A Case Study of Secure ATM Switch Booting

Shaw-Cheng Chuang

Computer Laboratory
University of Cambridge
Pembroke Street
Cambridge CB2 3QG
United Kingdom
Email: shaw.chuang@cl.cam.ac.uk

Michael Roe

Computer Laboratory
University of Cambridge
Pembroke Street
Cambridge CB2 3QG
United Kingdom
Email: michael.roe@cl.cam.ac.uk

Abstract

This paper examines a few techniques for booting Asynchronous Transfer Mode (ATM) switches securely over an insecure network. Each of these techniques assume a different trust model. This work is being carried out in the context of the Fairisle ATM switch environment. In this environment we are envisaging an open multi-service network where ATM switches are booted with third party software, possibly using a third party booting service. Hence we are faced with an increased security threat, compared with a closed network environment, in ensuring that the switch has been booted with authorised and authenticated boot code. In this paper, we examines these threats and presents three schemes of countering the threats.

1 Introduction

Most Asynchronous Transfer Mode (ATM) switches do not run a software development environment. Switch software is developed on host machines before being loaded to the ATM switches. The loading process could take the form of replacing the boot PROM with a version that contains the new software. However, this is a laborious and time consuming process. Software development for the ATM switches requires this new software down loading cycle to be shorter and easy. One alternative approach is to receive the code from a boot server over an untrusted network, i.e. booting over the network.

The secure bootstrapping process of ATM switches is an area that has not been investigated in much detail. In a regulated telecommunication environment, switches can be viewed as under the control of a single administrative management of the network operator, though the network operator might delegate some of the operation and maintenance functions to the switch manufacturer. Switch software is developed and loaded either by the switch manufacturer or the equipment owner. In such an enclosed environment, it is relatively easy to make secure, though it has been reported in the literature of outsiders intrusion into such network.

Recent ATM trials have shown that ATM networks will be able to support the multi-media services envisaged as part of the Broadband Integrated Services Digital Network (B-ISDN) [1]. However, there is a lack of infrastructure support for the provision and management of services in such a multi-service network environment. This leads to the proposal of open multi-service networks [2] [3], which there can be multiple network providers working with multiple service providers (in a manner similar to the network operator and service provider relationship in an Universal Personal Telecommunication (UPT) environment). In such an open multi-service networks, services can be provided by any entity within or or attached to the network, including third parties. This approach corresponds to the separation of switching and service aspects in Intelligent Networks [4], in which the emphasis is to allow provision of more sophisticated services and the rapid deployment of services.

The increasingly de-regulated telecommunication market also leads to a lot of alliances between different network operators. In Europe it is the Open Network Provision (ONP) directive, stipulated by the European Commission, that forces a network operator to provide network services to a third party. In such an environment, the use of network resources such as switches can be shared between the partners of the alliance. Therefore there might be situations where the network operator and service provider in reality are competitors. It might therefore be desirable to let the network operator and service provider to achieve a level of assurance that the switches have been booted with the correct software. The network operator would like to maintain the high degree of availability traditionally associated with telecommunication network by retaining the ability to ensure only approval software is loaded.

The above discussions suggest the need to boot ATM switches with third party software, possibly using a third party booting service and this presents an increased security threat. In this paper, we present a case study of secure ATM switch booting. The discussion is based on the experience of a particular booting process of an ATM switch. In Section 2 this particular booting process is first examined. Threats in such booting process are then analysed in Section 3. A few techniques for countering the threats and booting ATM switches securely are then presented, with each of these techniques assume a different trust model.

2 ATM Switch Boot

In this section, we present a way that an ATM switch can be booted. This is based on the Fairisle ATM switch [5] environment developed in the Computer Laboratory, Cambridge University. In this paper, we are attempting to protect this environment and provide a secure booting mechanism.

The Fairisle ATM switch consists of port controllers which are analogous to line cards in a telephone exchange. The main function of the Fairisle port controller (FPC) are to map virtual circuit identifiers, manage queues, select priority, select routeing tags, and deal with blocking in the switching fabric. Each of these Fairisle port controllers is based on a simple high speed RISC processor and runs a micro-kernel called Wanda. A version of Wanda is held in PROM. On reset, the software in the PROM attempts to obtain the latest version of the software from a boot server. The booting process is as illustrated in Figure 1.

