-configuration using a Directory Service

To simplify the configuration process, the model provides the device's application layer with an auto-configuration mechanism. The basic ideas of the auto-configuration are 1) to minimize pre-installed information in a device, and 2) to acquire most information from servers located in networks. IP address configurations are beyond the scope of the model. It can be done by DHCP (Dynamic Host Configuration Protocol) in IPv4 or RFC2462 in IPv6, with which the model can be combined. The auto-configuration requires not only name/address resolution like DNS but also general data handling, e.g. searching, getting and updating data. We introduce our own directory service named PS (Property Server) [12]. The following are the features of PS.

- It is not a prerequisite condition for PS to connect to the Internet because PS does not require global tree structures like DNS.
- PS maintains a device's attributes as metadata of the device's identity. A typical example is that an IP address IP_{FOO} is an attribute value of an attribute type $ATTR_{IPaddress}$, and the attribute, i.e. the type and the value, belongs to a device's identity FOO as metadata.
- PS supports two types of transactions, i.e. PUT and GET. PUT sets/updates attributes in PS. GET acquires attributes from PS. Any request of transaction has search conditions which designate attributes to be affected. For example, identity/IP address resolution is done by GET transaction, where search conditions is the value of $ATTR_{IPaddress}$ belonging to the identity FOO, which returns IP address(es) IP_{FOO}.
- PS's protocol uses XML for the future extension.
- In the proposed model, every node belonging to a system has to use the security mechanism described in Section 2.1. Illegal access to PS from outside can be prohibited with IPsec security policy simply. An access control list can be introduced into PS if accurate restrictions are required.

†Kerberos is a trademark of the Massachusetts Institute of Technology (MIT).

2.3 Bootstrap Sequence using the Chain of Trust

When considering secure auto-configuration, devices have to discover a trusted PS, then exchange data with PS under secure communication channels. The following bootstrap sequence, which we call the Chain of Trust, satisfies the above requirements.

1. Devices can trust Kerberos server KDC (Key Distribution Center) by sharing a key. It is a prerequisite condition of Kerberos.
2. Devices should trust PS which trusted KDC shows.
3. Devices register their own information, e.g. a principal-id and IP address(es), to trusted PS. The information will be used for discovering peers (see Section 2.4).
4. Devices should trust data which trusted PS provides. Then devices can complete the sequence. The communication is protected by IPsec.

Hence, the minimum information with which a device has to be pre-installed is a principal-id and a key shared with KDC, i.e. a secret key of Kerberos. Other information can be acquired from PS.

2.4 Device-to-Device Communication

In control networks, communication is occupied by control messages and notification messages between devices, e.g. controllers, sensors and actuators. Hence, devices have to discover trusted peers, then to exchange messages with them under secure channels. The Chain of Trust can be applied in that case.

1. Devices search their peers using PS because they have already know trusted PS. (see Section 2.3).
2. Devices should trust peers which trusted PS provides. The communication is protected by IPsec.

3. Details of the Model

Figure 1 shows the bootstrap sequence and the device-to-device communication using the chain of trust. Details of each function is shown in Figure 2 through Figure 6.

1. KDC Discovery (KDCD):
   DHCP server(s) advertise KDC related information, e.g. KDC's IP address(es) and realm name(s) where KDC offers authentication services and NTP related information, e.g. NTP server's IP address(es), (see Figure 2).
2. Authenticating KDC:
   Device X needs to authenticate KDC advertised by DHCP server (see Figure 3). First, X adjusts its clock with NTP server because Kerberos requires that every device synchronize its clock to prevent replay attacks. Second, X authenticates KDC, which X learned with DHCP, through verifying a returned packet. TGT (Ticket Granting Ticket) in the packet is necessary for subsequent Kerberos services.

