Distribution of Certificates, CRLs and Address Attestations in the Secure Border Gateway Protocol

by
Kavita Baball

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2002

Author ....
Department of Electrical Engineering and Computer Science
January 25, 2002

Certified by. .....
Karen Sollins
Research Scientist
Thesis Supervisor

Certified by... .....
Stephen Kent
Chief Scientist-Information Security, BBN Technologies
Thesis Supervisor

Accepted by ... .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Distribution of Certificates, CRLs and Address Attestations in the Secure Border Gateway Protocol

by

Kavita Baball

Submitted to the Department of Electrical Engineering and Computer Science
on January 25, 2002, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

The Secure Border Gateway Protocol, S-BGP, was designed to address security problems in the Border Gateway Protocol (BGP), the standard for inter-domain routing used on the Internet. S-BGP relies on the distribution of countermeasure information such as certificates, certificate revocation lists (CRLs), and route and address attestations (RAs and AAs). In order to provide the efficient distribution of this data, S-BGP makes use of several replicated, easy-to-access storage sites, or repositories. This thesis describes the design and implementation of one such repository. This prototype includes a database that is stored on an S-BGP server, interfaces to access and modify the contents of the database, and access control for the repository. It provides authentication mechanisms for all clients and replicated servers that access the repository. It also enforces authorization algorithms for submitting data to ensure that a client is allowed to upload the information sent to the repository. The repository supports two main types of data transfer: uploads of certificates, CRLs and AAs by the clients, as well as downloads of the information already stored in the database. Finally, servers synchronize on a timely basis to ensure that changes are propagated throughout the system.

Thesis Supervisor: Karen Sollins
Title: Research Scientist

Thesis Supervisor: Stephen Kent
Title: Chief Scientist-Information Security, BBN Technologies

Acknowledgments

I would like to acknowledge the following people without whom I could not have finished this thesis:

- My parents, Seree and Sandra Baball, and brother, Ravi, for their love, prayers and encouragement.
- My thesis advisor at MIT, Karen Sollins, for her help and support in finishing this thesis.
- The people at BBN Technologies: my thesis advisor Stephen Kent, Charlie Lynn, Karen Seo, Jean Duffy and the rest of the Information Security Department. In particular, I would like to thank Charlie for all of his time and guidance in getting this thesis done. I have learned a tremendous amount from him and am grateful for the opportunity to work with him.
- Janelle Prevost, Geoff Lee Seyon, Kevin McDonald and Keith Santarelli for reviewing drafts of this thesis and for their suggestions. A special thanks to my family for their input.
- Petros Boufounos, Ivan Nestlerode and Kevin for their help (and patience!) with my countless C questions.
- And to all my friends (especially Arin, Joanna and Janelle) for their support, understanding and threats. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the number of users on the Internet grows, the amount of data transmitted also increases. In order for this data to get to its intended recipient, it must be correctly routed. Internet routing is based on a distributed system composed of many routers grouped into management domains called Autonomous Systems (ASes). The Border Gateway Protocol (BGP) is the standard scheme used to distribute routing information between these autonomous systems through UPDATE messages. These UPDATEs are used to propagate changes made to the routing tables on the BGP routers throughout the network. Despite being a critical part of Internet infrastructure, the BGP protocol is highly vulnerable to a variety of attacks as it has no secure means of verifying the authenticity and legitimacy of BGP control traffic.

Secure Border Gateway Protocol

The Secure Border Gateway Protocol (S-BGP), an extension to BGP, was specifically designed by BBN Technologies to address the security problems associated with BGP. S-BGP adheres to the principle of least privilege[1] and uses countermeasures that create an authentication and authorization system that addresses most of the security problems associated with BGP. To facilitate adoption and deployment, S-BGP is designed to minimize the overhead (processing, bandwidth, storage) added by its

[1]This principle requires that a user be given no more privilege than necessary to perform a task.

countermeasures and to be interoperable with the current BGP so that it can be incrementally deployable on the Internet.

The S-BGP architecture uses three mechanisms to provide security. First, a Public Key Infrastructure (PKI) is used to authenticate the ownership of IP address blocks, AS numbers, identities of ASes and BGP routers and to determine whether or not a particular router is authorized to represent an AS. Second, a new, optional, BGP transitive path attribute containing *attestations* has been introduced. These attestations enable BGP speakers, when they receive route advertisements, to verify that each AS along the path has been authorized by the preceding AS to advertise the route, and that the originating AS has been authorized to advertise those prefixes. There are two types of attestations: route attestations, which are used to express a secure route, and address attestations, which contain information about the owners of address prefixes, and the ASes that are authorized to advertise these prefixes. Finally, IPsec [3] is used to provide data and partial sequence integrity for BGP peering sessions, and to enable BGP routers to authenticate each other for exchanges of BGP control traffic.

Security Mechanisms in S-BGP

The security approaches used by S-BGP rely heavily on the distribution of countermeasure information. We therefore need to design and implement a repository that supports efficient distribution of the certificates, certificate revocation lists (CRLs) and address attestations (AAs) in S-BGP. Each S-BGP speaker must have access to the public keys required to validate UPDATEs. An S-BGP speaker receives updates from all reachable ASes and needs certificates for the speakers in each of these ASes. This can amount to the full set of certificates encompassing all address space owners, AS owners, and some of the S-BGP speakers. Incorporating certificates in UPDATE messages is infeasible because of the limited size of UPDATE messages. Instead, the S-BGP architecture proposes to build a repository at the Network Access Points (NAPs). Since the NAPs are connected directly to the ISPs, they are accessible even without inter-AS routing and are good candidates for housing the repository. Addi-

tionally, processing the data at the Network Operations Centers (NOCs) simplifies the operations of the routers as the objects in the database can be validated and put in a more compact form before distribution to the S-BGP speakers.

S-BGP Server Repository

The goal of this thesis is to design and implement this repository, which is used to solve the S-BGP countermeasure distribution issues. The repository consists of server-replicated, easy-to-access storage sites. It handles the projected growth and usage of the Internet, and enforces access control based on the data source and the data submitted. The repository leverages off the existing Internet infrastructure and uses available technology that can be incrementally deployed.

The first stage of this project was the actual design of the repository. The repository needs to handle over 200,000 certificates, CRLs and AAs based on initial estimates. It also needs to store certification path objects, which are used to allow continued operation should some client, such as an operator at a Network Operation Center (NOC), prematurely delete a certificate in a certification path. If this occurs, objects that need this path for verification can still be uploaded or downloaded until they have been replaced by ones using a new path. The use of certification paths in the repository is discussed in Section 5.5.3. This requirement for storage was fulfilled by using a database that contains four tables, each of which holds objects of one of the following types: certificates, CRLs, AAs and certification paths. The repository must also allow for the secure download of these objects, as well as for the ability to upload new items and make changes to data already in the databases. There must also be a data replication mechanism to allow changes made at one site to propagate to the replicated copies of the repository.

The repository also has security requirements that need to be addressed. All accesses to the repository must be authenticated in order to ascertain that a NOC operator is who he claims to be, and the integrity of the data transferred must be verified. Furthermore, we also need to ensure that a client is authorized to upload the data submitted to the repository. All authenticated NOCs are authorized to perform

downloads from the repository.

Repository Structure

In order to provide these functions, we designed an Upload Processor, a Download Processor and a Synchronization Processor. The actual uploads and downloads are done by the NOC through a web interface. In order for a NOC to connect to this interface, it must have an SSL certificate issued under the S-BGP public key infrastructure to authenticate itself.

All upload files and reply files transferred to and from the repository are signed. This allows for any data corruption to be detected by performing the necessary signature validation. The Download Processor generates one file for each database table that contains all the objects of a given type, which the NOC can then retrieve. For uploads, the NOC provides a batch file which contains all the database transactions that it would like to perform. All the batch files are initially placed in an "Input" directory, from which they are sequentially retrieved by the Upload Processor. The signature on the file is then checked. If it is valid, the Processor extracts the individual transactions and runs an access control algorithm on them. It then performs the actual database function if the algorithm produced no error. This access control algorithm checks that the NOC operator is allowed to manage the object associated with the specific transaction. Finally, when a batch file is successfully processed, its filename is written to an Incremental Changes Table (ICT). The ICT is a file containing the names of all the files uploaded within a given time period. It is used during synchronization. During this process, the ICTs and upload files are copied between servers. Each server then goes through the ICTs in the order they were received, and processes the batch files listed in each ICT using the Upload Processor. NOCs can also download all of the objects in the database by retrieving the four database files that are generated by the Download Processor. Each file corresponds to one of the database tables and contains all of the objects stored in it.

The second stage was the implementation of the repository. This involved creating the PostgreSQL database that is used to store the objects, and setting up the directory

structure for the data transfers. It also included writing the software for the Upload, Download and Synchronization Processors, including the validation of the signatures on the files, as well as the access control checks for the uploads.

Thesis Outline

This thesis describes S-BGP and its advantages, as well as how the certificates, CRLs and address attestations are efficiently distributed in this system. Chapter 2 provides the reader with an overview of the BGP protocol and its vulnerabilities. Chapter 3 describes the architecture of the Secure Border Gateway Protocol and includes a discussion of the Public Key Infrastructures, the introduction of Attestations, and the use of IPsec. Chapter 4 describes the requirements and design of the repository, including its individual components and how data is transferred both to and from it. Chapter 5 presents the security aspects of the repository and the mechanisms used to fulfill them. Chapter 6 gives a more detailed description of the actual implementation. Chapter 7 presents possible additions to the repository, as well as the future of the S-BGP project, and concludes the thesis.

Chapter 2

Background

The Border Gateway Protocol (BGP) is the current standard for inter-domain routing. This chapter starts out with a discussion of routing in the Internet followed by a description of how BGP works. It discusses how BGP speakers store paths to a particular network in a routing table, and the use of UPDATE messages to advertise route changes. The vulnerabilities of BGP are then presented as a prelude to the motivation for the Secure Border Gateway Protocol (S-BGP) project.

2.1 Routing in the Internet

One important aspect in a large heterogenous internetwork such as the Internet is finding efficient paths to route information among the smaller component networks. Each of these networks consists of gateway nodes and host nodes. The hosts are computers that support users and run application programs, while the gateway nodes control traffic and route packets. Gateway computers forward packets to other gateway computers in different networks. For users to connect to the Internet they must connect to a router, which transmits the traffic to a gateway node if the destination is not in the same network. Gateway nodes use routing tables to route the packets to the different hosts. There are many schemes that node computers can use to route the packets to their destination. The Internet standard for inter-domain routing is the Border Gateway Protocol (BGP), which exchanges routing information between the

gateway nodes in the different routing domains, also known as autonomous systems (ASes). An autonomous system can be defined as an internetwork in which all the routers are under the same administrative control[10]. For example, Internet Service Providers (ISPs) and Downstream Providers (DSPs) are ASes. Currently, all ISPs use BGP, while only 10% of DSPs and approximately 5% of their customers (subscribers) are BGP users[1]. The others are singly connected to the Internet i.e., singly-homed and use "default" routing.

Figure 2-1 shows a simple BGP model, with both BGP and non-BGP routers. The Network Access Point (NAP) is a interconnection point in the Internet backbone which ties all the ISPs together by a high speed local area network (LAN). The NAPs provide network access for users. Most Internet traffic is handled using private peering arrangements between the largest ISPs, such as the direct connection between ISP 1 and ISP 2 in the figure. This connection is used not only for routing and other control traffic, but more importantly for data traffic.

2.2 Border Gateway Protocol (BGP)

The Border Gateway Protocol Version 4 (BGP-4), which is documented in RFC 1771 [13], is the protocol used to exchange routing information among different ASes in the Internet. Unlike distance-vector algorithms which keep track of particular nodes and the cost of their links to their neighbors, BGP keeps track of complete paths to a destination as an enumerated list of ASes to assist in routing loop detection.

BGP uses TCP as its transport protocol. It maintains routing tables, transmits routing updates, and makes routing decisions based on routing metrics. The primary function of a BGP system is to exchange network reachability information, including information on the list of AS paths, with other BGP systems. This information can be used to construct a graph of connectivity among the autonomous systems that can be used to prevent routing loops and to enforce AS-level policy decision.

[1]As reported in the presentation by Stephen Kent at http://www.net-tech.bbn.com/sbgp/Kent_CIP.ppt

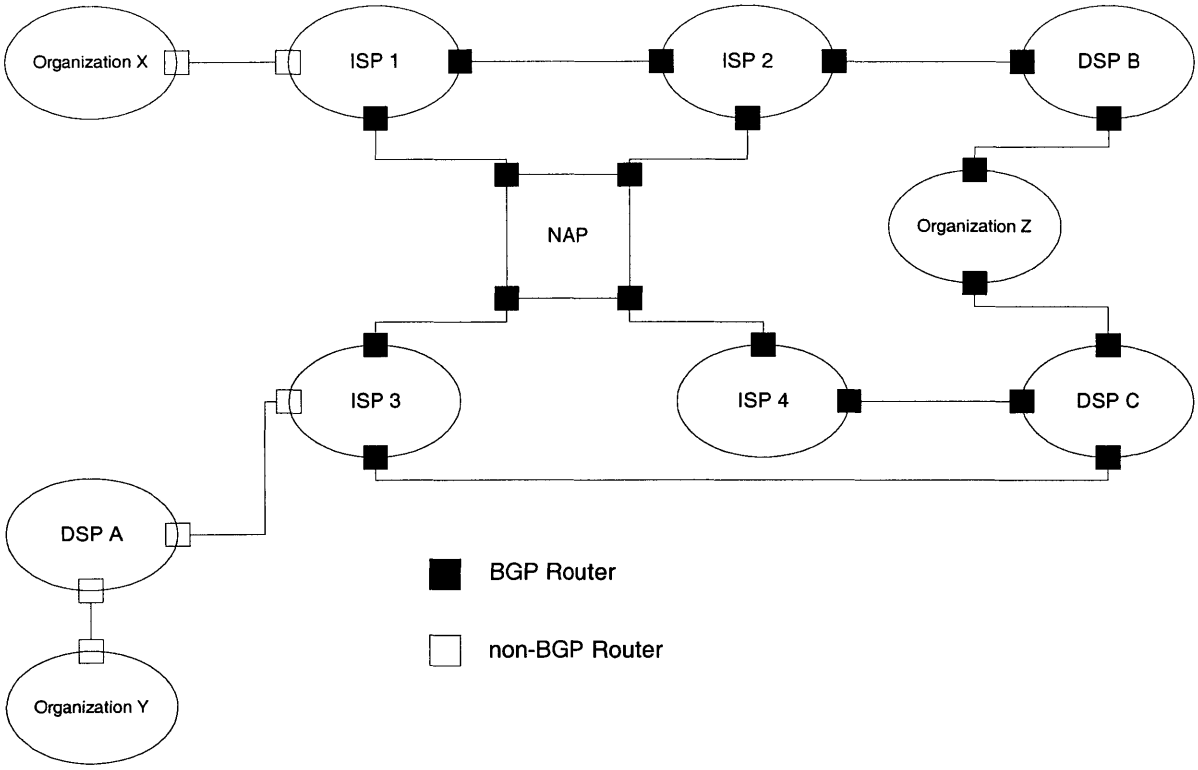

Figure 2-1: Simple BGP Model

2.2.1 How BGP Works

The administrator of an AS designates at least one node to be the spokesperson for the entire AS called the "BGP speaker". The BGP speakers are used to establish sessions with BGP speakers in other ASes and exchange routing information. Additionally, the AS also has one or more border "gateways" which are the routers through which packets enter and leave the AS. Usually a BGP speaker is the border gateway.

Each border gateway that implements BGP maintains a routing table that lists all feasible paths to a particular network. This routing table contains a list of next-hop routers, addresses they can reach, and the cost metric. However, the router does not refresh the routing table. Instead, routing information received from peer routers (other routers that it is directly connected with) is retained until an update is received. At the start of a connection, BGP peers exchange complete copies of their routing tables, which can be quite large. After these initial transfers, only changes are then exchanged in the form of UPDATE messages. These UPDATE messages

are used to advertise routes between a series of ASes. The destination is the network whose IP address is reported in the Network Layer Reachability Information (NLRI) field, and the path is the information reported in the path attribute field of the same UPDATE message. UPDATE messages are sent via TCP to ensure reliable delivery, and make long running BGP sessions more efficient than shorter ones.