One of the key services that the booting process uses is the Reverse Address Resolution Protocol (*RARP*) service. This provides an address lookup service to allow a stateless entity to discover its network address using some local hardware feature or a FPC's slot position in a switch. The "*RARP*ing" process make use of an ATM meta-signalling protocol [6, 7]. Note that this use of ATM meta-signalling protocol is local to the Computer Laboratory environment. The meta-signalling protocol only operates over the Meta-signalling Virtual Circuit (MSVC), which is permanently configured. The MSVC operates on promiscuous mode: there can be multiple senders and multiple receivers. As cells from different senders may be interleaved on a promiscuous circuit, all messages sent must fit into a single cell. This proves to be a constraint on the kind of operation we can perform over the meta-signalling protocol.

The boot process starts with the FPC sending a *RARP* request over the MSVC from all its network interfaces (transmission and switching fabric), since it does not know the location of its *RARP* daemon (*RARPD*). Upon receiving the *RARP* request, the *RARPD* lookups the configuration table, and constructs a *RARP* reply. The reply includes the address of the FPC and the location of its boot server.

On the Fairisle network, the *RARPD* can be configured to use either local hardware feature (such as the unique Id in the Identity PROM for the FPC) or its slot position in a switch. *RARP*ing by slot location is particularly useful in a situation where a group of FPCs are booted as part of a switch configuration. In this situation, one of the FPCs, which is connected to another switch, is designated the master FPC and will be booted first, using its hardware feature. This master FPC will then be booted with the *RARPD* for the switch that it is in. Subsequent FPCs are then booted and assigned their network address according to the respective slot position in the switch. Note that since a switch contains several processors, each of these needs to be booted in order to boot a switch. Logically the switch is only considered completely booted when all the FPCs within the switch are booted.

Given that the FPC now knows its own address, it then sets up a virtual channel to the boot server and a request for its boot image (based on its address) to be loaded over this virtual channel.

3 Environment

In this paper, we use the term *owner* to refer to the owner of the ATM equipment and the term *programmer* to refer to the software developer or the service provider that has provided the boot code to be loaded. For the above described booting process, we made a few assumptions about the environment:

- No one is sitting at the switch console to verify the version of the loaded software and perform the monitoring of the booting process. We however have a "network console" that provides some output.
- No keyboard input is available for activities such as typing in a password.

  In both of the above cases, we are considering the situation where the switches might be located

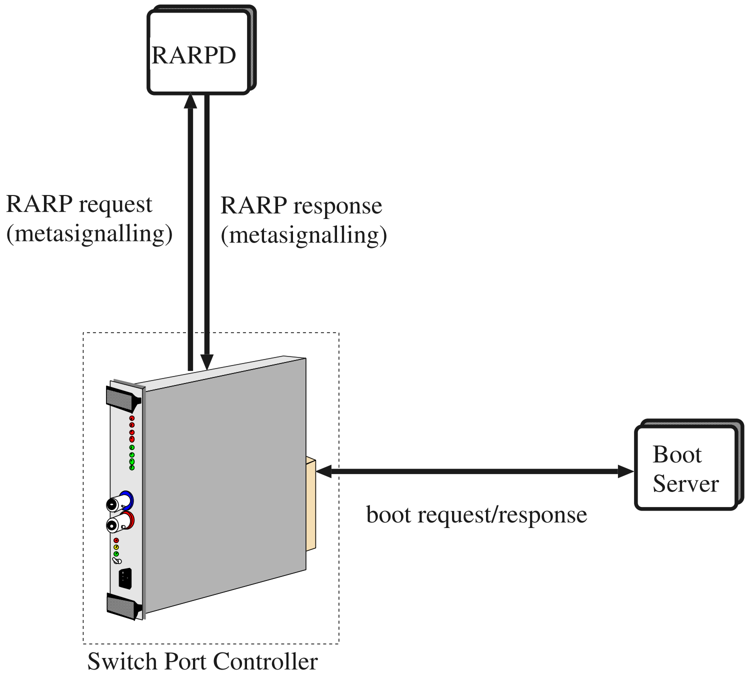

Figure 1: Fairisle booting

physically away from either the network operators or the software developer.

- The FPC is able to keep some secret, e.g. a Data Encryption Algorithm (DEA) [8] key. We want to minimise the amount of secret kept.
- We want to avoid keeping a universal key, such as the public key of a central authority. This help us to avoid updating all the FPCs in the situation where the universal key is revoked.
- The FPC does not maintain a real time clock.
- Software need not be written by the *owner*.
- *Owner* cannot trust the *programmer* to produce 100% correct code.
- *Owner* knows the secret in the FPC.
- It is the *owner*'s responsibility to maintain the correct mapping from the network address to the FPC specific secret.
- *Programmer* cannot be trusted with the secret.
- Each *owner* can own multiple FPCs. There can also be multiple mutually suspicious *owners* in the same network.
- There can be more than one *programmer* interacting with each *owner* and vice versa, i.e. there is a many-to-many mapping between *owners* and *programmers*.