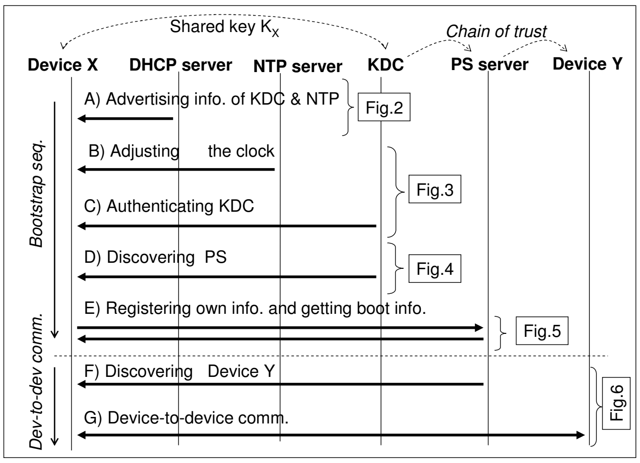

Fig. 1 Messages of the proposed model

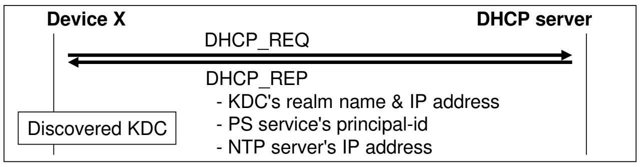
Device X
DHCP server
DHCP_REQ
DHCP_REP
- KDC's realm name & IP address
- PS service's principal-id
- NTP server's IP address
Discovered KDC

Fig. 2 KDC Discovery

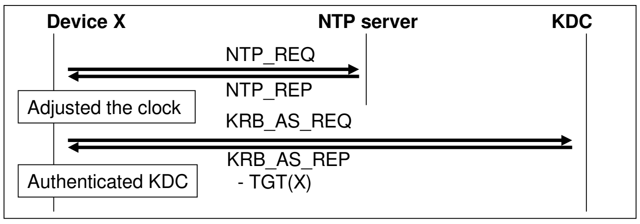

Fig. 3 Authenticating KDC

3. PS Discovery (PSD):
   Device X acquires PS's information from KDC (see Figure 4). X shows its principal-id to KDC. Then KDC returns PS's information, i.e. PS's principal-id and IP address(es).
   X has to acquire a service ticket for PSD, e.g. TICKET(PSD), prior to the above procedures. KRB_PRIV messages, which need a service ticket, are used for protecting PSD because IPsec/KINK cannot be used at this moment.
4. Booting up:
   Device X registers its own information to PS which is used for device discovery, and acquires its boot data from PS. Then, the bootstrap sequence is completed (see Figure 5).
   First, X acquires a service ticket for KINK with

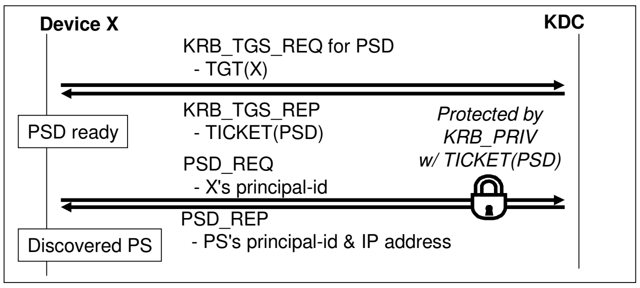

Fig. 4 PS Discovery

PS, e.g. TICKET(KINK w/ PS), from KDC. Second, IPsec is established between X and PS after exchanging KINK messages. Third, X registers its own information, e.g. a principal-id and IP address(es), which other devices or servers can use for discovering. Fourth, X acquires its boot data. Then X completes the bootstrap sequence.