If a BGP speaker chooses to advertise the route, it may add to or modify the path attributes of the route before advertising it to a peer. BGP provides mechanisms by which a BGP speaker can inform its peer that a previously advertised route is no longer available. There are three methods by which a given BGP speaker can indicate that a route has been withdrawn from service [5]:

1. The IP prefix that expresses destinations for a previously advertised route can be advertised in the WITHDRAWN ROUTES field in the UPDATE message, thus marking the associated route as being no longer available.

2. A replacement route with the same Network Layer Reachability Information can be advertised.

3. The BGP speaker-speaker connection can be closed, which implicitly removes from service all routes which the pair of speakers had advertised to each other.

2.3 The Vulnerabilites of BGP

Security for BGP may be defined by the correct operation of BGP speakers [5]. This definition is based on the observation that a successful attack against BGP should presumably yield incorrect operation of the protocol. Correct operation of BGP depends upon the integrity, authenticity, and timeliness of the routing information it distributes, as well as each BGP speaker's processing, storing, and distribution of this information in accordance with both the BGP specification and with the local, possibly confidential, routing policies of the BGP speaker's AS. The primary correct operation features of BGP, as reported by Kent et al [5], are:

- Each UPDATE received by a BGP speaker from a neighbor was sent by the specified neighbor, was not modified en-route, and contained routing information that is no less recent than the routing information previously received for the indicated prefixes from that neighbor.
- The UPDATE was intended for the neighbor that received it.
- The neighbor that sent the UPDATE was authorized to act on behalf of its AS to advertise the routing information contained within the UPDATE to the BGP speakers in the recipient AS.
- A neighbor that is withdrawing a route is the one authorized to advertise that route.
- The owner of an address space corresponding to a reachable prefix advertised in an UPDATE was authorized by its parent organization to own that address space.
- The first AS in the route was authorized by the owners of the address space corresponding to the set of reachable prefixes to advertise those prefixes.
- The neighbor that sent the UPDATE correctly applied the BGP rules and its AS's policy in propagating the UPDATE, in selecting the route, and in deriving forwarding information from it.
- The BGP speaker that received the UPDATE correctly applied the BGP rules and its AS's routing policies to determine whether to accept the UPDATE.

BGP has a number of vulnerabilities that can be exploited to cause its correct operation, as characterized above, to be violated.

Communication between BGP peers can be subject to active and passive wiretapping [5]. BGP uses TCP/IP for transport and this protocol can be attacked. A speaker's BGP-related software, configuration information, or routing databases may be modified or replaced illicitly by unauthorized access to the routers, or to servers from which router software is downloaded, or even via a spoofed distribution channel.

Most of these attacks can transform routers into hostile insiders. Effective security measures must address such Byzantine attacks [9].

Further exploitation of these vulnerabilities allows a variety of attacks [5]. For example, spoofing might occur by injecting fake BGP messages into a link. It is also possible for authentic BGP messages to be captured and either modified and re-injected into the link, combined incorrectly, or suppressed altogether. A compromised BGP speaker could generate UPDATEs for routes that do not legitimately pass through that speaker. Additionally, a compromised BGP speaker could generate UPDATE messages too frequently, or the selection of routes and distribution of UPDATEs could violate the local routing policies.

2.4 Motivations for the Secure Border Gateway Protocol (S-BGP)

As described previously, BGP is vulnerable to numerous attacks due to the lack of a scalable means of ensuring authenticity and legitimacy of BGP control traffic. BGP is a critical part of the Internet and errors can have adverse effects on message throughput. We may reduce some of these vulnerabilities by better physical and procedural security for network management facilities, BGP speakers, and communication links. We may also use encryption of the traffic at the inter-router (BGP speaker) links, or the end-to-end encryption of management information. However, some aspects of such security approaches are economically unattractive or infeasible e.g., they may require additional equipment. Moreover, accidental misconfiguration, which has proved to be a source of several significant Internet outages in the past, would not be prevented by such measures. Any security approach that leaves BGP vulnerable to such benign "attacks" leaves the protocol vulnerable. We thus try to implement countermeasures that would be strong against malicious attacks as well as accidents. A scalable system that ensures the integrity and authenticity for each BGP routing UPDATE, as well as the authorization of the initial speaker to advertise the

address space that is in the UPDATE is needed. Furthermore, as suggested by the end-to-end argument [14], lower layer mechanisms can make it easier or more efficient to do the upper layer end-to-end functionality, but cannot be an end-to-end alternative for the layer in question. Thus S-BGP was designed using end-to-end checks to minimize the adverse effects of compromise of any part of the BGP architecture, including the speakers and the BGP management system.

Chapter 3

Secure Border Gateway Protocol

Secure Border Gateway Protocol (S-BGP), an extension to BGP-4, was designed to address the security shortcomings of BGP. S-BGP adheres to the principle of least privilege and uses countermeasures to create an authentication and authorization system that removes most of the security problems associated with BGP. To facilitate adoption and deployment, S-BGP is designed to minimize the overhead (processing, bandwidth, storage) added by its countermeasures, and to be interoperable with the current BGP so as to be incrementally deployable. This chapter begins with a discussion of the security measures employed by S-BGP including the private S-BGP certificate extensions for Address Allocation, Autonomous System numbers and Router Identifiers. It also describes Route Attestations and Address Attestations, as well as the use of IPsec. Finally, it talks about the need for the efficient distribution of the S-BGP countermeasures and how the introduction of repositories helps to address this issue.

3.1 Security Mechanisms used by S-BGP

The S-BGP architecture employs three security mechanisms: a Public Key Infrastructure, a new BGP transitive path attribute containing *attestations*, which enable each BGP speaker to verify information about route advertisements that it receives, and the use of IP security (IPsec) protocol suite [5]. These security measures are used

by a BGP speaker to check the authenticity and integrity of the BGP updates that it receives and to verify the identity of the senders. While S-BGP cannot prevent an organization (ISP or DSP) from doing something detrimental to itself or its customers, it allows the ISP or DSP to detect detrimental actions from other entities.

3.1.1 Public Key Infrastructure

As described in Kent et al, *Public-Key Infrastructure for the Secure Border Gateway Protocol (S-BGP)* [6], a new Public Key Infrastructure (PKI) is used to support the authentication of ownership of IP address blocks, ownership of Autonomous System (AS) numbers, an AS's identity, and a BGP router's identity and authorization to represent an AS. It is based on X.509v3 certificates [2] that are used to achieve the authentication, and it mirrors the existing IP address and AS number assignment delegation system, thus avoiding many of the complications that may be encountered in the creation of a PKI.

Address Allocation

One private S-BGP certificate extension is for Address Allocation. The S-BGP PKI uses a certificate containing this extension, `SBGPIpAddrBlock`, to bind a public key to an organization and to a set of IP address families and prefixes. This certificate, in conjunction with address attestations, is used to verify that the IP address space owner has authorized one or more ASes to advertise the address space [5]. The details of this extension, as well as how the extension is used in the authorization process, are described in detail in Chapter 5.

The certificates containing this extension are arranged into a singly-rooted hierarchy [5], as shown in Figure 3-1, that parallels the existing IP address allocation system, and is issued through the same chain of entities that is responsible for address allocation in the existing environment. ICANN (or historically IANA) is the root of this tree. ICANN issues certificates for the address space ownership to the regional registries. Since the `SBGPIpAddrBlock` extension specifies the address space

being delegated, the validation of these certificates is constrained by the IP addresses in this extension. The proposed system does not require that address assignments be certified all the way to the subscriber, but only as far as the ISP/DSP from which its address is allocated, and similarly for DSPs that receive their address-space assignments from ISPs. The reason is that the subscriber's address is included in that of the ISP/DSP with which it is affiliated. Also note that a subscriber (or a DSP) who does not participate in BGP exchanges need not be issued a certificate if the subscriber's address space is derived from that of an encompassing ISP or DSP. Finally, only a single certificate[1] containing a list of address blocks is assigned if an organization owns multiple ranges of addresses so as to minimize the number of certificates needed to validate an UPDATE. Holders of these certificates can create an address attestation authorizing an AS to advertise the specified address space on their behalf.

Assignment of AS Numbers and Router Associations

Three types of certificates are used to support the authentication of ASes and BGP speakers, and the relationship between speakers and ASes in the S-BGP PKI [5]. A second S-BGP certificate extension is for the assignment of AS numbers and a third is used to bind a router to an AS. A certificate containing the second extension, `SBGPASNum`, binds a public key to an organization and a set of AS numbers. The second type binds an AS number and its public key (issued by an organization and signed using the private key corresponding to the public key in the certificate of the first type), and a certificate containing the third extension, `SBGPRouterId`, binds a public key to an AS number and to a BGP router's name, ID and public key. Together, these certificates allow BGP speakers to authenticate one another, and to verify that a given speaker is authorized to represent a specified AS. The details of these extensions, as well as how the extension is used in the authorization process are described in Chapter 5. Here too, the ICANN is the root of the hierarchy and the second tier consists of registries (see Figure 3-2). ICANN issues certificates for

[1]New certificates are issued to reflect the increased scope of ownership whenever organizations acquire additional address blocks.

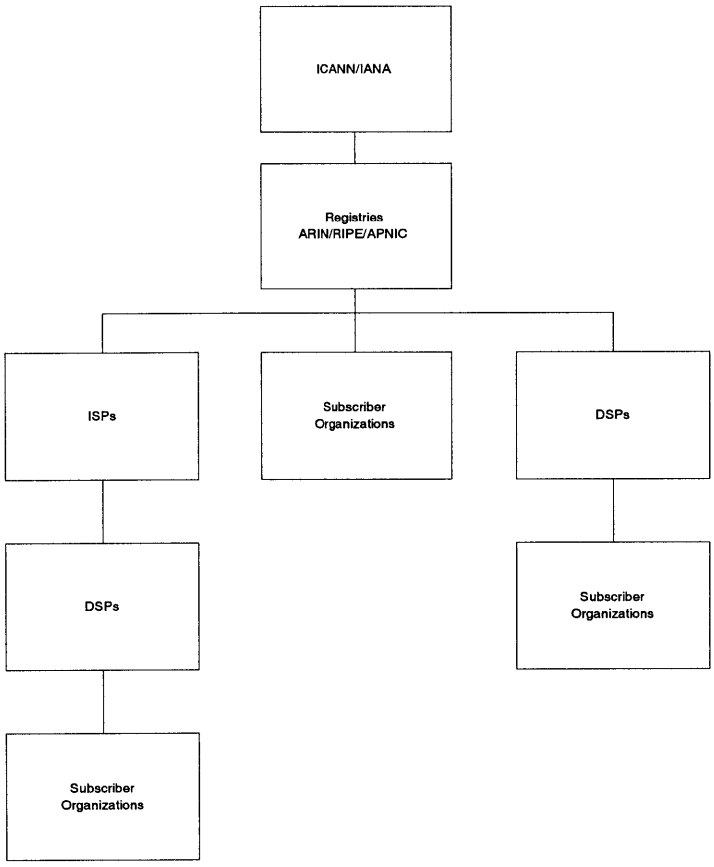

Figure 3-1: S-BGP Address Allocation

AS ownership to the registries, which then issue certificates to the ISPs, DSPs, and organizations that run BGP. AS operators issue certificates to routers as their AS representatives. Holders of router (or AS) certificates can generate route attestations (which are described in the next section), which authorize advertisement of a route by a specified AS.

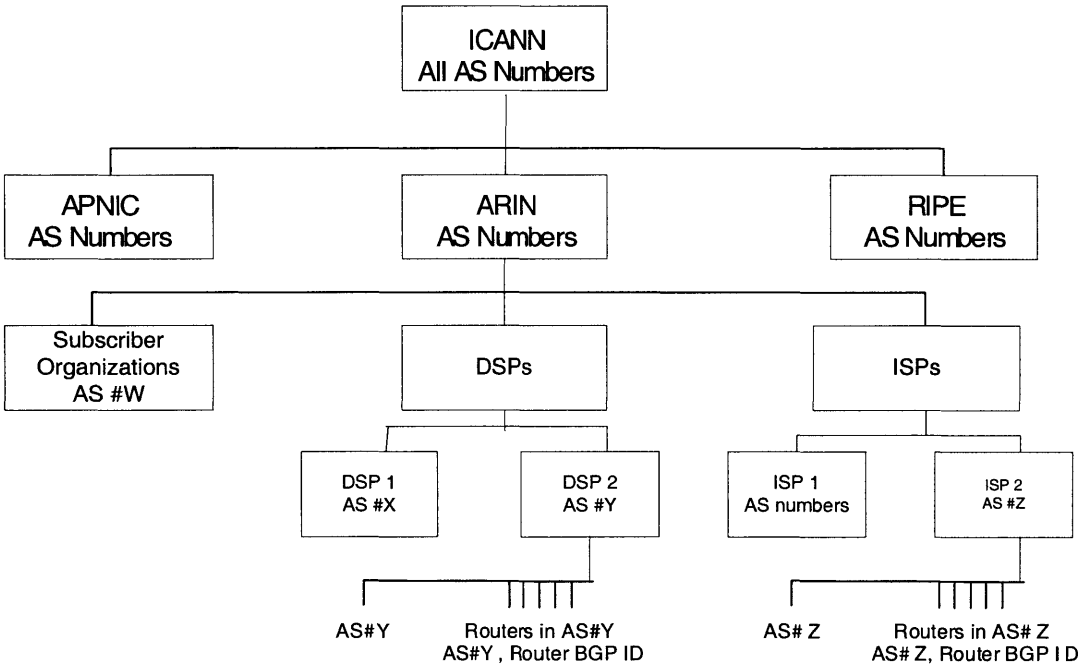

Figure 3-2: S-BGP AS Number Hierarchy

3.1.2 Use of Attestations

S-BGP introduced attestations in order to secure UPDATE messages. Figure 3-3 shows the structure of this new, secure UPDATE message as described in Kent et al [5]. The attestations are carried in a new, optional, transitive path attribute for route authorization that contains digital signatures, which protect the transitive routing information. The attestations are signed and verified using the keys and certificates. They enable each BGP speaker that receives a route advertisement to verify that each AS along the path has been authorized by the preceding AS along the path to advertise the route, and that the originating AS has been authorized by the owner of each IP address prefix contained in the UPDATE to advertise those

prefixes.

There are two types of attestations [5]:

- Route attestations (RAs) - where the issuer is a router authorized to represent the AS (or is the AS itself) and the subject is a transit AS. Route attestations are used to express a secure route as a sequence of AS hops. To sign the RA, the private key that corresponds to the public key of the certificate that binds the router to the AS is needed.

- Address attestations (AAs) - where the issuer is the organization that owns the address prefixes that it contains and the subject is one or more ASes that are authorized to advertise these prefixes e.g., the organization's ISPs. To sign the AA, we need the private key that corresponds to the public key of a network end-entity (EE) certificate that is issued by the certificate which assigns that address space to the organization.

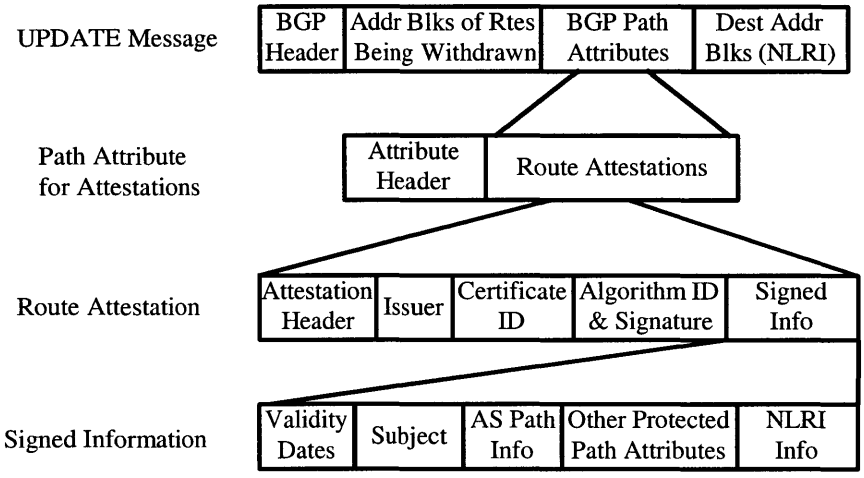

Figure 3-3: An UPDATE message with Attestations

3.1.3 Use of IPsec

IPsec is used to provide data and partial sequence integrity, and to enable neighbor BGP routers to authenticate each other (two way authentication) for exchanges of

BGP control traffic. It also prevents an active wiretapper from spoofing UPDATE messages or replaying valid ones.