3.1 Threats and protections

Based on the booting scenario described in the above section, the preliminary set of threats and protections that we envisage include the following. As we will show in later section, some of these can be relaxed while still achieving the effect of secure booting.

1. RARP information freshness

   We want to prevent *RARP* responses from being replayed.

2. RARP information modification

   We want to prevent a Trojan *RARPD* from generating false responses to a legitimate *RARP* request. A false response could cause either the FPC to take on a wrong address or wrongly booting from a, possibly Trojan, boot server.

3. Boot server authenticity

   Given that we have obtained the correct address and boot server, we want to make sure that the boot server that we have connected to is that which it claims to be, i.e. we want the authenticity of the boot server.

4. Boot code freshness

   We want to prevent boot code being replayed.

5. Unauthorised boot code

   We want to make sure that the boot code we are loading is authorised by the *owner* of the switch.

6. Boot code modification

   We want to make sure that the authorised boot code has not been modified.

7. Boot code confidentiality

   There are situations that the *owner*/*programmer* would like to maintain the confidentiality of the boot code. This is particularly a concern in a competitive commercial environment where one service provider might want to prevent competitors from gaining an advantage by reverse-engineering the boot code.

4 Secure Booting - Scheme 1

In this section, we describe a secure booting scheme that counters the threats addressed in Section 3. The trust model is as follows:

- *RARPD* is trusted and *RARPD* knows the secret that is embedded in each of the FPCs.
- *Owner* authorises the code to run and he knows the secret in the equipment that he owns.
- The boot server is trusted to function correctly once it is authorised by the *owner*. A misbehaving boot server will be revoked by the *owner*.

In this scheme, we will try to protect the meta-signalling protocol to ensure *RARP* information integrity and authenticity. A symmetric cryptographic algorithm is used due to the size limitation of the meta-signalling cell. Let's assume for the moment that the Data Encryption Algorithm (DEA) is used and the secret that the FPC keeps is a DEA key. We also want to minimise the use of this secret to frustrate cryptanalysis.

4.1 How does it work ?

The working of this scheme can be divided into four phases:

- Phase one - Certificate preparations

  This is the preparation stage that is done once for each *owner*-boot server binding and should happen before the actual boot process. The *owner* seal [1] two public key certificates [2] using the DEA key, one for *owner* public key and the other for the boot server public key.

- Phase two - Preliminary authorisation and signing of boot code

  The *programmer* submits his code to the *owner* to be authorised and the *owner* signs the boot code with his private key. This "signed" boot code, together with the two public key certificates are then deposited with the boot server. The *owner* also generates and encrypts a new public key pair and a new symmetric key using the DEA key. If confidentiality of the boot code is required, the boot code will be encrypted using this new symmetric key. This process is done once for each of the new boot code submitted. The new public key pair is assigned to this FPC and only intended for later operational use *after* the switch is booted.

- Phase three - *RARP*ing

  In this phase, the actual boot process starts. The FPC is reset and the booting FPC generates the *RARP* request as normal. The *RARP* request also includes a nonce as the challenge to the *RARPD*. Since the FPC does not maintain a synchronised clock, challenge-response is used to ensure the freshness of the *RARP* run. As the *RARPD* shares the DEA key with the FPC, it can construct a *RARP* reply that includes the response to the challenge and a corresponding Message Authentication Code (MAC), calculated using the DEA key. Upon receiving the *RARP* reply, the FPC checks the MAC in the response. If the check fails, the machine repeats the *RARP*ing process. If successful, the FPC can be sure that it has obtained correctly its own address and boot server address.

- Phase four - Boot code retrieval

  This phase involves connection to the boot server. Again a challenge-response protocol is used to ensure the freshness. The FPC sends a challenge

[1] This is a symmetric key Message Authentication Code (MAC) and not a digital signature. In this paper, we use the term "seal" to imply integrity protection using symmetric key and the term "sign" means using public key.

[2] This contains public key whose integrity is protected by a DEA key.

with the boot request. The boot server signs the response, the two public key certificates, the encrypted new keys and the boot code with his private key. The boot server then sends this signed message back to the FPC.

With the signed message, the FPC is able to perform the following checks:

1. Verify the challenge-response to ensure freshness.
2. Use the *owner* public key certificate to retrieve the *owner* public key and verify the authorisation of the boot code.
3. Use the boot server public key certificate to retrieve the boot server public key and verify that the boot code is indeed sent by the right boot server.