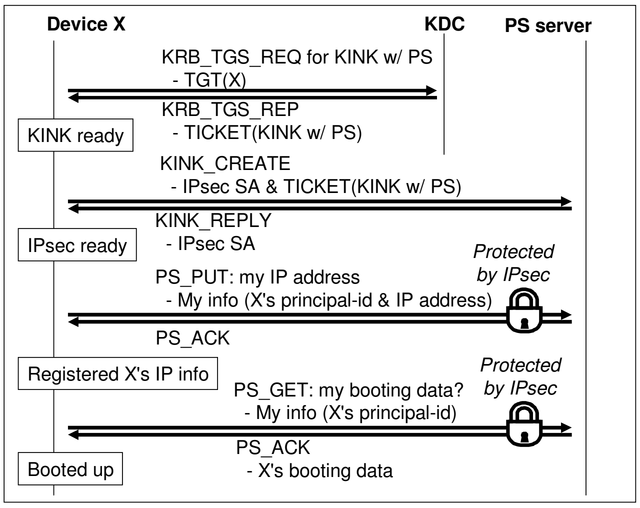

Fig. 5 Booting up

5. Device-to-device communication:
   Device X can discover device Y using PS, i.e. device discovery, then starts the device-to-device communication with Y (see Figure 6).
   First, X discovers Y through PS. A typical example is that X resolves Y's IP address from Y's identity like DNS. IPsec between X and PS has already been established at the bootstrap sequence. Second, X acquires a service ticket for KINK with Y, e.g. TICKET(KINK w/ Y), from KDC. Third, IPsec is established between X and Y after exchanging KINK messages. Then the devices can exchange application messages which are protected by IPsec.

4. Implemented System

We implemented the model to examine its practicability, i.e. object code size and performance, experimentally. Table 1 shows the specifications of an experimentally implemented device, whose CPU is H8/3029 (Renesas Technology Corp.), which has cryptographic hardware in a Xilinx's FPGA. Renesas's H8 family is a popular low-end CPU in Japan. Table 2 shows the specifications of servers which were used for the system.

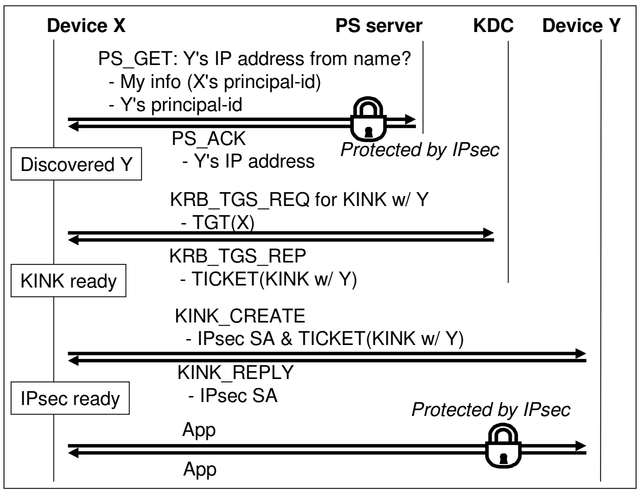

Fig. 6 Device-to-device communication

Table 1 The specifications of the device

H/W	H8/3029@20MHz, Crypto H/W@20MHz (3DES,MD5)
OS, IP	Toppers FI4 w/ Original IP stack
IPsec	ESP (3DES-CBC,HMAC-MD5)
Kerberos	Original code based RFC4120 (etype:des-cbc-md5)
KINK	Original code based RFC4430

Table 2 The specifications of the servers

DHCP	CPU:pentium-III@1.2GHz, MEM:128MB, OS:freebsd4.10R
NTP, KDC	CPU:pentium-III@750Mhz, MEM:896MB, OS:linux2.6.8, Kerberos:Heimdal-0.6.2, KINK:racoon2
PS	CPU:celeron@1.7GHz, MEM:1GB, OS:linux2.6.8.1

4.1 Object Code Size

Table 3 shows the code size of the device. The total size will be about 30K bytes greater if the cryptography, i.e. 3DES and MD5, is implemented by software instead of hardware.