BGP-4 has means for carrying authentication information, but no key management scheme or sequence numbering facility. IPsec supports automated key management and so is well suited for used with BGP. Furthermore, IPsec is now being put into routers by vendors and so it is becoming more readily available.

3.2 Distribution of Certificates, CRLs, and Address Attestations

The S-BGP architecture pays close attention to the performance and operational costs of introducing the proposed countermeasures. While previous work in the area of routing security has focused on the impact of generating and validating digital signatures, analysis suggests that bandwidth and storage requirements associated with signatures, certificates, and CRLs are much bigger issues [4]. This section describes how we propose to deal with the issue of the distribution of certificates, certificate revocation lists (CRLs), and Address Attestations (AAs) in S-BGP.

The security approach used by S-BGP relies on the distribution of countermeasure information [5]. Each S-BGP speaker must have access to the public keys required to validate UPDATEs. For non-leaf S-BGP speakers that receive updates from all reachable ASes, this can amount to the full set of certificates encompassing all address space owners, AS owners, and some of the S-BGP speakers.

While putting certificates in UPDATEs ensures that all BGP speakers get the data required to validate the attestations in an UPDATE, this process may be very redundant and make UPDATEs too big. Additionally, BGP updates are limited in length to 4096 bytes and so may not be big enough to carry all the certificates required to verify most UPDATEs. Alternatively, a new type of BGP message could be introduced for transmitting certificates and CRLs to address the packet size problem. This would still waste bandwidth, however, and would not be backward compatible

with BGP.

The proposed solution uses out-of-band distribution for certificates, CRLs, and AAs to all S-BGP speakers [5]. This database is relatively static and is thus a good candidate for caching and incremental update. Moreover, the certificates can be validated and processed against CRLs and only the necessary information such as the public key, subject, and selected extensions can be extracted and distributed to S-BGP speakers by ISPs/DSPs. This avoids the need for each speaker to perform the large number of signature verifications required for a certificate path validation, and thus saves both bandwidth and storage space. Although memory is inexpensive, most commercial routers that are currently deployed do not possess sufficient memory to store all of these certificates and attestations. Therefore, either additional memory or auxiliary systems are needed, even with preprocessing.

We make use of two tiers of repositories from which one can download the entire certificate, CRL and AA database [5]. The top tier is a system of several replicated, easy-to-access storage sites. These sites are housed on servers at Network Access Points (NAPs), which are interconnection points that tie the Internet Service Providers (ISPs) together. The reason for this is that if routing were to collapse, an ISP/DSP may still need to get to the repositories without using inter-AS routing. Therefore, the only points it would be able to reach in this situation are a neighboring AS or an attached NAP. The second tier of repositories is operated by the ISPs/DSPs, to provide local access for the S-BGP speakers within each AS. The second tier repositories request new certificates, CRLs, and AAs from the top tier. This allows an ISP/DSP to retrieve certificates before it actually needs them. However, the ISP/DSP-operated databases always push the extracted data to the routers in their ASes in order to minimize the propagation delay of this information. These repositories will allow us to avoid the dependency loop that would occur if the solution required inter-domain routing in order to access this database. Transfers of all the objects of each type in the database from the first to second tier repositories can also be carried out. Furthermore, in the first tier, the certificates, CRLs and AAs are signed so there is no need for additional integrity mechanisms of the individual

objects in the transfer. In order to minimize processing and storage overhead, the NOCs should validate certificates and AAs, and send processed extracts to routers. However, because this transfer involves extracted objects, an explicit signature is required. The ability to verify the signature for this transfer is a local one, i.e., the routers need know only a public key (or certificate) for the NOC.

This thesis seeks to design and implement the top tier of these repositories, which can be used to solve the distribution issues.

Chapter 4

Design of the Repository

Chapters 2 and 3 describe the background and the work done by BBN Technologies in designing the Secure Border Gateway Protocol. As mentioned in Chapter 3, the Secure Border Gateway Protocol (S-BGP) requires the efficient distribution of certificates, CRLs and address attestations (AAs). The rest of this thesis presents a design for the repository, and seeks to address the security issues that are relevant to its deployment.

The repository is used to store the certificates, CRLs and AAs that are used in S-BGP. We have multiple repositories at different sites, which are synchronized on a regular basis. This chapter begins with a description of the requirements that were used in designing the repository and then presents the repository structure, which includes the interfaces and the data storage. It then discusses how data is transferred to and from the repository. The clients, Network Operations Centers (NOCs), upload changes to the repository by providing new objects to add and copies of existing objects to delete. The NOCs also download the whole database, process the data, and upload the extracted pieces to their routers.

4.1 Overview

Figure 4-1 shows the structure of the repository. In this system, there are three interfaces: an Administrative Interface through which the system administrators access

the data, a Client Interface and a Server Interface through which the daemons run on the NOCs and the other S-BGP servers, respectively, can access the data. These interfaces allow for easy interaction between the system and anyone uploading data or retrieving the data stored. We assume that a NOC connects to the same repository on every occasion, i.e., the repository used by a NOC is configured into its client host with fallbacks, similar to the Domain Name System (DNS) configuration scheme. If a NOC attempting to connect to Repository A discovers that it is unavailable, it can then learn from a fallback table which it stores that the next preferred server for connection is some other Repository B. The NOC sends the connection message to Repository B and continue performing transactions as necessary at this site.

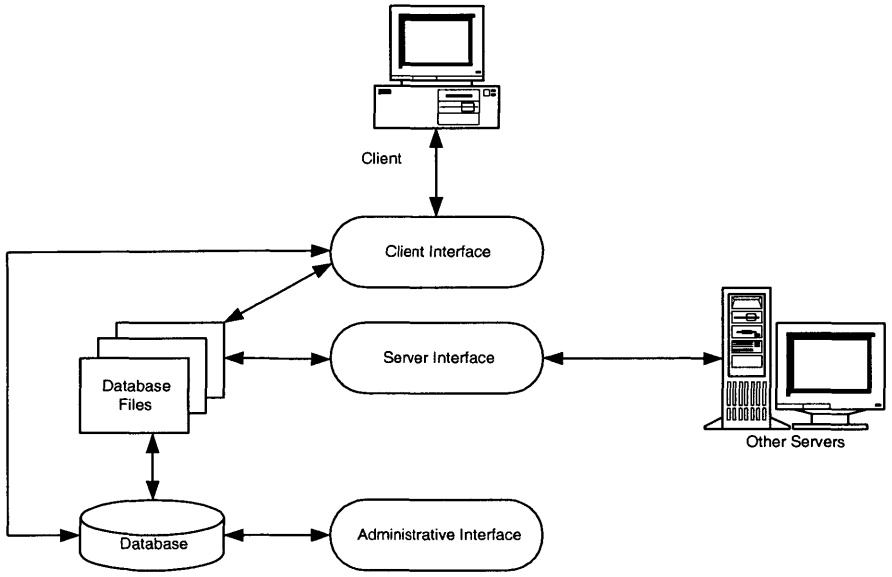

Figure 4-1: Repository Design

All NOC interaction with the repository uses the Client Interface, which supports a small number of fundamental operations. Thus there is a clearly defined, open interface for the repository that allows others to write client-side applications and higher level interfaces. A server's Incremental Changes Table simply contains the names of all the new files uploaded to the repository that other servers need to obtain. Servers contact each other every day to get the new objects that have been added to the repository.

The repository enforces access control based on what data is involved and who

accessed or submitted it. Each object is verified[1] by making sure that it is signed by the purported and authenticated organization. The repository can store certificates, CRLs and AAs, and Certification Paths.

4.2 Requirements

This section discusses some of the specific functions that need to be addressed by the repository.

4.2.1 Database Requirements

The database must support the storage of the following types of data objects in separate database tables:

- X.509 Certificates
- Certification Revocation Lists (CRLs)
- Address Attestations (AAs)
- Certification Paths

The first three types of objects are required by the S-BGP system. The fourth table contains certification path objects, which are used in the access control mechanism for the first three types. Instead of storing the actual paths, we could store certificates only and recreate the certification path whenever it is needed for verification. However, we choose to store the whole certification path to prevent other NOCs from not being able to validate the objects that they download because of a premature deletion. NOCs can look up the whole path in certification path table, even if one of the certificates constituting this path had been deleted from the database before other objects that need the path have been replaced by ones using a new path.

The repository must also support Additions and Deletions, which allow NOCs to modify the repository as new certificates, CRLs and AAs are generated, and old

[1]The actual signature is verified by the NOCs.

ones expire. While a search function is not required, it would help to increase the usability of the repository by the administrators. For example, an administrator might use the search function to determine if an object were in the database without downloading the whole database table. Search functions are not supported for the NOCs as explained in Section 4.4.1.

The repository is not going to handle life cycle requirements such as CRL replacements or certificate expirations. The NOCs have the responsibility of deleting any of their objects that expire and adding new objects.

4.2.2 Upload Requirements

The upload files (batch files) from the NOC have security requirements, as well as requirements for the correct processing of individual transactions included in the files. The repository must be able to retrieve the batch file signer's certificate, which contains the public key corresponding to the private signing key, and verify the signature on the file. It also has to be able to store the batch files which were uploaded until they can be processed. The uploaded files must be saved in a directory until the other servers have copied them during synchronization as described in Section 4.4.4.

The repository must be able to extract the individual transactions from the batch file and perform all these transactions. When a transaction fails, it must return an error code depending on the type of failure, which can then be incorporated into a reply message. These errors must be noted in the Log File as discussed in Section 5.6. If a transaction is error-free, the reply message must also include a "Successful" code to indicate this. A summary of transactions performed must be written to the Audit log, also described in Section 5.6. The repository also needs a mechanism for the reply message to get back to the NOC that uploaded the file about whether the addition or deletion of the particular items were successful. Each transaction may have a dependency transaction list. This list contains the IDs of all the transactions upon which the current transaction is dependent, i.e., the current transaction must not be performed unless all the transactions on which it is dependent have succeeded. Therefore, we must be able to check the status of all dependent transactions, and

ensure that they were successful before doing the current transaction. Furthermore, it must ensure that all previous uploads from a given NOC have been processed before performing the transactions in a later upload, due to dependency restrictions.

4.2.3 Download Requirements

The repository must support full database downloads. The webserver must also be able to identify which object files have been requested for download: certificates, CRLs, AAs, or certification paths. The Download Processor must be able to generate download files daily, each of which contains all the objects of a particular type in the database. The download files are saved on disk and transferred to the authenticated NOCs when they are requested.

4.2.4 Data Replication

The repository must also support the replication of data among multiple physical sites for load balancing potential and also for robustness in case of failure for any reason such as Denial-of-Service attacks, Operating Systems failures, power failures etc. It therefore must have a synchronization process for the servers, so that each server can be updated with transactions that were submitted to other sites.

4.2.5 Capacity

The following figures apply to the S-BGP database based on estimates of the Internet environment given in *Secure Border Gateway Protocol (S-BGP) - Real World Performance and Deployment Issues* by Kent, Lynn, Mikkelson and Seo [4]. This paper states that in February 1999 it was estimated that there would be about 58,600 certificates in total: 4 for Internet registries, 44,000 for organizations that owned address prefixes, 1,800 for organizations that had been assigned AS numbers, 5,300 for autonomous systems, and 7,500 for BGP speakers. We expect 1 AA per organization that owns an address prefix, and 1 CRL per autonomous system. Since we anticipate

that this number will grow each year, the database must therefore be able to handle this number of objects in each of its tables:

- 100,000+ certificates
- 100,000+ AAs
- 10,000+ CRLs

CRLs are expected to be replaced daily, whereas certificates and AAs are likely to be valid for one year. We therefore expect all of them to be replaced every year, and we assume that these replacements are spread out over the whole year. Therefore we have, number of adds and deletes to be approximately:
200,000 objects * 2 (add and delete) / (5 work days/week * 50 work weeks a year) = 1,600 objects/day. The paper also estimated that there would be 175 new certificates a year. So in total, we estimate that less than 15,000 objects will be uploaded every day. However, since each NOC will upload its own objects, and we have over 5,000 NOCs, each upload file will contain only about 3 transactions per day.

4.2.6 Security Requirements

The repository needs access control mechanisms to restrict who can access what data. The access control encompasses authentication and authorization. Authentication mechanisms are needed for NOCs uploading and downloading data, and for servers during synchronization, as well as for remote access by an administrator, whereas authorization checks are needed for individual objects in the upload files. The measures taken to obtain these security requirements is described in detail in Chapter 5.

As described in previous chapters, the repository provides storage for 4 types of objects: certificates, CRLs, address attestations (AAs) and certification paths. The repository supports both downloads and uploads of data items. Furthermore, all of the objects transferred to and from the repository are individually, intrinsically digitally signed, i.e., the signature is a part of the actual object. Upload files, ICT files and Reply files, are also signed as blocks. The motivation for the storage of the certification paths in the database is discussed in Section 4.4.2.

Download transactions are authorized for all authenticated members of the S-BGP community. This means that any user identified as a NOC operator or a repository administrator is authorized to perform a download. The identification is based on the possession of an SSL client certificate issued under the S-BGP PKI. This requirement is imposed primarily to prevent Denial-of-Service attacks against repositories through unauthorized download requests.

Upload requests must also be authenticated based on proof-of-possession of a SSL client certificate issued under the S-BGP PKI. In addition, it is necessary for transactions in upload files to be authorized correctly to prevent destruction of data in the repository by overwriting. Only users identified as NOC operators with one of these SSL certificates are allowed to initiate uploads. When an upload file is received from an authenticated user, the repository must verify that the set of data items to be uploaded is within the responsibility of the user, i.e., that the user is authorized to "manage" this data. These items include the certificates, CRLs, and address attestations that are associated with the portion of the address and AS number space that the user represents. This scope of authorization is inferred from the user's certificate, specifically the S-BGP extensions.

4.2.7 Efficiency

Time constraints are important for S-BGP so that data gets distributed within the system without significant delay. Repository interactions performed by the web server should therefore be optimized. It should only take a few minutes for a NOC to upload files or download the whole database. Synchronization of the servers should occur within reasonable time intervals (approximately 24 hours) in order to propagate changes within the system in an acceptable time period. This time period is considered reasonable as we expect NOCs to provide uploads on a day-to-day basis.

4.3 Repository Structure

The high level repository design is very simple. The components of the repository are: the database and database files (the download files) for data storage, the Client Interface through which the NOC interacts with the repository, and the Server Interface through which synchronization occurs. We now describe each of these components, as well as the Administrative Interface. Figure 4-2 shows how the Administrative Interface and the components of the Client Interface, the Apache Server, the Upload Processor and the Download Processor, fit into the repository structure. The Server Interface is shown in Figure 4-4 in Section 4.4.4 on Synchronization.

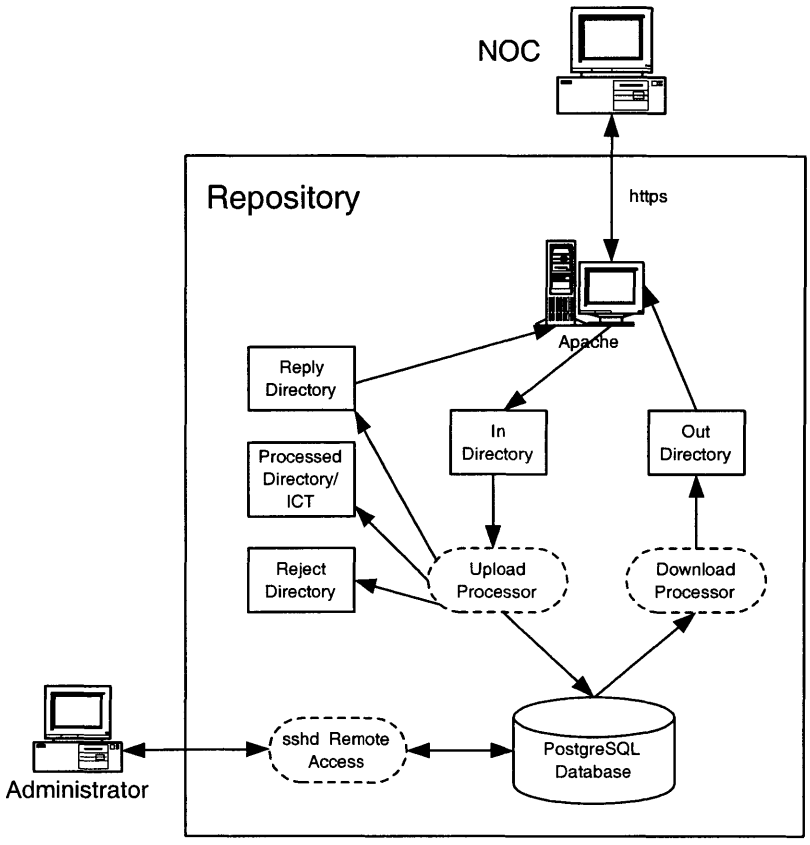

Figure 4-2: Structure of the Repository

4.3.1 Data Storage

Information in the repository is held in the persistent store (database). The implementation of the database is completely hidden from the outside and is accessed using the Client and Administrative Interfaces only.