The protocol runs is as below (with notations as shown in Table 1):

For Phase three - *RARP*ing

$$F \rightarrow R : N_{F1}, H_F, S_F$$

$$R \rightarrow F : \{f(N_{F1}),\ ID_F,\ ID_B,\ VN_B\}_{K_F}$$

For Phase four - Boot code retrieval

$$F \rightarrow B : N_F, ID_F$$

$$B \rightarrow F : \begin{array}{l} \{f(N_F),\ ID_F,\ \{ID_F, O, K_O\}_{K_F}, \\ \{ID_F, ID_B, VN_B, K_B\}_{K_F}, \\ < K_{ST}, K_S, K_S^{-1} >_{K_F}, \\ [\{code\}_{K_O^{-1}}]_{K_{ST}}\}_{K_B^{-1}} \end{array}$$

Notes:

1. The encryption with K_{ST} on

   $$[\{code\}_{K_O^{-1}}]_{K_{ST}}$$

   is only necessary if boot code confidentiality is needed.
2. There should be no danger for *f(N_F)* to return N_F

There are few points worth noting in the above protocol. Notice that the *owner* is only involved in the first two phases and need not be online. Besides that, the secret is only used at the *RARP* process over a very small amount of data and for signing the two certificates. The two certificates change infrequently and remain the same over many boot sequences. This makes cryptanalysis more difficult.

Notation	Meaning
O	Owner
R	*RARP* daemon
B	Boot server
F	Fairisle port controller
V	Online verifier
ID_F	Address of the booting FPC
ID_B	Address/ID of the boot server
VN_B	Boot server version number
K_F	The secret DEA key in the FPC
H_F	FPC local hardware ID
S_F	FPC slot position
f(x)	A function for generating the response to a challenge
N_F, N_{F1}	Nonces used as challenge
K_O^{-1}, K_O	*Owner* private and public key
K_B^{-1}, K_B	Boot server private and public key
K_S^{-1}, K_S	New public key pair for the FPC
K_{ST}	A symmetric key for boot code confidentiality and for generating random number (see below)
$\{msg\}_K$	The message, *msg* is signed with key K
$[msg]_K$	The message, *msg* is encrypted with key K
$< msg >_K$	The message, *msg* is both integrity and confidentiality protected with key K
H(msg)	Hash function of the message

Table 1: Notations

For efficiency reasons, the order of data items in the protocol's second message (from boot server to FPC) has been carefully arranged so that the various checking processes can be performed in a pipeline fashion. The response and the address of the booting FPC are the first two items in the message to allow early elimination of an invalid boot reply. The public key certificates and the newly generated key are placed before the actual boot code so that the validation and optionally the decryption process can start as soon as the boot code is received.

The *RARPD* in this scheme also serves the role of the revocation server for the certificates. Notice that there is no timestamp on the public key certificate, since there is no real time clock available on the FPC to validate any timestamp (we shall recall that the challenge-response protocol was used for precisely this reason). Once a certificate for a boot server is been given out, we cannot revoke a certificate by time expiry. However we should be aware that the *owner* controls the *RARPD* configurations. For the *owner* to revoke the validity of a particular boot server, the *owner* can simply remove the boot server entry from the *RARPD* configurations. The replacement boot server can take on a new version number.

In this scheme, we are assuming that the initial boot code in the PROM is capable of performing both symmetric and public-key cryptography.

We use a nonce in the above protocol for the challenge and response. However experience tells us that it is difficult to generate good random numbers for nonces, especially without random number generator hardware support. One solution to alleviate the problem is to have a counter, C, in non-volatile memory and use K_{ST} (a symmetric key) to generate the random number.

i.e. $N_S = K_{ST}(\mathrm{C})$

Different K_{ST} can be loaded each time with the boot code, as shown in the above protocol runs.

As we have indicated before in the assumption, the *owner* cannot trust the *programmer* for producing completely correct code, especially, the *owner* cannot be sure that the *programmer* will not try to leak the secret in the FPC. Hence we need to prevent accidental or malicious leakage of K_F by the secondary boot code. One of the solutions to this problem is to have a read-once mapping for K_F and arrange the primary boot code to erase K_F from memory before starting the secondary code.