Table 3 Object code size of the device (K bytes)

Module	Size	Module	Size
OS	64	Kerberos	25
IP (v4/v6)	132	KINK	20
IPsec	8	Crypto	7
		App	16
		total	272

4.2 Performance of the Bootstrap Sequence

Table 4 shows the processing time of each function on the device, whose conditions are with and without cryptographic hardware. The values without parentheses mean net processing times, and the values in parentheses mean waiting times from sending a request til receiving a reply. KINK$_I$ and KINK$_R$ mean the processing of KINK initiator and responder. PS$_{PUT,IPsec}$ means PS's PUT transaction which is for registering the device's IP address. PS$_{GET,IPsec}$ means PS's GET transaction which is for getting the device's boot data (512bytes). Both transactions include the overhead of IPsec ESP. The waiting times of KINK$_I$, i.e. the values in parentheses, are varied because they depend upon the performance of a peer. The waiting time of 112 msec occurs where a device initiates KINK with PS. The device with and without cryptographic hardware make a kink initiator wait for 73 msec and 93 msec respectively. Kerbeeros related performance, i.e. TGT, TGS, PSD, KINK$_I$ and KINK$_R$, is improved and about five times faster at the greatest than the previous prototype.

Those values exclude the processing time of IP address configurations, i.e. DHCP in IPv4 or RFC2462 in IPv6, and L2 address resolution, i.e. ARP (Address Resolution Protocol) in IPv4 or ND (Neighbor Discovery) in IPv6.

Table 4 Processing time of each function on the device (msec)

crypto H/W	w/	w/o
KDCD	59 (0.5)	59 (0.5)
NTP	58 (0.5)	58 (0.5)
TGT	26 (34)	74 (34)
TGS	49 (35)	178 (35)
PSD$_{KRB\_PRIV}$	114 (35)	318 (35)
KINK$_I$	65 (112 or 73)	81 (112 or 93)
KINK$_R$	73 (0)	93 (0)
PS$_{PUT,IPsec}$	40 (13)	164 (13)
PS$_{GET,IPsec}$	51 (17)	362 (17)

The bootstrap sequence described in Section 2.3 requires the following functions: KDCD, NTP, TGT, two TGSs, PSD$_{KRB\_PRIV}$, KINK$_I$, PS$_{PUT,IPsec}$ and PS$_{GET,IPsec}$ (see A through E of Fiture 1 or Figure 2 through Figure 5 for details). the processing time of the sequence is shown in 5 The values without parentheses mean net processing times and the values in parentheses mean waiting times.

Table 5 Processing time of the sequences on the device (msec)

	Initiator		Responder	
crypto H/W	w/	w/o	w/	w/o
Bootstrap	511 (282)	1472 (282)	-	-
Device-to-Device	165 (125)	621 (145)	73	93

Table 6 Processing time of IPsec on the device

crypto H/W	w/	w/o
IPsec ESP	4.8μsec/byte	300μsec/byte

4.3 Performance of the Device-to-Device Communication

As described in Section 2.4, the device-to-device communication has the overhead (see F through G of Figure 1 or Figure 6 for details). If the device is an initiator of the communication, the overhead is PS$_{GET,IPsec}$, TGS and KINK$_I$. If the device is a responder of the communication, it is only KINK$_R$. Once IPsec is established between devices, the performance of IPsec is one of the major factors.

The overhead of the sequence and IPsec are shown in Table 5 and Table 6. The values without parentheses mean net processing times and the values in parentheses mean waiting times. The throughput of IPsec ESP inbound is omitted because both performances are nearly the same.

5. Considerations

5.1 Object Code Size

Object code size is improved from the previous prototype [13], i.e. from 419K bytes to 272K bytes. Kerberos related implementations(Kerberos and KINK) contributes the improvement. The previous implementation was ported from MIT implementation whereas the current one is our original code which simplify program of encoding/decoding messages, especially ASN.1. To reduce the current code, it is necessary to shrink the IP part which occupies about 50% of the entire code. Hardwired functions, such as off-load engine, may contribute the code size. It can be a further study item to shrink the object code size.

5.2 Performance of the Bootstrap Sequence

The sequence takes 793 msec or 1754 msec with or without cryptographic hardware (see Table 5). It will usually take one or several days to start the entire system if the system is a large one due to starting subsystem by subsystem for making sure. If assuming to start a system described in Section 1 with device-by-device manner, i.e. seventeen thousand devices, within 24 hours, every device will have to start within 5 seconds on average. The actual margin is longer than the above time because of starting a system in a subsystem-by subsystem manner instead of device-by-device usually. Hence, the performances of the sequence can be acceptable.