The data in the repository is organized according to the object type. The database thus contains four database tables. There is a certificate table, a CRL table, an AA table, and, finally, a certification path table, which stores the certification paths from the batch files uploaded by the NOCs. These paths will be used for validating the certificates, CRLs and AAs when they are downloaded by the NOCs. Within each table, we store the fields needed to identify a particular object, as well as perform all the functions that our repository supports (mainly authentication, add, delete, and access control). These fields are outlined in Tables 4.1-4.4. The actual objects are also stored in the database tables. They are encoded because the database does not support the inclusion of the null bytes that appear in a DER-encoded object[2]. The first three tables also contain a reference to the sequence number of the certification path so that its references to the `count` can be easily maintained when these objects are deleted from the database. This is described further in Section 4.4.2.

For each table there are also fields that are used to give unique identifiers for the objects. For the certificates, we use the hash of the issuer name and the serial number since certificates issued by an entity have unique serial numbers. The hash of the issuer name is enough for the CRL database since we only store the most recent CRL per issuer distinguished name, while for the last two databases we use the hash of the objects (`aaHash` and `cpathHash`) as these have a very high probability of being unique (since the chance of collision with SHA-1 hash is very small[3]). In the rare occurence that a collision does occur, both objects are written to the database with the same key. This means that if either of the objects is to be deleted, the database would return both records instead of just the one intended for the operation. The database then does a byte-by-byte comparison of the object to be deleted and the

[2]Distinguished Encoding Rules (DER) is a type of encoding used for ASN.1 objects.

[3]The probability of collision for a 160-bit hash with 1 million objects is $\binom{1000000}{2}/2^{160} < 2^{-120}$.

Field Name	Description
seq	Index Number (generated when the object is added)
sn	Serial Number
iHash	Hash of Issuer DN (distinguished name)
iDns	Issuer DN - DNS form
cert text	Actual Certificate (encoded)
cpath_seq	Index Number of the Associated Certification Path

Table 4.1: Certificate Table

Field Name	Description
seq	Index Number (generated when the object is added)
iHash	Hash of Issuer DN (distinguished name)
iDns	Issuer DN - DNS form
crl text	Actual CRL (encoded)
cpath_seq	Index Number of the Associated Certification Path

Table 4.2: CRL Table

Field Name	Description
seq	Index Number (generated when the object is added)
aaHash	Hash of the address attestation (AA) object
aa text	Actual AA (encoded)
cpath_seq	Index Number of the Associated Certification Path

Table 4.3: Address Attestation Table

Field Name	Description
seq	Index Number (generated when the object is added)
cpathHash	Hash of the certification path object
count	Number of references to that certification path
cpath text	Actual certification path (encoded)

Table 4.4: Certification Path Table

objects in the records returned, and then only deletes the record that was requested for deletion.

4.3.2 Client Interface

The Client Interface is the part of the repository that interfaces with the outside world i.e., where the NOCs request services from a repository. The Client Interface implements all accesses to the data stored, converts between the internal and external forms of objects, and manages rights and permissions. A web server is at the front end of the Client Interface. The web software provides a user-friendly interface to a NOC for uploading and downloading objects. It also performs authentication of all the users connecting to the repository. Authentication is based on proof-of-possession of an SSL certificate issued to the NOC operator under the S-BGP hierarchy, as described in the next chapter. The Client Interface also includes the Upload Processor and the Download Processor, which are the functions used to process the upload and download files. The Upload Processor performs the individual transaction in the batch file, and does authorization checks for the objects presented by the NOC. The Download Processor generates the database files for the NOCs to retrieve. The NOC interacts directly with the web interface only, but the web interface accesses the same directory structure as the Processors.

4.3.3 Server Interface

This interface is used for server-to-server communication. The methods it supports are for the sole purpose of synchronization. The files received during synchronization are processed by the Synchronization Processor, which is described in Section 4.4.4. The Server Interface also performs server-to-server authentication during this process. The process is automated and is scheduled to run at certain times. We are using a "pull" mechanism in which the servers ask for the updates on a regular basis. If it becomes necessary, "push" methods can be implemented later on to propagate changes. The pull model allows the server to request the Incremental Change Tables

(ICTs) and batch files on a scheduled basis and so it can keep track of which others it receives information, and decreases the load on the overall system since there would be less server-to-server connections made. Additionally, it is used to prevent unexpected interruptions or errors. For example, Server A may "push" data to Server B while Server B is running its Synchronization Processor. Server B may get the ICT and begin processing it before the batch files have been received. In this case, Server B gets a number of "File not found" errors, and so does not get synchronized properly. Otherwise, servers can be configured not to receive "push" data while their Synchronization Processor was running. If this were the case, Server A would have to keep trying to send the files to Server B until Server B was ready to receive them.

4.3.4 Administrative Interface

The administrator has an account on the system that is used to log into a particular repository remotely and to manage any problems that might come up. He or she must be able to perform all the standard functions, i.e., additions, deletions and searches, in addition to being able to view the audit file and the log file.

The Administrative Interface connects the administrator to the database and thus any changes made are written directly to the database tables. It is therefore very important that only authorized people are allowed to use this interface. The administrator logs into the server and provides an SSL certificate. Authentication is therefore based on proof-of-possession of the SSL certificate. These mechanisms allow us to track who accessed the data at which points in time. Each administrator also needs an Operating System account, and the use of a password could be optional, depending on the requirements for a particular repository. For example, if the administrator always connects via SSL, he or she does not need a password, whereas if he or she logs in to the server from the console, a password may be required. All transactions performed by the administrator also have to be logged. Furthermore, the Upload or Download Processors must not be run while an administrator is doing anything on that server, with the exception of viewing the audit log. The reason for this is that the administrator may be changing the data in the database, which can impact

both uploads and downloads. However, access to the repository by the NOCs is not affected since the web interface is used to access the directory structure that holds the download files for retrieval by the NOCs and the upload files until they are processed, but not the actual database.

4.4 Data Transfer

Figure 4-3 shows the flow of data between the S-BGP server and the NOC, and among the S-BGP servers. The distribution of data, i.e., the certificates, CRLs and AAs, occurs in the following ways:

1. Uploads of the data items to the S-BGP server after they are generated by the NOCS.

2. Downloads of the entire database from the S-BGP server.

3. Synchronization among S-BGP servers in which uploads made to one server are copied by the other servers, which minimizes the effects of outages on the repository. For example, if one repository site cannot be accessed, NOCs can simply be routed to another server using a fallback table which it stores. This fallback table is an ordered list of the repository servers to which it can connect, so if one is not available, it can go down the list to find an alternative.

4.4.1 Single Transactions

This section gives a brief description of the additions, deletions and searches transactions that can be executed by the repository. However, for upload files, only additions and deletions are currently supported. For a replace operation, the NOC needs to perform a deletion of the old object and then addition of the new one. Before doing a transaction from the batch file, we need to verify that the NOC that uploaded the file is authorized to perform this transaction. The algorithm used for this authorization process is described in Section 5.5. The signatures of the individual objects that

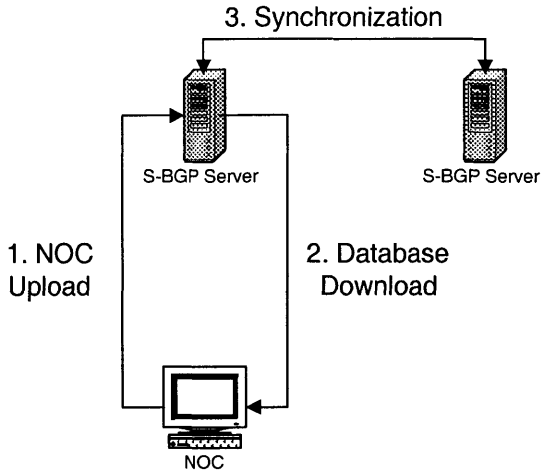

Figure 4-3: S-BGP Server Data Transfers

are added and deleted are not checked since the batch file is signed and the Upload Processor can verify the integrity of its contents by validating this signature (by the end-to-end argument [14]). Whenever a transaction generates an error, it is written to the Log File. All transactions performed and their results are noted in the Audit File. These are described in Section 5.6.

Additions

Additions are performed by providing the new object to be entered. First, the object being added is checked to ensure that it is issued by the authenticated organization, and the authorization algorithm is run. If these checks are successful, any additional information that is needed is calculated (such as `iHash` for certificates and CRLs). A query containing all the information to be put in the database is then built. The "Add" first checks that the object is not in the database. If a match is found, the object is already in the database. If this happens, we check that the record in the database has the same certification path as the object we are presenting. In this case, we do nothing. However, if the certification paths are different, we delete the old record in the database and add a record with the new certification path reference. If the object is not found, the appropriate record (containing the new object) is added to the database. The reference to the certification path `count` is updated as described

in Section 4.4.2. A reply is generated based on the results of the transaction. These reply codes are given in Appendix A.2.

Deletions

Deletions are performed by providing a copy of the object to be deleted. The first step is to verify that the purported operator is authorized to delete the particular object. Next, any additional information is calculated as needed and the database query is built. The "Delete" checks if there is any such object in the database. If there is a match, the object is then deleted. The reference to the certification path `count` is updated accordingly as described in Section 4.4.2. If there is no match, there is no object to delete. A reply code is then generated as for the Addition.

Searches

Searches are performed by providing a copy of the object. As currently implemented, searches can only be performed by an administrator and they confirm whether or not an object is in the database. Since all administrators are allowed to search for objects in the database without any restrictions, there is no need to perform any verifications on the object submitted to the search. Currently, the additional information is calculated as for the Addition and the database query is built. The "Search" checks if there is any such object in the database. If there is a match, then an "object found" message is returned. Otherwise a "not found" message is returned.

Searches based on certain criteria will be supported by the repository in the future, as mentioned in Section 7.1.1. This functionality will be supported only for administrators. Since it is very unlikely that administrators will have copies of all the objects in the database, if there is any problem, they would need to search by some other criteria, most probably the issuer name and serial number. However, the repository does not allow the NOCs to perform searches as this adds too much overhead to the servers. If the NOC does want to search for a specific item, it can search in its local database or the database files that it downloaded. Although we expect that all objects should be available in the downloaded files a long time before

they are needed by the routers, the repository does not need to invest resources to handle this case as the NOCs may simply change the router configuration to accept the "missing" object, and use the next day's download files to fix the problem.

4.4.2 Uploads to the Repository

NOCs could make uploads to the repository in two ways: update the repository with additions and deletions as single transactions, or use an upload file (batch file) which would contain all the transactions that a NOC wants to upload. In our design, we use the batch file format for uploads as this is more efficient than using single transactions.

Single transactions could be sent from the NOC to the S-BGP server in a synchronous fashion. This allows for the result to be made available immediately instead of in a reply file. Thus when a transaction fails, the operator is notified and so can fix the transaction so that others that list this transaction as a dependency do not fail as well. However, doing things this way would lead to longer sessions with the server. Also, having the operator check these results would likely mean that uploads would have to be done during the work day when response times may be longer (net congestion). This also includes the human factor i.e., users could become frustrated easily by long delays, which may lead to requirements for faster server hardware. Changes to the server are therefore made in a batch file format. Note that if a NOC only has one change to upload, it still produces a batch file containing that single transaction.

Upload Files (Batch Files)

Table 4.5 gives the structure of the Upload file and a brief description of each of the fields it contains. The `serialNumber` is used to uniquely identify each batch file from a particular `submitter`. The `certificationPath` from the upload file is extracted and used as described in the Section on Certification Paths on page 57.

The batch file also contains a list of all the transactions that a NOC wants to upload to the repository. The file is signed, and the signature must be verified. Each transaction in the upload file contains an add or delete indicator which directly re-

Batch File Field Name	Description
`serialNumber`	The Serial Number of the Upload File
`submitter`	The Issuer Distinguished Name of the NOC operator that submitted the file.
`keyIndentifier`	Key Identifier of the submitter.
`email`	This is an optional field for the email address of the NOC operator.
`certificationPath`	The certification path for the upload file.
`transaction-list`	List of transactions to be processed.

Table 4.5: Structure of the Upload Files

lates to the type of transaction to be performed, an object type (certificate, CRL, or AA) which identifies the database table relevant to the transaction, and the objects to be added or deleted. The transaction also has a list of dependency transactions that are only valid for the duration of the interval during which the batch file is being processed. Before doing a transaction, we first ensure that the NOC is authorized to add or delete the particular object from the database, and, if so, we check our dependency transaction list, which contains all the transaction IDs in the current batch file upon which this transaction is dependent, to make sure that all the transactions listed there were successful. To do this check, the NOC operator must order the transactions in the Upload file such that transaction IDs that occur in the dependency list of a given transaction represent entries that precede it in the batch file. If this ordering is correct, the transaction can proceed. Otherwise, the reply code for a dependency failure is generated and the next transaction is processed. Similarly, if there is any other error in a transaction, the appropriate reply code must be returned before the rest of the batch file is processed. The ASN.1 specification of the Upload file (`S-BGP-Repository-Transactions`) is included in Appendix A.1.

If the upload file is not in the correct format or the signature on the file is invalid, then we have an intolerable error (as defined in Section 6.6) and the file is moved to a reject directory. This way it is not propagated throughout the system. However if there are no fatal errors but one of the transactions within the batch file fails while being processed, the file is still considered "good" and it is moved to a directory

where it is made available to other servers during synchronization. The file name is also written to the Incremental Changes Table as described later in this Section.

Reply Files

The reply file is constructed by the Upload Processor after it processes a batch file. The file contains the same serial number and submitter as the original batch file, the name and key identifier of the repository, and the certification path for the repository. It also includes a list of replies. Each entry in the list contains a transaction ID, the type of operation performed, the object type, and the corresponding reply code that was generated on performing that single transaction. The reply file is then signed and placed in a special directory from where it can be retrieved by the NOC. The NOC is able to identify the reply due to the naming of the files. The reply files have the same name as the upload file, with the suffix `.reply` added. The ASN.1 specification for the reply file (`S-BGP-Repository-Replies`) is given in Appendix A.1.

Incremental Changes Table

At each repository, an instance of the Incremental Changes Table, ICT, keeps track of the upload files that have been processed at that particular location. It is used for synchronization and is modified every time the server processes a batch file. Each item in the ICT is the full name of a batch file. Whenever a server processes a batch file, if there are no intolerable errors then the server writes the full pathname of the batch file to the ICT. If the synchronization period on the system is set to x hours, i.e., the servers retrieve data from other servers once every x hours, then a new ICT should also be started every x hours. The old ICT is then signed by the server using the private key corresponding to the public key in its Repository end-entity (EE) certificate, and placed in a special synchronization directory from which other servers can obtain it. We can configure the servers to keep more than one ICT (and not just the latest one) in this directory. This is useful if a server is down at the time it would have normally synchronized with another server, since it can now retrieve not just one ICT but all the ICTs that it missed. The old ICTs can then be deleted after

a few synchronization periods have passed since it is assumed that they would have been retrieved and processed by then. Note that the synchronization period will be standardized for the servers.

Certification Paths

The certification paths stored in the database are constructed by the NOC and are included in the Upload files. As mentioned earlier, the retention of certification paths is to prevent problems if a NOC deletes a certificate that is needed to validate another object that is still in the database. Certification paths are not explicitly added to the database, i.e., they are never added by a NOC. Instead, they are added to the database as part of the Upload Processor functions. If the path already exists in the database, then its sequence number is found, while, if the certification path does not exist in the database, it is added and its sequence number is returned. For every "Add" transaction in the batch file, the `count` related to that `seq` number is incremented, and for every "Delete" transaction, the `count` is decremented. In this way we can keep track of when a certification path is still being used (its `count` is greater than zero). When the `count` becomes zero, the certification path is no longer needed to verify the signature on any object in the database and so it is deleted. Furthermore, objects added at the Administrative Interface have no certification path associated with them so the `cpath_seq` for those records are `NULL`. The certification path `count` is decremented when objects that reference that path are deleted.