4.2 Weaknesses

The weakness of this scheme mainly lies in its failure modes. In this scheme, we are heavily relying on the *RARPD* being able to be trusted with the FPC's secret. The *RARPD* is typically running on a FPC as an application, and the FPC could be in a remote site. As a result *RARPD* is vulnerable to attacks of all forms. If one of these *RARPD* is compromised, then all FPCs whom it has been entrusted with the secret can be loaded with illegal code.

The other shortfall is the reliance of the *RARPD* on reliable "neighbours". As we have described in Section 2, when a switch boots, the master FPC (which later runs the *RARPD* for the switch later) needs to rely on a neighbouring *RARPD* to boot. This *RARPD* however might belong to a different administrative domain. Since the *RARPD* needs to possess the master FPC's secret, that implies disclosing the master FPC's secret to a party whose trustworthiness can not be ensured. This problem can be overcome by having those FPCs on the administrative domain border to be served by trusted *RARPD* on the Ethernet interface (the FPC can have a Ethernet interface).

5 Scheme 2

In scheme 2, we addresses the weaknesses identified in Section 4.2 and we have a different trust model. The *RARPD* no longer keep the secret but instead serve as a redirector to a master trust daemon. This is illustrated in Figure 2. This master trust daemon keeps the FPC secret and it needs not run on the switches themselves. The *owner* controls this daemon and we can have one master trust daemon per FPC, running on a secure machine (possibly physically secure in the *owner*'s premises). The number of master trust daemons to run is a matter of configuration. The *RARPD* maintains a table of the master trust daemon for each FPC that it is serving. The master trust daemon is also configured with the *RARPD* that should be serving the corresponding FPCs.

In scheme 2, the booting process looks exactly the same for phase one, two and four of Scheme 1 as described in Section 4. In phase three, the booting FPC again generates the authenticated *RARP* request. However, now the *RARPD* is not able to verify the request since it does not possess the secret. There are two possible variations of the behaviour of the *RARPD* and the master trust daemon at this point:

- The *RARPD* maintains the *RARP* information. However since it does not possess the secret to construct the *RARP* reply, it forwards the *RARP* reply information to the master trust daemon for the MAC to be calculated. The reply from the master trust daemon is then used to complete the meta-signalling *RARP* reply message.

- The *RARPD* acts purely as a relay and does not possess any *RARP* information. The master trust daemon constructs the *RARP* reply and the *RARPD* will further relay it back to the booting FPC. Since the master trust daemon possesses all information and is in the control of the *owner*, the *owner* has absolute control over the FPC.

Notice that in both variances, the channel between the *RARPD* and the master trust daemon is authenticated and is checked against the respective configuration information.

6 Scheme 3

In this section, we will re-examine the threats and protections involved in providing a secure booting service. The ultimate aims of secure booting are to en-