We may have to consider for burst accesses to servers if the system has a large number of devices. For the device side, randomly delayed bootstrap can be a solution. However, the necessity and the validity are future study items. For the server side, redundancy can be a solution, i.e. the redundancies of KDCs and PSs. It is not difficult to make KDC redundant [14]. But it is a further study item for PS.

5.3 Performance of Device-to-Device Communication

The overhead of the initiator takes 290 msec or 766 msec with or without cryptographic hardware (see Table 5). The overhead happens when both the device starts and the lifetime of Kerberos's tickets or IPsec SAs is expired. The former case can be acceptable with the reasons in Section 5.2. The latter case should be considered because the response time of control network systems should usually be on the hundred microsecond order. However, the overhead can be acceptable if the lifetimes are long enough, e.g. days, weeks or months, and are tuned operationally. The overhead of the responder can also be acceptable because it is shorter than the initiator's. Another possible way is to introduce priority into the IP packet processing of a device. For example, the overhead described above will have less impact if the IP stack of the device has fast-path and slow-path, and application packets are assigned to fast-path and other packets including Kerberos and KINK are assigned to slow-path. This can be a further study item.

As an example of IPsec's throughput, the processing time for 1024-byte payload takes 5 msec or 307 msec with or without cryptographic hardware (see Table 6). Considering the response time, i.e. the hundred microsecond order, the former case is fast enough, but the latter is not. The performance can be improved with AES instead of 3DES if the device cannot introduce cryptographic hardware.

5.4 NTP Server

Kerberos assumes that the clock of each device is adjusted to the clock of the KDC. It is used for preventing from some replay attacks. This is why our proposed model uses NTP server. However, for example, a malicious NTP server can advertise bogus time to devices. If a difference between the clocks of the KDC and the device is more than the allowable clock skew, the KDC will reject any request from the device, then the bootstrap sequence will be failed.

[15] showed an extension that was able to relax the requirement of the time synchronization. The extension has already included into the specifications of both Kerberos and KINK. If the difference is greater than the skew limit, the KDC returns an error message that includes the difference. Consequently, the device stores the difference, and uses it to adjust the time to process further messages. Hence, the device can finish its bootstrap sequence.

If a system requires a trusted NTP server the device-to-device communication can be used additionally. First, the NTP server implements the proposed model described in Section 2. Second, the NTP server starts its service before any device starts. Third, a device can find the NTP server from PS, then can synchronize its clock again under security of Kerberos, KINK and IPsec after the device completes its bootstrap sequence.

6. Related Work

There are models of device's auto-configuration. This section shows differences between our model and others.

Jini [†] has the following features: 1) Engaged distributed object technology, e.g. RMI, CORBA, SOAP, can distribute service entities over networks. 2) Servers named Lookup Service manage objects named Proxy. A client has to load an appropriate Proxy when using a distributed service remotely. 3) Lookup Service, which is necessary for registering a Proxy by a service entity and for loading a Proxy by a client, can be discovered on demand with IP multicast. Jini may be suited to the purpose at which our model aims since both models can provide any required data to devices. The early version of Jini had security issues, e.g. [16], then Jini v2 [17] enhanced it. However, it is difficult to compare the actual security mechanism of Jini with that of our model because it is hidden away from the specification by Java Class. Hence, Jini's applicability to devices cannot be identified. Jini v2 introduced Trust Verifier by which a client can verity the integrity of a loaded Proxy. The idea of Trust Verifier can be useful for our model because our model can also provide program code to devices remotely.