4.4.3 Database Downloads

A NOC could obtain all changes made in a given time period by obtaining all the relevant changes that were made at a repository since it last synchronized its database or by downloading the whole database (certificates/CRLs/AAs). In the repository, we support full database downloads.

Allowing the server to obtain relevant changes can be performed in two ways: allowing the NOC to query for changes made to the repository since its last set of

downloads, or by generating incremental download files on a daily basis. Files that are generated on a daily basis containing changes to the database (incremental download files) may be used in the future as described in Section 7.1.1. While getting the changes may be more efficient than processing the whole database, it may cause a strain on the server as each NOC would have to query the server for the transactions since its last connection, and the server would then need to generate the appropriate file. This also opens the door for potential errors with time inconsistencies, which could lead to the NOCs not getting all the objects. To prevent this, we therefore use full database downloads. At a specified time every day, the server generates four files, one for each type of object stored in the database (certificates, CRLs, AAs and certification paths). The files contain the time, the name of the repository, the object-type, and all the objects in the database of that type concatenated together. The NOCs then download these files from the repository. These files are not signed since the objects they contain are individually signed and each signature is verified by the NOC after the download. Also, the download takes place over an SSL connection which ensures data origin authenticity and connection-oriented integrity. Note that this is due to our requirement that database downloads should only take a short time. The ASN.1 specification for the download file (`S-BGP-Repository-Downloads`) is given in Appendix A.

Methods of Secure Download File Transfer

We use HTTPS as it is the most popular and widely available of the three options investigated, gives comparable performance to the other options, and can be used with a simple web interface. HTTPS also uses SSL, the same protocol employed for uploads, and that the use of this protocol is consistent with the PKI that S-BGP employs. The NOC therefore connects securely to this web interface and proceed to download the database files.

In choosing this method, we considered HTTPS and two alternatives for transferring the database download files from the server to the NOC.

1. **A "secured" FTP**: FTP clients do not encrypt a connection automatically

before asking for a password. Therefore we would need a special FTP client which was modified to authenticate and protect the FTP session. This could be done using IPsec (in which case the certificates for authentication would have to be passed in the IKE exchange[4] that sets up the security association), or Kerberos (as in Kerberized FTP).

2. **SCP**: scp is a secure alternative to the regular UNIX rcp command for copying files between hosts. scp uses SSH for authentication and data encryption. It can be used it between hosts with SSH installed.

3. **HTTPS**: The last alternative was one in which the NOC does an HTTPS `GET` request for the download file.

In each option, the NOC sends an HTTP response immediately, indicating whether or not the operation succeeded.

4.4.4 Synchronization

The repository uses multi-master replication i.e., any server may accept modifications for a given entry. There is no need for conflict resolution because each object is not being identified by a sequence number but by some other handle that is unique globally. As noted in Section 4.3.3, we adopt a "pull" model in which the servers query for the changes instead of forwarding any changes made to the other servers ("push" model).

Synchronization between servers must be done fairly often to ensure that routers can get changes made to the repository by the upload files before the routes that require this new information are propagated, and also to minimize inconsistencies. All batch file updates within a certain time period on a given server can be received by another server simply by retrieving the Incremental Changes Table of that server and the batch files that it specifies. The ICT, as mentioned earlier, is signed using the private key of the server that generated it. Other servers can therefore validate

[4]The Internet Key Exchange protocol given in RFC 2409. http://www.ietf.org/rfc/rfc2409.txt

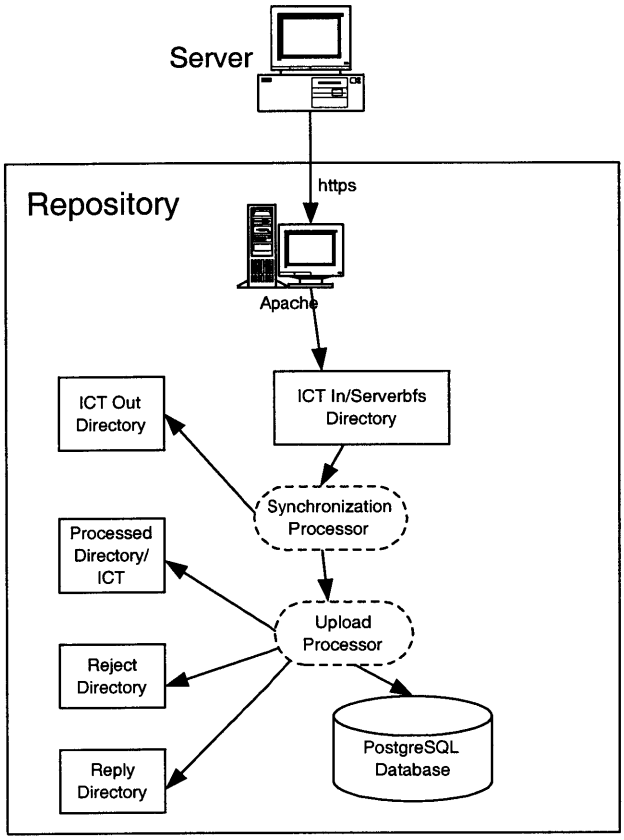

Figure 4-4: Synchronization

the signature on the ICT to verify the integrity of its contents. Server-to-server authentication must be performed before any information is transferred.

The Synchronization Processor and the directories used during this process are shown in Figure 4-4. For synchronization, the other servers get the batch files from the "Serverbfs" directory, and can validate their signature without actually checking the signatures on all the digital objects that they contain. When all the files (ICT and batch files) are received by the server, the Synchronization Processor processes all the batch files in the ICT in the order in which they are listed, if they have not previously be processed. It checks whether or not each batch file listed in the ICT has already been processed by comparing the names of the files to minimize the time wasted on going through transactions that were already done.

Consider Figure 4-5, which gives an example of synchronization among 3 servers, A, B, and C:

1. Server B gets Server A's ICT. It processes all the files contained in that ICT and adds those new batch files to its current ICT. Server B's ICT now contains all the uploads at both Server A and Server B up to that specific time.

2. Server C gets Server B's ICT and makes all the changes in that ICT. Its current ICT now contains all the uploads at both Server B and Server C, as well as the uploads at Server A up to the time that Server B got Server A's ICT.

3. Server C then gets Server A's ICT. Server C will already have processed some of the uploads in this ICT since it received them from B's ICT. However there may have been objects added since Server B got A's ICT. Server C can now get these batch files immediately instead of waiting for the next time it synchronizes with Server B at which point B should have received A's new batch files.

4. Server A gets Server B's ICT at a later point in time. When processing this ICT, it checks if a batch file has already been processed as described in Section 6.8 and only performs the transactions in the files that it had not received previously.

5. Server A gets Server C's ICT. Again, Server A ignores all batch files that it has previously processed.

6. Server B gets Server C's ICT. Similarly, Server B ignores all batch files that it has previously processed.

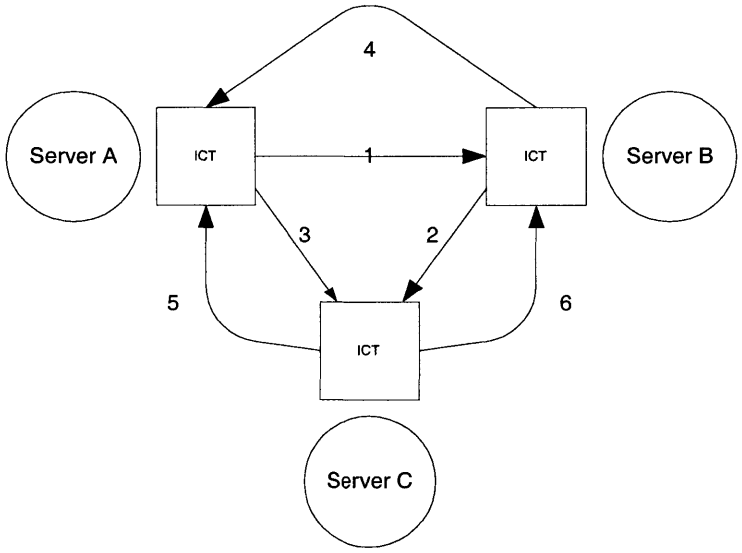

Figure 4-5: Synchronization Example

Since we anticipate only 10-15 servers in the system, it is feasible for each server to get updates from all others during every synchronization period, as outlined above. Alternatively, the system administrators for the S-BGP system can specify some subset of servers that any particular server needs to synchronize with. This subset can be carefully chosen to ensure that changes propagate throughout the whole system. These accesses should be done in the same order every day, and should be staggered. Note that the NOCs are not affected in this process since they are still be able to access the web interface. Initially, our servers synchronize with all the other servers.

Chapter 5

Security Mechanisms in the Repository

While Chapter 4 described the requirements and overall design of the repository, this chapter discusses the security mechanisms that are employed. It begins with Authentication and Confidentiality and then goes on to Integrity and Data Origin Authentication, which includes the use of digital signatures and storing the server's private key. It then gives a detailed discussion of the repository's Access Control mechanism. It details the S-BGP private extensions, how the certification path is used, and gives an algorithm for the authorization mechanism, which used the IP address space and AS identifier subset relationships. The chapter concludes with a description of logging and other security concerns.

5.1 Overview

When a NOC connects to the repository, it is authenticated by the web server by presenting an SSL certificate issued under the S-BGP hierarchy to the server. If this is successful, the NOC can now download files or upload files. For uploads, the batch files are processed by the Upload Processor. The Processor first validates the signature on the batch file. If the signature verifies, it then checks that the certification path provided in the batch file is also valid. When processing each individual transaction,

it runs the authorization algorithm on the particular object to ensure that the NOC operator is allowed to upload the data. This algorithm uses the S-BGP private extensions `SBGPIpAddrBlock`, `SBGPASNum` and `SBGPRouterId` that were introduced in Chapter 3 to perform these checks.

5.2 Authentication Mechanisms

In the repository, a multi-phase authentication scheme is used for uploads. First the upload session needs to be authenticated. This is done to prevent unauthorized upload sessions. When a client, i.e., a Network Operations Center (NOC), wants to upload files, it is authenticated by the web server using an SSL client certificate. These authentication tools are built in to the Apache OpenSSL server that we are using. Second, the uploaded file has to be authenticated. This is done during the actual processing of the file by verifying its digital signature. If the signature is invalid, this is treated as a fatal error and the file is rejected.

5.3 Confidentiality

Confidentiality refers to the disclosure of data in the system by policy only. Most policies restrict who can see the information within the system. In our repository, confidentiality is not a key concern as most of the data transmitted is public domain information. Instead, we are more focused on ensuring that the data transmitted to or from the repository, or stored within it, cannot be altered undetectably, either maliciously or by an error.

5.4 Integrity and Data Origin Authentication

Data integrity refers to ensuring that any data received by a NOC or server in the repository have exactly the contents that were sent. Data can be altered through a malicious attack or through an unintended communication error. The repository

therefore aims to prevent modifications by unauthorized users, as well as improper modifications by authorized users. Preventing unauthorized modifications is achieved by data origin authentication through the use of a certificate, which is presented to the web front-end. Each authenticated NOC is authorized to manage only a specific set of objects in the database and these modifications are controlled by the Authorization Algorithm described in Section 5.5.4.

5.4.1 Digital Signatures

Most of the data transferred to and from the repository must be signed. This includes the individual certificates, CRLs, and AAs in both the upload and download files, the actual upload files, the reply files generated when a batch file is processed, and the Incremental Changes Table (ICT that is used in synchronization). Integrity is achieved by having a NOC operator sign the upload files using a private key that is authorized for digital signature usage, which can then be verified using an end-entity certificate for the operator. The download files are not signed by the server. Instead, SSL ensures that the NOC operator is retrieving a file from the repository in question and protects the integrity of the file during download. The download files are validated using a number of checks including validating the signatures on each certificate, CRL or AA, checking the revocation status of the certificates, and verifying IP address space and AS number delegation.

Generating the Digital Signature for the ICT and Reply Object

In order to generate the digital signature, we use the server's private key. The server's public and private key pairs are generated at the time that the server is set up. Again, there may be two server certificates: an RSA certificate for the SSL session and a DSA certificate for signing the ICT files and the Reply object generated by the Upload Processor on processing a batch file. The private key of the DSA certificate is used to sign the data in the ICT and the Reply object, and the public key is used for verifying the signature by other servers and NOCs respectively. To sign the ICT, we supply

the software used for signing with the data, i.e., the file to be signed, generate the signature, and save the signature and the public key in files. Signing the Reply object is carried out in a similar manner; we supply it with the software and the reply data to be signed, generate the signature, and save the signed object to the Reply file.

Verifying the Digital Signature on the upload file

In order to verify a digital signature, we need:

- The data i.e., the `SignedTransactions` object from the batch file (see Appendix A.1 for ASN.1 specification)
- The signature (which is in the batch file)
- The public key corresponding to the private key that was used to sign the data (from the NOC operator's certificate, usually a DSA key)

We first get a signature object and initialize it with the public key for verifying the signature. We then supply it with the data whose signature is to be verified, and verify the signature. The NOC operator's certificate also needs to be verified using the certification path and trusted root certificate as discussed in Section 5.5.3.

5.4.2 Storing the Server's Private Key

As described earlier, certificates are used both for authentication and for verifying the integrity of the data. However, these certificates may not necessarily be the same. For instance, the operator certificate that is used for web authentication may not be the same certificate used for signing upload files. The reason for this is that common browsers (e.g., Netscape and Microsoft Internet Explorer) require that the subject have an RSA key, and the private key in the browser cannot be used to sign arbitrary objects such as the upload file. Therefore another certificate that typically contains a DSA key may be associated with the signature on the upload file i.e., a second operator end-entity certificate may exist. Similarly, the server also may have two certificates: an RSA certificate to authenticate itself to the NOC through the

browser, and a DSA certificate to sign the data in the reply and ICT files. Thus the server also has two key pairs.

The server must be able to store its public and private keys securely and must be able to keep the private key secret. It is recommended, where possible, that hardware be used to protect the private keys, and the reply files can be signed off-line or on a separate box and then transferred to the server. While this hardware protection is good, it has not been implemented for the repository. Furthermore, the compromise of the server could cause the hardware to sign bad data structures anyway.

If the private keys are compromised, then an adversary can use the SSL key to impersonate a server, and use its DSA key to fraudulently sign invalid download or ICT data. If a server is impersonated, a NOC could be fooled into "uploading" its data to the compromised server. The changes that it intended to make would therefore never be seen by the repository, and thus routers would never see the data needed to validate any routes that require this new information. Similarly, if the DSA key is compromised, a NOC can be tricked into downloading invalid data which it would then try to verify, or other S-BGP servers can get bad ICT data during synchronization, and pass the data on to all the NOCs that connect to them. Furthermore, if the keys were compromised, it is possible for an unauthorized deletion of data to occur, which is more difficult for the NOCs to detect.

5.5 Access Control for the Repository

In the Repository, we want to ensure that a NOC operator is authorized to perform the add/delete transactions that it has submitted for processing. When a NOC wants to upload files, it is authenticated by the web server using SSL, and then uploads a file. The file is placed in an input directory, a holding queue that is accessed by the server management software. The checks for whether or not the NOC is authorized to perform a given transaction are done by the Upload Processor and not the web server. Thus there is no requirement for the web server or client browser to interpret S-BGP certificate extensions.

Type of Certificate	S-BGP Extensions contained
Repository admin certificates	`SBGPIpAddrBlock` and `SBGPASNum`
Server EE certificates	`SBGPIpAddrBlock` and `SBGPASNum`
NOC operator certificates	`SBGPIpAddrBlock` and/or `SBGPASNum`
ISP/DSP/Org CA certificates	`SBGPIpAddrBlock` and/or `SBGPASNum`
Network EE certificates	`SBGPIpAddrBlock`
Router certificates	`SBGPRouterId`

Table 5.1: S-BGP Private Extensions Present in S-BGP Certificates

There are three S-BGP private certificate extensions: the IP address extension, the AS identifier extension, and the Router identifier extension. As mentioned in Chapter 3, the S-BGP PKI uses certificates with these extensions to convey authorization to use IP address blocks, AS numbers, or to bind a BGP router identifier to a specific AS. This authorization is done by checking the subset relationship, which refers to the prefixes for one of these extensions in one certificate being also included among the prefixes for that extension in another certificate. This is discussed in Section 5.5.4, but first we describe the S-BGP extensions in more detail.