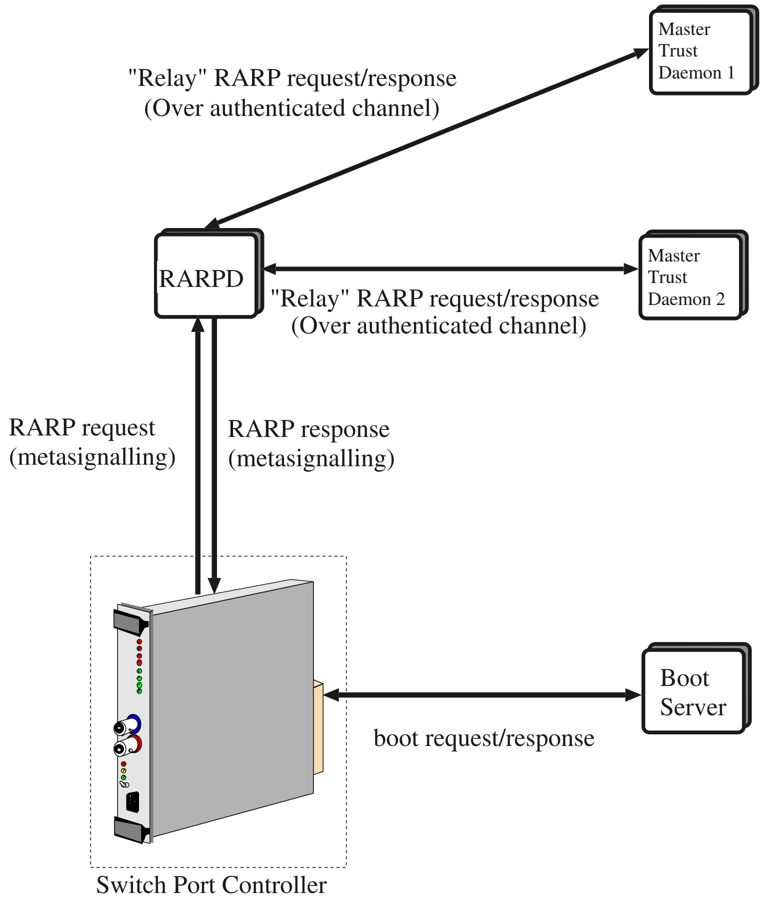

Figure 2: Scheme 2 booting

sure the boot code's integrity, authenticity and authorisation. The protection of boot server authenticity and *RARP* information integrity and authenticity are just the preliminary steps that can be used to achieve our ultimate aims. However in this section we show that we can still reach our target of secure booting with more relaxed protection requirements on the boot server and *RARP*.

An alternative approach to avoid the total trust required on the *RARPD* is to remove the *RARPD* from any process that requires the secret in the FPC. However in this case we might not be able to ensure the integrity of the *RARP* information, i.e. there is no protection of *RARP* information integrity and authenticity as indicated in Section 3. Indeed, a form of denial of service attack can be made by having malicious *RARPD* that always respond with incorrect information. Of course, the FPC boot PROM can perform a more intelligent filtering of the *RARP* replies and hope to eliminate the malicious replies. The FPC can detect a wrongly given address since the boot code being given by the boot server will not be correctly verified with the secret in the FPC. However if the wrong boot server is given [3], and with the phase two operation described in Section 4, the malicious boot server can replays wrong version of the boot code. This is the consequence of not having a revocation server (*RARPD*) for the certificates.

The remedy to the problem is to introduce an online verifier for the boot server. Similar to the approach in scheme 2, the boot server becomes the redirector to the online verifier. The boot server replay the challenge to the online verifier. If the boot server is indeed the correct one for the FPC, then the response sequence will be signed and returned to the boot server.

[3] Note that the boot server address is given in the *RARP* reply and a *wrong* boot server can be a Trojan boot server who possess an earlier but invoked boot server certificate.

The boot server can then return boot code together with the response back to the FPC. This is illustrated in Figure 3.

As we recalled in Scheme 1 and 2, the *owner* performs an off-line authorisation of boot server and authentication of the boot code. The boot server ensures the freshness of the boot code. In Scheme 3, the online verifier is needed to ensure the authenticity and the freshness of the boot code. The authorisation of the boot code can still be done on an off-line basis.

We are trusting the online verifier to examine all down loading software. The authority that owns the online verifier might be the network operator. So it might be reasonable that they examine all code to be loaded into their machines.

In a real-world configuration, the online verifier is under the control of the network operator (since only the network operator possesses the secret K_F) and the service providers can control the boot servers.

There can be two variants of this scheme, depending on the boot server's level of involvement.

The boot server can act purely as a relay between the FPCs and the online verifier. The online verifier needs to store the boot code and perform real-time sealing and encryption if necessary, of the boot code. This real-time processing is only necessary if we use a different K_{ST} for every boot.

$$F \rightarrow B : N_F, ID_F$$

$$B \rightarrow V : N_F, ID_F$$

$$V \rightarrow B : \begin{array}{l} < ID_F,\ f(N_F),\ K_{ST},\ K_S,\ K_S^{-1} >_{K_F}, \\ < code >_{K_{ST}} \end{array}$$

$$B \rightarrow F : \begin{array}{l} < ID_F,\ f(N_F),\ K_{ST},\ K_S,\ K_S^{-1} >_{K_F}, \\ < code >_{K_{ST}} \end{array}$$

Note that the encryption, $< code >_{K_{ST}}$, is only necessary if boot code confidentiality is needed. Otherwise it should be $\{code\}_{K_{ST}}$. This also applies to the rest of this document.

Notice that it could be argued that the *RARPD* could point directly to the online verifier since the boot server acts merely as a relay unit in the above protocol. However, the boot server can be controlled by the service provider or a third party boot service and hence this arrangement allows a check to ensure that the designated boot code is being sent to the FPC. This is particularly important in the environment where the service provider and network operator in reality are competitors.