SLP (Service Location Protocol) [18] provides locations of services as URLs. SLP servers, i.e. Service Agent and Directory Agent, can be discovered on demand with IP multicast. Cache servers named Directory Agent contribute scalability to SLP. Please note that SLP does not have any service entity itself. Our model has a service entity named PS which can provide any required data, e.g. application program and data, under a secure environment. Hence, to examine if SLP can be used in our model is appropriate rather than to compare both models equally. SLP may be applicable to KDCD instead of DHCP.

[†]Jini is a trademark of Sun Microsystems, Inc.

UPnP [††] (Universal Plug and Play) [19] uses SOAP (Simple Object Access Protocol), HTTP and TCP for control messages and GENA (General Event Notification Architecture), HTTP and TCP for notification messages whereas existing control networks usually use UDP for those purposes. This means that they will have to be changed if introducing UPnP. Hence, the goal of UPnP is different from our model because one of our goals is to minimize the impact on them when introducing an auto-configuration mechanism. Public key cryptography is mandated for UPnP. So UPnP's applicability to devices is also different from our model.

7. Further Study Items

7.1 Sharing Mechanism of a Kerberos's Secret Key

Kerberos assumes that each device shares their Kerberos's secret keys with KDC. However it is important for control networks how to share a secret key between a device and KDC because there can be a huge number of devices whose user interfaces are limited in a system. There are two approaches to share a secret key. One is to ship a device with a secret key. The user reads the secret key, then sets it to KDC. The other is on-site installation. The user creates and installs a secret key into the device. However, we have to consider those approaches further because of their insufficiency as described below.

For the former approach, it assumes that KDC has a pair of a public key and a private key, and KDC is specified before a device is shipped. Then it will be possible to ship the device with a Kerberos's secret key and a KDC's public key. Consequently, the device can provide the secret key which is protected with the KDC's public key. For example, this technique may be suited to small embedded devices mentioned in Section 1 if using low exponent RSA because the operations of encryption and verification are cheap while the operations of decryption and signature are expensive. However, this technique has the following restrictions. 1) A device requires a communication channel to read the secret key encrypted with the KDC's public key. 2) The channel is out-of-band from the proposed model described in Section 2. 3) Readout is done with device-by-device manner. 4) A device has the KDC's public key when shipping. Please note that the secret key can be updated with Kerberos's administration command ,e.g. kpasswd, remotely after finishing sharing the secret key.

For the latter approach, [20] proposed resurrecting duckling security policy model where a weak device can establish a master-slave relationship with other device securely under ad-hoc environment. In ad-hoc environment, a device have to assume the absence of an online server. Figure 7 shows a summary of resurrecting duckling security policy model. A device called a *duckling* has two states, i.e. *imprint-able* and *imprinted*, and the initial state is *imprint-able*. When other device called a *mother duck* sets a shared key called an *ignition key*, the *duckling* changes the state to *imprinted*, then obeys the *mother duck*. When the *duckling* deletes the *ignition key* by the *mother duck*'s order, the *duckling* changes the state to *imprint-able*, then is ready to accept a new *mother duck*. If a writer which installs a Kerberos's secret key into a device is a *mother duck* and if a device where the secret key is installed is a *duckling*, the Kerberos's secret key will be installed into the device under ad-hoc environment such as an installation field. It may be better to separate the Kerberos's secret key from the *ignition key*. However, this technique has the following restrictions. 1) A device has a communication channel with a writer. The channel have to provide confidentiality and integrity by itself, e.g. electrical contact rather than wireless, because an *ignition key* from a *mother duck* to a *duckling* is plain text (see [20] for details). 2) The channel is out-of-band from the proposed model described in Section 2. 3) Installation is done with device-by-device manner.