5.5.1 S-BGP Certificate Extensions

Table 5.1 characterizes the certificates used in the S-BGP PKI, based on the use of both standard certificate fields and extensions, plus S-BGP-specific extensions.

The `BasicConstraints` extension contains the `cA` flag which is set to `TRUE` for the ISP/DSP/Org CA certificates. All other certificates in the table are end-entity certificates, and so the `cA` flag is set to the default `FALSE`. The S-BGP extensions provide methods for associating additional attributes with users or public keys and for managing the S-BGP certification hierarchy. There are three S-BGP private certificate extensions: the IP address extension, `SBGPIpAddrBlock`, the AS identifier extension, `SBGPASNum` and `SBGPRouterId`, the Router identifier extension. The ASN.1 specification of these extensions is given in Appendix A.2.

```
SBGPIpAddrBlock ::= {
 { addressFamily  '000101'H ,  -- IPv4 unicast
   ipAddressChoice  addressesOrRanges : { addressPrefix = '080A'H }  -- 10/8
 }
}
```

Figure 5-1: `SBGPIpAddrBlock` Example

IP Address Extension

The IP address extension, `SBGPIpAddrBlock`, in a certificate consists of a sequence of AFI or Address Family Identifier (usually IPv4 or IPv6) and SAFI or Subsequent Address Family Identifier (SAFI 1 - unicast, SAFI 2 - multicast, and SAFI 3 - both) entries. The root has all IP address blocks that can be allocated. However, there are some that cannot be allocated e.g., net 10 is for general use, while others e.g., nets 0 and 127 are reserved. Therefore these nets are not in the root certificates. Each registry's certificate contains the ranges for which that registry is responsible. Each AFI/SAFI entry contains either an "Inherit" flag or a sequence of address blocks. For example, we can have a sequence of address blocks as shown in Figure 5-1, which asserts that the operator can manage upload files containing certificates and AAs with IPv4 unicast prefixes 10/8 i.e., addresses through 10.0.0.0 through 10.255.255.255.

AS Number Extension

The AS identifier extension consists of a sequence of AS numbers as well as RDI (Routing Domain Identifier) entries. As for IP address blocks, the root has all AS numbers that can be allocated. Each registry's certificates contain the AS number ranges for which that registry is responsible. Each entry contains either an "Inherit" flag or a sequence of AS numbers. For example, we can have an extension as shown in Figure 5-2. This extension asserts that the operator can manage upload files that have certificates with AS number 1 in their `SBGPASNum` or `SBGPRouterId` extensions but that have no RDIs.

```
SBGPASNum ::= {
  asnum  asNumbersOrRanges : { num : 1 } -- AS Number 1
}
```

Figure 5-2: SBGPASNum Example

Router ID Extension

The Router identifier extension is used to identify router end-entity (EE) certificates. Any certificate containing a Router identifier extension does not contain either an IP address or an AS identifier extension and indicates that the certificate binds a public key to an AS number and to a BGP router's name and ID. Instead, the Router identifier extension has an AS number or an RDI that must be in the SBGPASNum extension of the operator certificate that is authorized to manage any certificate with this extension.

5.5.2 Boolean Inherit

Inherit is a boolean, which specifies whether or not a certificate is allowed to inherit a particular AFI/SAFI sequence, AS Number sequence, or RDI sequence from its issuer. Inherit is used only in NOC operator and repository related certificates i.e., the repository operator and repository CA certificates. If set to TRUE, then the subject associated with the certificate e.g., a NOC operator, is authorized to manage files containing certificates and AAs that have a subset of the AFI/SAFI entries and the ASnumber/RDIs that are in its (the operator's) issuer's certificate.

Figure 5-3 shows an example of an IP address extension which specifies the inheritance of IPv6 unicast addresses, so the operator can manage any objects containing IPv6 unicast prefixes (or their sub-nets) that are in the issuer's certificate.

```
SBGPIpAddrBlock :: = {
  { addressFamily   '000201'H ,  -- IPv6 unicast
    ipAddressChoice  inherit : { TRUE }  -- Inherit from issuer
  }
}
```

Figure 5-3: SBGPIpAddrBlock: Inherit = TRUE

Similarly, for the AS Identifier, we can have a sequence as shown in Figure 5-4. This sequence specifies the inheritance of AS numbers, so the operator can manage objects that have AS numbers that are contained in the SBGPASNum extension in its issuer's certificate, but not any containing certificates with RDIs.

```
SBGPASNum ::= {
   asnum    inherit : { TRUE }  -- Inherit AS numbers
}
```

Figure 5-4: SBGPASNum: Inherit = TRUE

5.5.3 Validation using the Certification Path

Certification paths are used in two forms in the repository: (1) when the SSL client certificate is presented to the web server for authentication, and (2) for upload files. While these two certification paths may be the same, we treat them differently since we are potentially considering two different certificates: an RSA certificate for authentication at the server, and a DSA certificate which is used to sign the upload files. Also, the paths are used by different processes.

When the web server is presented with an SSL client certificate, it needs to validate the certificate. The repository may not yet hold either an assured copy of the public key of the CA that signed the client certificate (since none of the certificates in the database are validated) or the CA's name and related information. It might therefore need the CA's certificate to obtain that public key. In general, we use a chain of multiple certificates that comprises a certificate of the client (the end entity) signed by one CA, and possibly one or more additional certificates of CAs signed by other CA certificates (which also become part of the chain). This chain is known as a certification path. Processing the certification path includes verifying basic information for each certificate, checking that the certificate has not been revoked, and ensuring that the issuer of the i^{th} certificate in the chain was the subject of the $(i-1)^{th}$ certificate in the chain.

The certification path in the upload file is the one that is stored in the database. The certification path in the upload file may work for both the NOC operator's SSL

certificate as well as the NOC operator certificate used to sign the upload file. It is constructed by the NOC and is never explicitly added to the `db_cpath` table. Instead, the Upload Processor makes changes to this table when it performs the transactions from a batch file that contained this path. The CA certificate that issued the operator certificate used to sign the batch file may not always be in the database. For example, when an ISP/DSP/Organization is first issued a CA certificate, this is not the case. For this reason, the certification path is included in the upload file. One benefit of this is that we never need to access objects in the database when checking authorizations, and so our process is quicker. Additionally, as we state in Section 4.4.2, this avoids any errors that may otherwise be generated if a certificate that should be in the certification path is not in the database.

Certification path validation for the path included in the upload file also utilizes the `SBGPIpAddrBlock` and `SBGPASNum` extensions in the certificates that are described in Section 5.5.1. In the validation process, we must check that each preceding certificate in the certification path has an `SBGPIpAddrBlock` extension that contains all of the IP address prefixes in the certificate being verified, recursively, until a trusted certificate has been verified. Similarly, each preceding certificate in the certification path must contain an `SBGPASNum` extension that contains all of the AS numbers and RDIs being verified, recursively, until a trusted certificate has been verified.

The certification path validation tool has been developed in the NOC tools suite and is used in the repository to validate the certification paths. The validation tool uses the Certificate Management Library (CML), which is government-sponsored freeware currently being maintained by Getronics [1]. One loads a set of trusted root certificates into the CML, and sets initial path validation policy parameters per S-BGP requirements. This function takes either a single ASN.1-encoded X.509 Certificate or a complete ASN.1-encoded CertificationPath as its parameter. It then outputs the validated public key and parameters returned by the CML, and returns either SUCCESS or an error code.

5.5.4 Authorization Mechanism

After the certification path is validated, the following additional checks are applied as part of the S-BGP extended validation process:

- IP address space subset relationship for certificates that bind a public key to an organization and to a set of IP address families and prefixes.
- AS identifier subset relationship for certificates that bind a public key to an organization and a set of AS identifiers.

The subset relationship refers to all prefixes for an S-BGP extension in one certificate, X, being also included among the prefixes for that extension in another certificate Y. To check the subset relationship, each AFI/SAFI entry in the `SBGPIpAddrBlock` extension and each AS number/RDI entry in the `SBGPASNum` extension is processed separately. For an operation to be authorized, we must check that each of the AFI/SAFI entries and each AS number/RDI entry or router identifier is valid. Only if all of these components are acceptable, then the operator certificate is allowed to perform database transactions using that object.

For the purpose of repository access control with regard to uploading, the following algorithm is applied to determine whether an operator certificate, X, is authorized to upload the data in question:

1. Check that the issuer of X (the issuer of the certificate that uploaded the batch file) is the same as the issuer of the certificate that signed the batch file.
2. Validate operator certificate using S-BGP extended validation algorithm.
3. For each certificate being uploaded,
   (a) If Y is the certificate belonging to X's CA, check that the subject name in Y is the name of the CA that issued X. Otherwise check that the issuer name in the certificate being uploaded, Y, is the name of the CA that issued X.

(b) If INHERIT = FALSE for any AFI/SAFI or AS number/RDI, then X is not authorized to upload any object containing that AFI/SAFI in the `SBGPIpAddrBlock` extension or that AS number/RDI in the `SBGPASNum` or `SBGPRouterId` extensions.

(c) If INHERIT is not present,

i. Check that the AFI/SAFIs prefixes in Y are all subsets of the `SBGPIpAddrBlock` extension in X.

ii. Check that the AS numbers/RDIs in Y are all subsets of the `SBGPASNum` extension in X.

iii. Check that any Router ID extension in Y contains an AS number/RDI that is one of the AS numbers/RDIs in the `SBGPASNum` extension in X.

(d) If INHERIT = true for X,

i. Check that the AFI/SAFI prefixes in Y are all subsets of the `SBGPIpAddrBlock` extension that X inherits from its issuer's certificate.

ii. Check that the AS numbers/RDIs in Y are all subsets of the `SBGPASNum` extension that X inherits from its issuer's certificate.

iii. Check that any Router ID extension in Y contains an AS number/RDI that is one of the AS numbers/RDIs in the `SBGPASNum` extension that X inherits from its issuer's certificate.

4. For each address attestation being uploaded,

(a) Check that the network certificate, Z, that signed the AA has the same issuer as the issuer of X

(b) If Inherit is not present, check that the network prefixes in the AA are all subsets of the prefixes in X,

(c) If INHERIT = TRUE, check that the network prefixes in the AA are all subsets of the prefixes that X inherits from its issuer.

5. For CRLs, check that the issuer name in the CRL is the name of the CA that issued X.

Table 5.2 gives some examples of valid and rejected upload data for the IP address extension. Consider the following certificates:

- Certificate A: A CA certificate, with subject A and issuer R, and an `SBGPIpAddrBlock` extension containing all addresses with IPv4 unicast prefixes 18/8 i.e., addresses 18.0.0.0 to 18.255.255.255.
- Certificate B: An Operator EE certificate, with issuer A and subject B, and an `SBGPIpAddrBlock` extension with `inherit : TRUE`.
- Certificate C: An Operator EE certificate, with issuer A and subject C, and an `SBGPIpAddrBlock` extension with `inherit` not present, and containing IPv4 unicast prefix 17.18/16.
- Certificate D: An Operator EE certificate, with issuer A and subject D, and an `SBGPIpAddrBlock` extension with `inherit` not present, and containing IPv4 unicast prefix 18.10/16.
- Certificate E: A Network Certificate, with issuer A, and network prefixes 18/8.
- CRL K, with issuer name A.
- CRL L, with issuer name M.
- AA Y, signed by E, with network prefix IPv4 unicast 18/16.

5.6 Logging

All transactions in the repository are logged on disk. These include:

- Administrative logins and changes.
- Accesses by NOC operators.

Operator Certificate	Object to be Uploaded	Status
B	Certificate A	Valid
B	Certificate C	Reject
B	Certificate D	Valid
B	Certificate E	Valid
B	CRL K	Valid
B	CRL L	Reject
B	AA Y	Valid

Table 5.2: Examples of Upload Data

- Individual object Additions/Deletions.
- Downloads by NOC operators.
- Retrievals of Incremental Change Files by servers for synchronization.

Logging in the repository involves two separate files: an audit log and an error log. The audit log records the transactions done at the server while any errors generated at the repository are written to the log file. Ideally, in order to ensure the security of these logs and make them tamper-proof, we would use a write-once media. However, this is too expensive for our purposes so we try to minimize security issues by only allowing administrators of the repository access to them after they are authenticated by the system.

When a transaction is about to be performed on a server, an entry is made to the audit log indicating this. This entry consists of a "Begin Transaction" message, the type of transaction, the entity carrying out the transaction, the handle(s) of the object(s) involved, the first 16 bits of the hash of the object and a timestamp. After the transaction is complete, a new message is entered into the log saying "Transaction Complete". Since our Processors only perform a single transaction at a time, there is no need to worry about multiple NOCs performing many different transactions simultaneously. Note that this is different from the web front-end which allows multiple processes for uploading and downloading files to the "Input" directory and to the "Download" directory.

The audit log also has a secondary function. It is used to record valid additions and deletions. For these transactions, an entry exists in the log. In this way, if an administrator needs to determine if a transaction was valid, he or she can simply check through the entries in the log until he or she finds an entry for that particular object, and ensure that it was indeed legitimate. This may occur if there are inconsistencies among the servers, and the administrator wants to determine which object is valid.

The error log is used to record all failed transactions on the system. This allows the administrator to know what went wrong on the system. Failed attempts are also important because they may signal attacks on the repository. The error log contains short events that need attention and so should be examined every day.

5.7 Other Concerns

While the earlier part of this chapter detailed the mechanisms in the repository designed to provide authenticity, integrity of data, and access control, we also need to consider other aspects of the system in order for it to be truly secure.

5.7.1 Physical Protection for the Server

Physical access to the servers must be controlled. The servers should be placed in an access-controlled room so that only authorized personnel can gain physical access to them. This way, we avoid Denial-of-Service attacks in which an unauthorized user deliberately shuts down the system, as well as decrease the probability of any unauthorized modification of the data. Operations staff with the necessary authorization can then perform any maintenance needed by the servers. It is noteworthy that the server is apt to be located at an unmanned facility so administrators need secure, remote access. For example, SSL can be used since it use of certificates, which are consistent with the S-BGP PKI.

5.7.2 Backups

The data on the servers should be backed up to off-line storage on a regular basis. Backups help with recovery and data integrity through the use of redundancy. Since we have multiple copies of the repository, backups of the database are not critical. Furthermore, since there is no one to physically supervise the backup, they also need to be done remotely and securely. The backup process should include copying the audit file and the log file.

Chapter 6

Implementation Details

This chapter gives some of the implementation details associated with building the different pieces of repository. It includes the specifics related to the pieces of the repository including the database, the interfaces, and the actual server that hosts it. It also talks about the data transfers including uploads, download and synchronization. Finally it gives a recap of the Security Implementation which has been described in Chapter 5 in detail.

6.1 Overview

Figure 4-2 on page 46 shows the Client Interface and Figure 4-4 on page 60 shows the Server Interface of the repository, including the specific Processors and directories that would be used. Files uploaded at the web interface go to the "In" directory. From there, they are retrieved by the Upload Processor and processed. If the file is of an incorrect format, or the signature is invalid, it is put in the "Reject" directory, and an appropriate reply file is generated and sent to the "Reply" directory for retrieval by the NOC that uploaded the file. Otherwise, the transactions in the batch file are processed, the batch file is placed in the processed directory, and its name is written to the ICT. The Download Processor builds the download files and places them in the "Out" directory where the NOC can access them from the web interface.

6.2 Server Details

As mentioned in the previous chapter, the server used for the repository needs support for two-way SSL authentication. The Apache HTTP server does not have security built in, but there are packages that can be added to it to provide this support. We use OpenSSL [15], which is a cryptography toolkit that implements the Secure Sockets Layer (SSL) and related cryptography standards. In this way we provide a server with built-in security and capability for authentication. The clients (NOC operators) access the server using secure HTTP (HTTPS) from web browsers, e.g., Netscape (TM) or Microsoft Internet Explorer (TM).

6.3 Database

PostgreSQL database [11] is a powerful relational database. It is an enhancement of the original postgres database management system that retains the data model and rich data types of postgres, but replaces the PostQuel query language with an extended subset of SQL. This database is able to manage large numbers of objects, thus making it very useful for the S-BGP implementation. However, the objects are not stored in their binary formats in the database. Instead, six-bit encoding is used since PostgreSQL, like most other relational databases, has limitations on the type of object it can store as it cannot handle the null byte that appears in several places in a DER-encoded object.

Existing PostgreSQL database methods were modified to reflect the database schema that was used, as well as to give added functionality needed for the data objects. These methods are listed in Appendix B.