The above protocol involves the passing of boot code data between the online server and the boot server. This extra message passing can be avoided if the online server is merged with the boot server (this however does imply that the combined server is no longer a third party boot server). We thus have a simplified case of:

$$F \rightarrow Server : N_F, ID_F$$

$$Server \rightarrow F : \begin{array}{l} < ID_F, f(N_F), K_{ST}, K_S, K_S^{-1} >_{K_F}, \\ < code >_{K_{ST}} \end{array}$$

The second message includes some known plaintext, $(ID_F,\ f(N_F))$, encrypted under the key K_F. In order to minimise the use of the key K_F to frustrate cryptanalysis, the message could be transformed to be:

$$Server \rightarrow F : \begin{array}{l} < K_{ST},\ K_S,\ K_S^{-1} >_{K_F}, \\ \{f(N_F),\ ID_F\}_{K_{ST}}, < code >_{K_{ST}} \end{array}$$

We can also further increase the difficulty of cryptanalysis by employing collisionful hash function as described in [9] [10] when we are providing the integrity protection. Collisionful and collision-free hash functions are not strictly opposites, though it is convenient to maintain that the fiction that they are opposite since no function may be both collisionful and collision-free. If the function *g(k, msg)* is collisionful, it exhibits collisions in the parameter *k* but not in the parameter *msg*. For a value of *msg* and the result of applying the function *g()*, there are many values that k could have taken. However, for a particular value of *k*, there is only one value *msg* that could have produced the same result (or it is computationally infeasible to find other values). Therefore if the sealing of messages in our protocol is done using collisionful hash function, then an attack on the message could find one of the possible values of key consistent with that message. If the attacker then try to forge a replacement message, then the chances are that the key will be incorrect and the forged message thereby detected.

The other way that the extra message passing can be avoided if the boot code is kept with the boot server. The programmer goes to the owner with the boot code and the owner seals and optionally encrypts the boot code. The boot code is thereafter stored in the boot server, however, the boot server is not in possession of the key K_F which is used to decrypt the token where key K_{ST} is included. K_{ST} is the key which is used to seal and optionally encrypt the boot code.

The protocol runs is as below:

$$F \rightarrow B : N_F, ID_F$$

$$B \rightarrow V : \{N_F,\ ID_F,\ N_B,\ ID_B\}_{K_B^{-1}}$$

$$V \rightarrow B : \begin{array}{l} \{f(N_B), ID_B, \{ID_F, f(N_F), ID_O, K_O\}_{K_F}, \\ \{f(N_F), ID_F, H(code), \{ID_F, ID_B, K_B\}_{K_F}, \\ < K_{ST}, K_S, K_S^{-1} >_{K_F}\}_{K_O^{-1}}\}_{K_O^{-1}} \end{array}$$

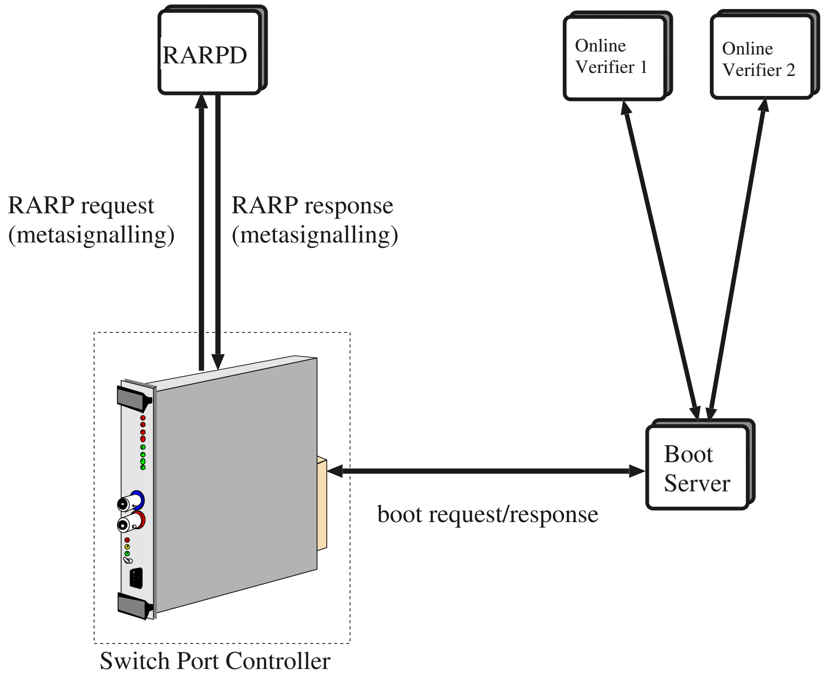

Figure 3: Scheme 3 booting

$$B \rightarrow F: \begin{array}{l} \{\{ID_F, f(N_F), ID_O, K_O\}_{K_F}, \\ \{f(N_F), ID_F, H(code), \{ID_F, ID_B, K_B\}_{K_F}, \\ < K_{ST}, K_S, K_S^{-1} >_{K_F}\}_{K_O^{-1}}, \\ < code >_{K_{ST}}\}_{K_B^{-1}} \end{array}$$

However this protocol can be greatly simplified since the authenticity of the boot server does not affect the boot code's integrity, authenticity and authorisation.