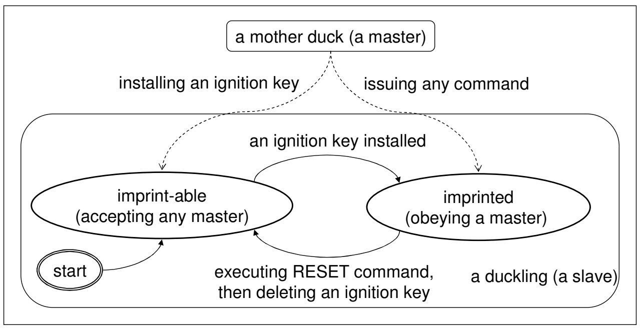

Fig. 7 Resurrecting duckling security policy model

7.2 Kerberos's Inter-realm Operation

It is common for management to divide a large system into small manageable domains, which are called realms in the manner of Kerberos. In the case of Kerberos, inter-realm is the technique to federate operational realms by sharing a secret between KDCs (see Figure 8). However, inter-realm has issues when being applied to control networks as follows.

- inter-realm costs a device when the device speaks to another device which belongs to a different realm (see Figure 9). 1) The device has to traverse realms until it reaches the destined realm. 2) The device has to find a path to traverse realms. It is the device's responsibility to find the next hop

[††]UPnP is a trademark of the UPnP Implementers Corporation.

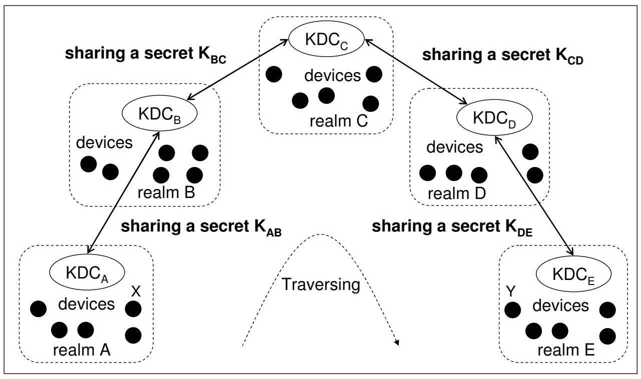

Fig. 8 An example of inter-realm

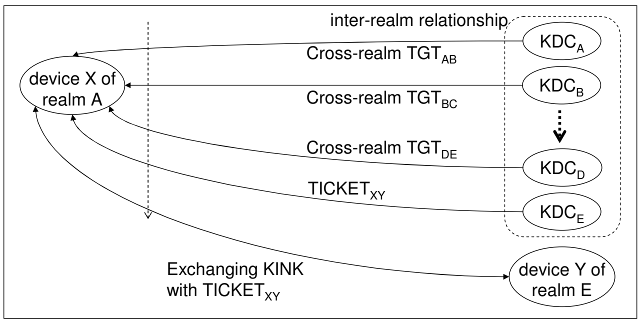

Fig. 9 An example of traversing realms

of KDC. 3) The device has to acquire special tickets, which are called cross-realm TGT, to traverse realms. The number of tickets to be acquired are in proportion to the number of realms to be traversed. These can be expensive for small embedded devices.

- Host centric fashion mentioned above can be a cause of inconsistency if a system has a huge number of and a variety of devices. For example, it is the device's responsibility to find the next hop of KDC. However, compatibility of device's behavior about finding path is not obvious because the specification of Kerberos does not define about traversing realms precisely. Consequently, the device's behavior depends upon its implementation.
- Inter-realm is formed by chaining KDCs. The device will be unable to traverse KDCs even if one of the intermediate KDCs is unavailable.

PKINIT [21], PKCROSS [22] † and KDC referrals [23] which are under discussion in IETF may be able to solve some of the above issues.

8. Conclusion

Through implementing the model experimentally, this paper shows the practicability of our proposed model which is intended to solve the issues, i.e. security and configuration complexity, while satisfying the restrictions, i.e. small embedded devices, isolated networks, private name space/naming system and inheriting the property. Secure bootstrap sequence and device-to-device communication using the chain of trust are the points of the model.