6.4 Client Interface

The NOC connects to the repository via a webpage. It uses an SSL certificate installed in a client web browser to authenticate the NOC to the Apache server and allow it to upload and download files. The web server also has an SSL certificate, which is

validated by the web browser. A screen shot of the web interface is shown in Figure 6-1. It shows the main functions offered by the Client Interface: "Upload Transaction Files" and "Download Objects". The Client Interface also encompasses the Upload, which is used to process the files that are uploaded, and Download Processors, which generates the download files.

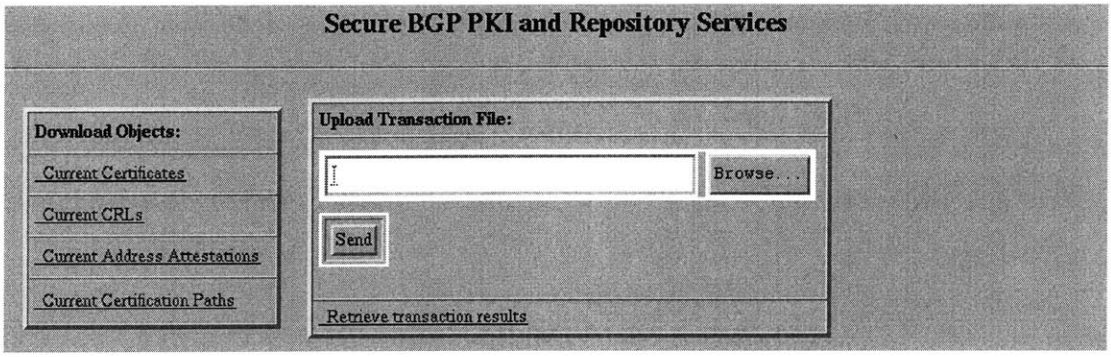

Figure 6-1: Screen Shot of Web Interface

6.5 Administrative Interface

The Administrative Interface is implemented as a command line interface. Figure 6-2 shows some of its screens. As required, it allows for viewing the audit and log files and for performing the database functions of add, delete and search when presented with a copy of the object. As mentioned in Chapter 4, an extended search mechanism will be implemented in the future. The interface returns a message to the administrator to indicate whether or not his operation was successful (Screen 4).

Currently, there is no way for other servers to get changes made by an administrator. While this may be implemented later, we currently specify that any modifications made by an administrator must be done at every single server on which he or she wants the change to be made. We expect that administrators do not make frequent changes, and since each transaction should take less than 30 seconds, changing the data at every server is feasible. However, Chapter 7 describes a mechanism in which a file similar in structure to a batch file is built and propagated during synchronization.

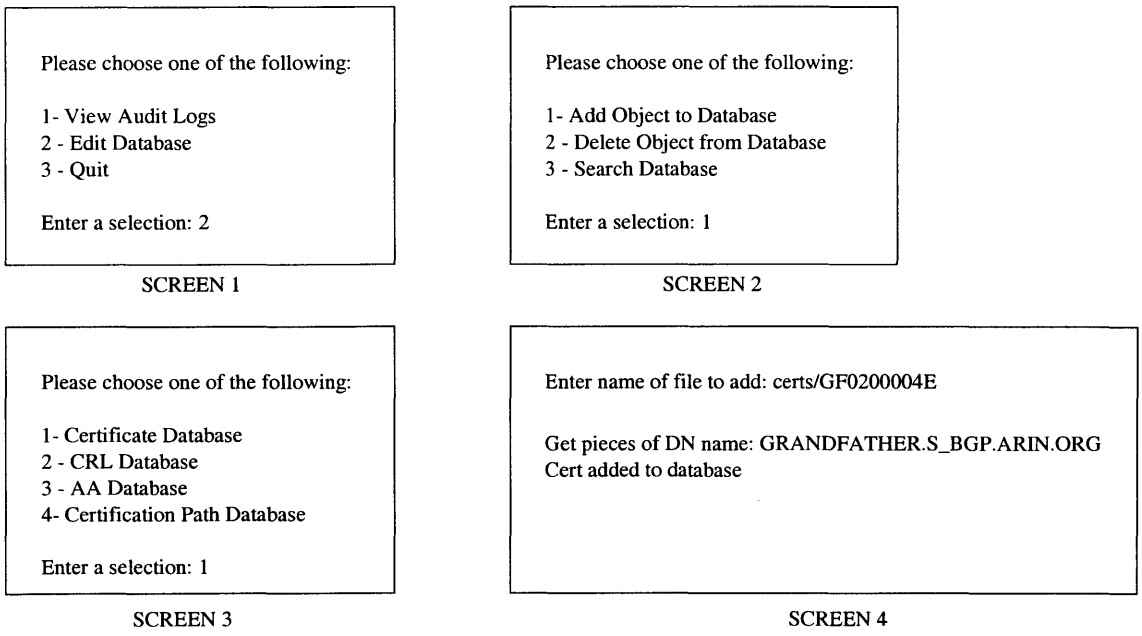

Figure 6-2: Select Screens from the Administrative Interface

6.6 Uploads to the Repository

The NOC operator uploads a file to the repository after the necessary authentication checks. These files are placed in an "In" directory. The Upload Processor then goes through that directory, sorts the files in order of arrival, and then processes them one at a time, the earliest first.

The upload files have the following name structure:

`yyyymmdd.hhmmss.mmm.<DC1>.<DC2>...<DCn>.hashhashhashhash`

The date and time is placed into `yyyymmdd.hhmmss.mmm`, the NOC operator's certificate's ISSUER name is in the DCs which together give the DNS name, and the first 8 octets of the SHA-1 hash of the signed transactions are appended. The date and time help to sort the files in the directory so we can process them in order of upload, while the issuer name and hash can help to detect "replays" from a NOC i.e., if the same file is uploaded at different times.

If the file is good, it is renamed and placed into a "Processed" directory. The file name is also written to the ICT. However, if the batch file gives an intolerable error,

i.e., is of an incorrect format, or the signature is invalid, it is put in the "Reject" directory and no entry is written to the ICT. This prevents bad files from being propagated to other servers in the system. In both cases, a reply file is generated containing the same structure as described in Section 4.2.2. Replies to these files are named by appending a "`.reply`" for easy retrieval by the NOC, assuming that the NOC remembers the name of the files it uploads. Note that good files may also generate error reply codes if single transactions within that file fail. We do not attempt to remove those bad transactions but just leave the file as is. When the NOC retrieves the reply file, it sees the error codes for the particular transactions and so has the responsibility of fixing them.

6.7 Database Downloads

A secure HTTP mechanism is used for the NOC to get the database downloads. The NOC uses the web interface to request the database files which are stored in a special "Out" directory on the server. The database files are generated using the Download Processor which goes through a particular database and builds a single file containing all the objects from that database. The files are then moved to the "Out" directory and replace the older files that were there.

6.8 Synchronization

During synchronization, server A requests the ICT file and the processed upload files from server B. This process is authenticated as described in Chapter 5. When server A gets the ICT, it puts it in an "ICT_in" directory, and puts the batch files in a "serverbf" directory. It then goes through all the ICTs in the ICT_in directory and uses the Synchronization Processor to process them serially. It does this by retrieving each file name from the ICT, checking if it had been processed before and if not, using the Upload Processor to process that particular file. We make use of the fact that batch file names are unique in order to perform this check for duplicate batch files.

First, we try to open the file with that file name in the Processed Directory. If the file is successfully opened, then it has already been processed and there is no need to do it again. However if it has not yet been processed, an error is returned, and we then process the file. If we get a fatal error when we process a batch file listed in an ICT, then that file is placed in the Reject directory and no entry is written to the server's own ICT. Otherwise, the batch files are placed in the Processed directory. When an ICT is processed, it is then placed in the "ICT_processed" directory. Note that when a batch file is processed, its name is written to the server's own ICT, so the ICTs from other servers do not need to be propagated to other servers. In this way, Reply files are generated at each server for every file uploaded, whereas in reality the replies are only retrieved from one server. Again we assume that the files can be deleted some fixed time after being processed to preserve disk space.

The connection between the servers for synchronization is very much like the client-server SSL connection that the NOCs use to upload and download files. If Server A is getting files from Server B during synchronization, then Server A acts as the "client" and Server B acts as the "server". Server A then fetches the ICT and batch files from the "server" using the same HTTPS `GET` mechanism that a NOC would use to download files from the repository.

6.9 Security

Chapter 5 describes the mechanisms in detail. Access control for the data in the repository consists of both authentication and authorization mechanisms. When a NOC accesses the repository, it is authenticated by the web server using an SSL client certificate. These authentication tools are built-in to the Apache OpenSSL server. The web server also validates the certification path of the NOC's SSL certificate, which does not come from the path in the upload file, but instead is built by the SSL engine and passed as part of the SSL handshake. After the NOC is authenticated, it is allowed to retrieve any of the database download files since these can be accessed by anyone. To upload data, however, the NOC presents a signed file containing all the

transactions it wants to perform. First, the signature on the file must be validated by the server (this is done in software). Second, we check that the issuer of the uploader's certificate of the file is the `submitter` as listed in the batch file. Next, the certification path from the upload file must be validated. This certification path may be different from the path verified by the web server for the SSL certificate. This is done using software tools developed for the S-BGP project. If there is no error, the access control algorithm that is described in Chapter 5 is used to determine whether the file signer's operator certificate is allowed to perform database transactions for each of the objects in the batchfile. If all these checks are completed without error, the server processes the batch file and generates the reply file which it then signs. Again, this is carried out in software.

Chapter 7

Future Work and Conclusion

This chapter gives an overview of the future work related to the S-BGP repository and in the S-BGP project. It discusses possible additions to the repository including incremental downloads, better search capabilities, the use of an email option for reply files, and allowing administrator changes to propagate during synchronization. It then gives the next steps for the integration of the client and server parts of the repository, as well as activities intended to enable the deployment of S-BGP on the Internet. Finally, it gives a conclusion for this thesis.

7.1 The S-BGP Repository

Future work on the repository can be separated into two areas. The first area involves possible ways in which the functionality of the repository can be improved while the second area deals with the integration of the current implementation of the NOC tools and the server functions.

7.1.1 Possible Additions to the Repository

The current implementation of the repository is limited in certain aspects. It has the minimal functionality needed to perform the tasks that it must support. There are enhancements that could be made to the system in order to make it more efficient

and user-friendly including: (1) incremental downloads to improve performance, (2) search capabilities, (3) email as transport for Reply files, and (4) improvement in synchronization. Each is addressed individually here.

Incremental Downloads

Currently, when a NOC wants to download data from the repository, it must retrieve each of the four database files. Each of these files contains all of the objects of a given type that is stored in the database. It is reasonable to allow NOCs to download incremental changes to the repository instead of the whole database. Incremental downloads are only useful if the Upload Processors process the changes to the repository more often than the download files are generated. Otherwise, if there are no new changes, then the incremental files would be empty.

Incremental downloads could be done in two ways: (1) allow NOCs to only download changes to the database since the last time it downloaded data from the repository, or (2) generate incremental files similar to the full database files but which only contain the objects that were modified within a given period. The first method will significantly decrease the number of items in each download file, and thus the actual downloads, as well as NOC processing of these files, will take less time. Also, it gives the NOC control over how often it can get the changes made to the repository, i.e., the NOC can choose to do incremental downloads three times a day, whereas in (2) or in full database downloads, any changes made to the repository after the download files are generated cannot be obtained by other NOCs until the next set of files are made. Again, this assumes that the Upload Processor is run frequently so that batch files are processed more frequently than the NOC retrieves the download files. The overall system will therefore be more efficient. One drawback to incremental downloads, however, is that the download file has to be generated after it is requested by the NOC. It is possible that we can have as many as 500 NOCs connecting to a single site everyday (since we estimate 5,000 clients and approximately 10-15 servers), which leads to more processing time on the part of the server. The second method is very similar to the full database downloads, except that the files contain fewer transactions

and are smaller. This makes the overall system more efficient since they will be faster to download and the NOC will have fewer items to validate on a day to day basis. However, the download file formats would need to be changed to keep track of what types of transactions were performed, i.e., adds or deletes.

Better Search Capabilities

The search capability implemented by the repository is minimal. It takes an object (certificate, CRL, AA or certification path) and checks whether or not the object is in the database. This can be extended to search for objects by particular fields without producing a copy of the object. This provides the ability to retrieve all objects with a given attribute, which can be useful to the NOC. For example, we can extend the Administrative Interface to allow administrators to search for certificates in the database by issuer name and/or serial number. In this way, we use the search function to locate all of the certificates issued by a particular organization without having to download the whole database and sorting through it. This search capability can be implemented through the use of the Lightweight Directory Access Protocol (LDAP) [7]. LDAP can be used as a front-end interface to the PostgreSQL to retrieve objects based on the attributes (in this case the search indices). The objects can then be returned to the administrator.

Email Option for Reply Files

Another possible extension to the repository is to support email error messages for upload files. This can be used instead of waiting for the NOC to retrieve the reply files. Email replies would potentially allow NOCs to receive notification of errors more quickly since email would be almost instantaneous after the Upload Processor is run. It also gives the added benefit of convenience since the NOC no longer has to connect to the repository and download the replies. This feature should not be difficult very implement since the upload file already contains an `email` field as mentioned in Section 4.4.2 that can be used as the email address to which we send the replies that are generated. However, it does require an additional step of base64 encoding the

reply message for transport via SMTP.

Synchronization Changes

One attribute that can be added for synchronization is to allow changes made by an administrator on a single server to be propagated to the other servers during this process. This can be achieved by making a special file with a similar structure to the upload files, except with the administrator's name in place of the client's name as the submitter. The file can then be signed by the server as it would sign an ICT. During the synchronization process, this file is transmitted with the other batch files, and processed in the same way as each of them. We therefore gain the advantage of not needing the administrator to make the changes at every server in the system.

7.1.2 Client-Server Integration

This thesis discusses the server-side design and implementation of the S-BGP repository. The client-side tools and applications are also being developed simultaneously. These tools will be used to actually generate the certificates, CRLs and AAs that are to be stored in our databases, as well as the upload files that provide these objects and other changes to the repository. The NOC tools will also be able to process download files which are retrieved from the database, and to generate the certificate and AA extraction files which are used by the routers. The next step will be to integrate the repository and the NOC tools.

Performance Analysis

The system needs to be tested as a whole to ensure that all the requirements are indeed met, as well as to determine its actual performance with "real" uploads and downloads. The transfer of data between the NOCs and the server needs to be relatively short. We also need to determine how efficiently the NOC can generate AAs and certificates and upload them, and how quickly the server can process them, as well as how well the NOCs process the download files. It is worthwhile to note

that this task is dependent on the actual data for the certificates and AAs, which have not yet been made available by the registries.

7.2 S-BGP: The Project

In order to provide a proof of concept demonstration, BBN developed a prototype implementation of S-BGP and deployed it in DARPA's CAIRN testbed. Real Internet BGP control traffic was fed to the testbed routers via replay of a recorded BGP peering session with an ISP's BGP router. The results of the tests supported prior analysis which suggested that the overhead added by the S-BGP countermeasures needed the CPU/memory equivalent of a desktop PC (see [4] for more details on S-BGP performance in these tests).

However, getting the relevant organizations to accept and use S-BGP has a number of challenges. Firstly, deployment of S-BGP on the Internet requires the participation of various types of organizations including the registries such as ARIN, RIPE and APNIC, router vendors, and ISPs. Since there are not significant security benefits unless a few of each type of the organizations adopt the protocol, we need to find ways to encourage each organization to invest in this new technology.

Therefore, the next step in the S-BGP project towards the deployment of the protocol involves closely working with these organizations. One aspect of this plan is to work with the Internet registries and ISPs to set up the Public Key Infrastructure described in Chapter 3, and gain experience with the relevant policies and procedures. A Certification Authority (CA) will be set up at an Internet Registry, ARIN, including the installation of the CA system, and training for operations staff. ARIN will also provide its database of IP addresses and AS numbers which would allow for the creation of an initial set of certificates and attestations to cover the existing address and AS number allocations that would allow the organizations to begin using the repository. We would then work with all the different organizations to replace these certificates that are stored with valid ones. ARIN will also specify how the requests for IP addresses and AS numbers would work, and how and when these new certificates

would actually be generated.