The simplified protocol is as:

$$F \rightarrow B: N_F, ID_F$$

$$B \rightarrow V: N_F, ID_F$$

$$V \rightarrow B: \begin{array}{l} < K_{ST}, K_S, K_S^{-1} >_{K_F}, \\ \{ID_F,\ f(N_F),\ H(code)\}_{K_{ST}} \end{array}$$

$$B \rightarrow F: \begin{array}{l} < K_{ST}, K_S, K_S^{-1} >_{K_F}, \\ \{ID_F,\ f(N_F),\ H(code)\}_{K_{ST}}, \\ < code >_{K_{ST}} \end{array}$$

7 Related Work

The secure booting of networked workstation is described in [10]. It assumes the workstation contains no cryptographic secrets and relies on user to be at the workstation console to detect the loading of incorrect boot code. This is different from our assumption of unattended booting event where network operator need not be at the switch console.

A related area of work is on secure distribution of software. One example of such system is the Bellcore's Trusted Software Integrity (Betsi) System [11] which deals with the trusted distribution of freeware in the internet. However, booting is *not* the same as secure software distribution mainly because we have very little local state with booting. Therefore we need to design a secure protocol that doesn't need much local state. With software distribution, the machine is already running and hence doesn't have this restriction.

8 Conclusion

We have presented three schemes of secure booting an ATM switch in this paper. In scheme 1, we attempt to protect both the *RARP*ing process and the connection to the boot server. Although this scheme has the advantage of not needing the *owner* to be online, the *RARPD* is acting on behalf of the *owner* and the scheme is vulnerable if the *RARPD* is attacked. This weakness is addressed in scheme 2 and 3 with the *owner* playing a more active role. In both of these schemes, the *RARPD* and the boot server can act as an proxy in the protocol. Scheme 2 and 3 also results in a simplified protocol. With the assumptions we made about the environment, we also observe that the secret on the FPCs needs to be shared and put online in all three schemes. Therefore it is important that the process that handles this secret is run at a secure machine, as shown in scheme 2 and 3.

Acknowledgement

The authors would like to thank colleagues of the Security Research Group and the System Research Group at the Computer Laboratory, Cambridge University for their valuable comments on this work. Those stimulating discussions with Mark Lomas, Richard Black and Paul Wernick have been most useful.

References

[1] ITU-T Recommandation I.211, "Integrated Service Digital Network (ISDN) Service Capabilities - BISDN Service Aspects." ITU-T, 1993.

[2] Korbus van der Merwe and Shaw-Cheng Chuang, "Support for Open Multi Service Networks," in *International Teletraffic Seminar on Teletraffic Engineering for Evolving Networks, Pretoria, South Africa*, September 1995.

[3] A. A. Lazar, S. K. Bhonsle and K. S. Lim, "A Binding Architecture for Multimedia Networks," in *Procceding of the Multimedia Transport and Teleservices, Vienna, Austria*, November 1995.

[4] James J. Garrahan and Peter A. Russo, "Intelligent Network Overview," *IEEE Communications Magazine*, pp. 30–36, March 1993.

[5] I. Leslie and D. McAuley, "Fairisle: An ATM Network for the Local Area," in *Computer Communication Review*, vol. 21(4), pp. 327–336, ACM SIGCOMM, September 1991.

[6] R. Black, "FDL Cell formats and Meta-Signalling," in *ATM Document Collection 3 (The* Blue *Book)*, ch. 5, University of Cambridge Computer Laboratory, March 1994.

[7] R. Black, "Common MSDL Features," in *ATM Document Collection 3 (The* Blue *Book)*, ch. 1, University of Cambridge Computer Laboratory, February 1994.

[8] ANSI X3.92, "Data Encryption Algorithm." American National Standard Institute, 1983.

[9] T.M.A. Lomas and B. Christianson, "To Whom am I Speaking ? Remote Booting in a Hostile World," *IEEE Computer*, pp. 50–54, January 1995.

[10] T.M.A. Lomas, "Collision-Freedom, Considered Harmful, or How to Boot a Computer," in *Procceding 1995 Japan-Korea Joint Workshop on Information Security and Cryptology*, January 1995.

[11] Aviel D. Rubin, "Trusted Distribution of Software Over the Internet," in *Procceding of The Internet Society Symposium on Network and Distributed System Security*, pp. 47–53, February 1995.