There are several further study items. The first is to shrink the IP protocol stack. The object code size of the prototype device achieves 270K bytes. It will be necessary to shrink the IP protocol stack if smaller code is required. The second is priority processing in the IP protocol stack. The device's performance of the bootstrap sequence and device-to-device communication can be acceptable if the lifetimes of Kerberos's ticket and IPsec SAs are turned operationally, and if cryptography is implemented reasonably, i.e. cryptographic hardware or faster algorithm than 3DES in software. However the overheads can not be zero. Hence, priority processing in the IP protocol stack, such as fast-path and slow-path, may reduce influence of the overheads. The third is redundancy of the servers, i.e. KDC and PS. The fourth is the mechanism of sharing a Kerberos's secret key between a devince and KDC. The fifth is to optimize inter-realm operation for small embedded devices.

†[22] has already expired. However, PKCROSS can revive after PKINIT is completed because PKCROSS depends upon PKINIT.

References

[1] Fieldbus Foundation, FF-581-1.3, FOUNDATION Specification: System Architecture, 2003.
[2] PROFIBUS International, IEC 61158, Digital Data Communication for Measurement and Control - Fieldbus for Use in Industrial Control Systems, 1999.
[3] MODBUS.ORG, Modbus Application protocol V1.0, 2002.
[4] ASHRAE, ANSI/ASHRAE Standard 135-1995, BACnet A Data Communication Protocol for Building Automation and Control Networks, 1995.
[5] N. Okabe, "Issues of Control Networks When Introducing IP," The 2005 Symposium on Applications and the Internet Workshops (SAINTWKSP 2005), Jan. 2005.
[6] N. Okabe, S. Sakane, K. Miyazawa, K. Kamada, A. Inoue, and M. Ishiyama, "Security Architecture for Control Networks using IPsec and KINK," The IEICE Transaction on Communications, vol.J88-B, pp.1910–1921, Nov. 2005.
[7] A. Inoue, M. Ishiyama, K. Kamada, S. Sakane, and N. Okabe, "A Secured Autonomous Bootstrap Mechanism for Control Networks," Annual Review of Communications, Volume 57, International Engineering Consortium, Nov. 2004.
[8] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol." RFC2401, 1998.
[9] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)." RFC2409, Nov. 1998.
[10] S. Sakane, K. Kamada, M. Thomas, and J. Vihuber, "Kerberized Internet Negotiation of Keys (KINK)." RFC4430, March 2006.
[11] C. Neuman, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)." RFC4120, July 2005.
[12] K. Kubo, J. Murakami, and T. Hoshi, "Hybrid Peer-to-Peer System for Network Monitoring of Field Devices," SICE Annual Conference 2003, Aug. 2003.

[13] N. Okabe, S. Sakane, M. Ishiyama, A. Inoue, and H. Esaki, "A Prototype of a Secure Autonomous Bootstrap Mechanism for Control Networks," The 2006 Symposium on Applications and the Internet (SAINT 2006), Jan. 2006.
[14] C. Kaufman, R. Perlman, and M. Speciner, Network Security: Private Communication in a Public World, ch. 10, Prentice Hall, 1995.
[15] D. Davis, D.E. Green, and T. Ts'o, "Kerberos with clocks adrift: History, protocols, and implementation," USENIX Computing Systems, Jan. 1996.
[16] P. Eronen and P. Nikander, "Decentralized Jini Security," Network and Distributed System Security Symposium (NDSS01), Feb. 2001.
[17] Sun Microsystems, Inc., Jini Specifications Archive - v2.1, Oct. 2005.
[18] E. Guttman, J. Veizades, and M. Day, "Service Location Protocol, Version 2." RFC2608, June 1999.
[19] UPnP Forum, UPnP Device Architecture 1.0, Version 1.0.1, 2003.
[20] F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks," Security Protocols, 7th International Workshop Proceedings, LNCS, pp.172–194, 1999.
[21] L. Zhu and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos." draft-ietf-cat-kerberos-pk-init-34, Feb. 2006.
[22] M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld, "Public Key Cryptography for Cross-Realm Authentication in Kerberos." draft-ietf-cat-kerberos-pk-cross-08, Nov. 2001.
[23] L. Zhu and K. Jaganathan, "Generating KDC Referrals to Locate Kerberos Realms." draft-ietf-krb-wg-kerberos-referrals-07, March 2006.