The project team also plans to work with router vendors to integrate S-BGP countermeasures into their products. Additionally, BBN will also perform outboard tests with ISPs. This means that before routers that support S-BGP are available for ISP evaluation, the protocol will be tested with existing routers and a PC system running the S-BGP software. This will help the ISPs to integrate the tools, policies and procedures that have been developed to support the NOC operations which are needed for S-BGP. Policies also need to be developed for how S-BGP will work while in the incremental deployment stage. There must be some consistent way for routers to handle path information in which some of the ASes use S-BGP, and others do not.

7.3 Conclusion

The Border Gateway Protocol (BGP) is a critical part of Internet routing infrastructure. However, it is vulnerable to many attacks as it has no secure means of verifying the authenticity and legitimacy of BGP control traffic. The Secure Border Gateway Protocol (S-BGP) introduces Public Key Infrastructure technology, a new BGP path attribute containing "attestations" and IPsec. It uses these mechanisms to create an authentication and authorization system that addresses most of the security problems associated with BGP.

The security approaches used by S-BGP rely on the distribution of countermeasure information. Since putting certificates in UPDATE messages is inefficient, and storing the database of certificates, CRLs and AAs on S-BGP speakers is difficult due to memory restrictions, we therefore design and implement a repository that provides for the storage and efficient distribution of certificates, CRLs and address attestations (AAs) in S-BGP.

The repository consists of server-replicated, easy-to-access storage sites. Each site consists of a database which is used to store the actual objects, as well as a web interface through which the NOCs interact with the repository. All objects transferred to and from the repository are signed. The repository supports two forms

of transactions: uploads and downloads. Downloads transfers the whole contents of each of the databases in the repository, i.e., all the objects of a particular type, and are authorized for all members of the S-BGP community. Uploads are performed in the form of batch files which contain a set of data items associated with a specified NOC. Checks are made using our access control algorithm to determine whether or not a NOC is authorized to upload the data in question, and then individual transactions are performed. Synchronization among the servers is achieved by using an Incremental Change Table which keeps track of all the uploads performed within a specific time period. Furthermore, accesses to the repository are denied unless the NOC is authenticated, i.e., the client must have an SSL certificate issued under the S-BGP PKI.

In conclusion, the repository satisfies the requirements for storage, uploads, downloads and synchronization. It also provides the necessary authentication and access control mechanisms for the enhancements introduced by the new S-BGP countermeasures. Overall, it is feasible to build such a repository that enables the efficient distribution of the S-BGP countermeasure information, and thus supports the deployment of S-BGP.

Bibliography

[1] Getronics Web site. http://www.getronicsgov.com/hot/cml_home.htm. May 2001.

[2] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile. Technical report, MIT LCS TR429, October 1998.

[3] The IP Security Protocol Working Group Web site. http://www.ietf.org/html.charters/ipsec-charter.html. May 2001.

[4] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (S-BGP) - Real World Performance and Deployment Issues. Technical report, BBN Technologies, 2000.

[5] Stephen Kent, Charles Lynn, and Karen Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, April 2000.

[6] Stephen Kent, Charles Lynn, and Karen Seo. Public-Key Infrastructure for the Secure Border Gateway Protocol (S-BGP). Technical report, BBN Technologies, 2001.

[7] Innosoft Internation Inc., Lightweight Directory Access Protocol (Version 3) Specification. http://www.critical-angle.com/ldapworld/ldapv3.html. May 2001.

[8] Charles Lynn, Joanne Mikkelson, and Karen Seo. IETF Internet Draft, Secure BGP (S-BGP). Technical report, BBN Technologies, 2000.

[9] R. Perlman. Network Layer Protocols with Byzantine Robustness. Technical report, Network Working Group, January 1999.

[10] Larry L. Peterson and Bruce S. Davie. *Computer Networks, A Systems Approach.* Morgan Kaufmann Publishing, 1999.

[11] PostgreSQL Web site. http://www.postgresql.org/. May 2001.

[12] RSA Web site. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-8/index.html. May 2001.

[13] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4). Technical report, Network Working Group, March 1995.

[14] J. H. Saltzer, D.P. Reed, and D. D. Clarke. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 1984.

[15] OpenSSL Web site. http://www.openssl.com. May 2001.

Appendix A

S-BGP ASN.1 Specifications

A.1 S-BGP Upload and Download File Formats

```
-- File:     s-bgp.asn
-- Contents: ASN.1 specification - S-BGP Repository Batch Files
-- System:   S-BGP Repository.
-- Created:  14-Dec-2000
-- Author:   Kavita Baball

-- COPYRIGHT 2000-2001 BBN Technologies

DEFINITIONS ::= -- explicitly encoded !

IMPORTS
    AlgorithmIdentifier FROM Algorithms IN Algorithms.asn,
    Certificate FROM Certificate IN certificate.asn,
    CertificateRevocationList FROM CertificateRevocationList IN v2crl.asn,
    CertificationPath FROM CertificationPath IN certificate.asn,
    KeyIdentifier FROM KeyIdentifier IN extensions.asn,
    Name FROM Name IN name.asn,
    SerialNumber FROM SerialNumber IN serial_number.asn;

EXPORTS S-BGP-Repository-Transactions,
S-BGP-Repository-Replies,
S-BGP-Repository-Downloads;

S-BGP-Repository-Transactions ::= SIGNED SignedTransactions

SignedTransactions ::= SEQUENCE {
```

```
    serialNumber              SerialNumber,
    submitter                 Name,
    keyIdentifier             KeyIdentifier, -- of submitter
    email                     IA5String OPTIONAL,
    certificationPath         CertificationPath,
    transaction-list          Transactions
    }

Transactions ::= SEQUENCE OF RepositoryTransaction

RepositoryTransaction ::=  SEQUENCE {
    transaction-number                INTEGER,
    dependency-transaction-numbers    Dependencies OPTIONAL,
    operation-type                    OperationType,
    object-type                       ObjectType,
    object                            ANY DEFINED BY object-type
    }

Dependencies ::= SEQUENCE OF INTEGER

OperationType ::= ENUMERATED { addObj ( 1 ), deleteObj ( 2 ) }

ObjectType ::= ENUMERATED { cert ( 1 ), crl ( 2 ), aa ( 3 ), certpath ( 4 ) }
            -- certpath only in download

S-BGP-Repository-Replies ::= SIGNED SignedReplies

SignedReplies ::= SEQUENCE {
    serialNumber              SerialNumber,
    submitter                 Name,    -- from SignedTransactions submitter
    repository                Name,
    keyIdentifier             KeyIdentifier, -- of repository
    certificationPath         CertificationPath, -- of repository
    reply-list                Replies -- Transaction reply codes
    }

Replies ::= CHOICE {
    repository-reply          SEQUENCE OF RepositoryReply,
    file-reply                FileReply
    }

FileReply ::= ENUMERATED {
    bad-signature ( 1 ),          -- Bad signature on file
    wrong-format ( 2 ),           -- File format not valid
    bad-user ( 3 ),               -- Unauthorized user
```

```
    invalid-cpath ( 4 )                -- Invalid certification path
}

RepositoryReply ::=  SEQUENCE {
    transaction-number          INTEGER,
    operation-type              OperationType,
    object-type                 ObjectType,
    response                    TransactionReplyCodes
    }                                                                    80

TransactionReplyCodes ::= ENUMERATED {
    ok ( 1 ),                          -- Successful
    duplicate ( 2 ),                   -- Transaction previously done
    dep-failure ( 3 ),                 -- Dependency Failure
    exists ( 4 ),                      -- Key in use
    access-denied ( 5 ),               -- Insufficient permissions
    invalid-operation ( 6 ),           -- Invalid Operation
    invalid-obj-type ( 7 ),            -- Invalid Object Type
    bad-object ( 8 ),                  -- Invalid Object Format          90
    database-error ( 9 )               -- Other
    }

S-BGP-Repository-Downloads ::= SEQUENCE {
    databaseTime                GeneralizedTime,
    repository                  Name,
    object-type                 ObjectType,
    objects                     Objects
    }
                                                                         100
Objects ::= SEQUENCE OF RepositoryObject

RepositoryObject ::=  ANY DEFINED BY object-type IN
S-BGP-Repository-Downloads
```

A.2 S-BGP Extensions

```
-- File:      extensions.asn
-- Contents: ASN.1 specification - X.509 certificate (version 3)
-- System:    S-BGP server development.
--

-- PKIX Object Identifiers
id-pkix                         OBJECT IDENTIFIER ::= { 1.3.6.1.5.5.7 }
```

```
id-pkix-pe                      OBJECT IDENTIFIER ::= { id-pkix 1 }

id-pe-sbgp-ipAddrBlock          OBJECT IDENTIFIER ::= { id-pkix-pe 7 }
id-pe-sbgp-autonomousSysNum     OBJECT IDENTIFIER ::= { id-pkix-pe 8 }
id-pe-sbgp-routerIdentifier     OBJECT IDENTIFIER ::= { id-pkix-pe 9 }


   SBGPIpAddrBlock      ::= SEQUENCE OF IPAddressFamily

   IPAddressFamily      ::= SEQUENCE {
     addressFamily        OCTET STRING (SIZE (2..3)), -- AFI & opt SAFI
     ipAddresses          IPAddressesOrRanges }

   IPAddressesOrRanges ::= SEQUENCE OF IPAddressOrRange

   IPAddressOrRange     ::= CHOICE {
     addressPrefix        IPaddress,          -- IPAddress collision
     addressRange         IPAddressRange }

   IPAddressRange       ::= SEQUENCE {
       min                IPaddress,
       max                IPaddress }

   IPaddress            ::= BIT STRING

   SBGPASNum            ::= SEQUENCE {
      asnum         [0] IMPLICIT ASNumbersOrRanges OPTIONAL,
      rdi           [1] IMPLICIT ASNumbersOrRanges OPTIONAL}

   ASNumbersOrRanges    ::= SEQUENCE OF ASNumberOrRange

   ASNumberOrRange      ::= CHOICE {
      num             ASNumber,
      range           ASRange }

   ASRange ::= SEQUENCE {
      min             ASNumber,
      max             ASNumber }

   ASNumber ::= INTEGER

   SBGPRouterId ::= SEQUENCE OF OwningASNumber

   OwningASNumber       ::= CHOICE {
     asnum          [0] IMPLICIT INTEGER,    -- autonomous system number
```

```
rdi             [1] IMPLICIT INTEGER }  -- routing domain identifier
```

Appendix B

Database Implementation

These are descriptions of the database functions that were written or modified for the repository.

B.1 Creating the Database

Function: `void create_tables_update_config`

Description: This routine creates tables for the database and updates the config file if operation is successful. Otherwise, exits if operation fails.

Arguments: `config_file` and database name passed globally.

Return Value: None; Exit for failure

Function: `int db_create_table_db_cert`

Description: This routine creates the table `db_cert` for the database.

Arguments: Database name passed globally.

Return Value: SUCCESS if no error; PG_NULL_DATABASE if no database name is given; DB_OPEN_ERROR if cannot open the database ; FAILURE for other failures.

Function: `int db_create_table_db_crl`

Description: This routine creates the table `db_crl` for the database.

Arguments: Database name passed globally.

Return Value: Same as for `int db_create_table_db_cert`

Function: `int db_create_table_db_aa`

Description: This routine creates the table `db_aa` for the database.

Arguments: Database name passed globally.

Return Value: Same as for `int db_create_table_db_cert`

Function: `int db_create_table_db_cpath`

Description: This routine creates the table `db_cpath` for the database.

Arguments: Database name passed globally.

Return Value: Same as for `int db_create_table_db_cert`

B.2 Adding and Deleting Objects

The higher level functions were called directly by the Adminstrative Interface and the Upload Processors. These in turn used the Postgres functions to build and execute the query to give the required modification to the database.

B.2.1 Higher Level Functions

Function:`int write_cert_to_database`

Description: This procedure writes a certificate to the database.

Arguments: conn - Pointer returned from DBI_OPEN, hashIssuerP - hash value of issuer DN, issuerP - issuer DN in DNS format, snP - serial number, cpath_seq - sequence number of the certification path associated with the object, cert - Certificate passed in globally

Return Value: SUCCESS if a certificate is successfully written to the database, or appropriate negative values for errors; FAILURE for other errors

Function:`int write_crl_to_database`

Description: This procedure writes a CRL to the database.

Arguments: conn - Pointer returned from DBI_OPEN, hashIssuerP - hash value of issuer DN, issuerP - issuer DN in DNS format, cpath_seq - sequence number of the certification path associated with the object, crl - CRL passed in globally

Return Value: Same as for `int write_cert_to_database`

Function:`int write_aa_to_database`

Description: This procedure writes an AA to the database.

Arguments: conn - Pointer returned from DBI_OPEN, aaHashP - hash value of AA, cpath_seq - sequence number of the certification path associated with the object, aa - AA passed in globally

Return Value: Same as for `int write_cert_to_database`

Function:`int write_cpath_to_database`

Description: This procedure writes a certification path to the database.

Arguments: conn - Pointer returned from DBI_OPEN, cpathHashP - hash value of certification path, cpath - certification path passed in globally

Return Value: `seq` if a certification path is successfully written to the database, or appropriate negative values for errors; FAILURE for other errors

Function:`int delete_cert`

Description: This procedure deletes a certificate from the database.

Arguments: conn - Pointer returned from DBI_OPEN, hashIssuerP - hash value of issuer DN, issuerP - issuer DN in DNS format, snP - serial number, cpath_seq - sequence number of the certification path associated with the object, cert - Certificate passed in globally

Return Value: SUCCESS or FAILURE

Function:`int delete_crl`

Description: This procedure deletes a CRL from the database.

Arguments: conn - Pointer returned from DBI_OPEN, hashIssuerP - hash value of issuer DN, issuerP - issuer DN in DNS format, cpath_seq - sequence number of the certification path associated with the object, crl - CRL passed in globally

Return Value: Same as for `int delete_cert`

Function:`int delete_aa`

Description: This procedure deletes an AA from the database.

Arguments: conn - Pointer returned from DBI_OPEN, aaHashP - hash value of AA, cpath_seq - sequence number of the certification path associated with the object, aa - AA passed in globally

Return Value: Same as for `int delete_cert`

Function:`int delete_cpath`

Description: This procedure deletes a certification path from the database.

Arguments: conn - Pointer returned from DBI_OPEN, cpathHashP - hash value of certification path, cpath - certification path passed in globally

Return Value: Same as for `int delete_cert`

Function:`int update_cpath_count`

Description: This procedure updates the `count` field for a certification path in the database.

Arguments: conn - Pointer returned from DBI_OPEN, seqP - `seq` of certification path entry, +N (for increment) or -N (for decrement) **Return Value**: `count` of certification path entry

B.2.2 Postgres Functions

Function:`int DBI_writeCertEntry`

Description: This procedure writes a cert to the db_cert table in the database.

Arguments: conn - Pointer returned from DBI_OPEN, DB_cert - database record

to be written to the database, hashIssuerP - hash value of issuer DN, issuerP - issuer DN in DNS format, snP - serial number, cpath_seqP - sequence number of the certification path associated with the object

Return Value: 0 for SUCCESS or FAILURE for failure

Function:`int DBI_writeCrlEntry`

Description: This procedure writes a CRL to the db_crl table in the database.

Arguments: conn - Pointer returned from DBI_OPEN, DB_crl - database record to be written to the database, hashIssuerP - hash value of issuer DN, issuerP - issuer DN in DNS format, cpath_seqP - sequence number of the certification path associated with the object

Return Value: 0 for SUCCESS or FAILURE for failure

Function:`int DBI_writeAAEntry`

Description: This procedure writes an AA to the db_aa table in the database.

Arguments: conn - Pointer returned from DBI_OPEN, DB_aa - database record to be written to the database, aaHashP - hash value of AA, cpath_seqP - sequence number of the certification path associated with the object

Return Value: 0 for SUCCESS or FAILURE for failure

Function:`int DBI_writeCpathEntry`

Description: This procedure writes a certification path with a particular `count` value to the db_cpath table in the database.

Arguments: conn - Pointer returned from DBI_OPEN, DB_cpath - database record to be written to the database, cpathHashP - hash value of cpath, countP - count value

Return Value: `seq` of cpath entry for SUCCESS or FAILURE for failure

Function:`int DBI_readCpathEntry`

Description: This procedure reads a certification path entry from the db_cpath ta-

ble in the database.

Arguments: conn - Pointer returned from DBI_OPEN, DB_cpath - database record to be written to the database, cpathHash - hash value of cpath

Return Value: `seq` and `count` of cpath entry for SUCCESS or FAILURE for failure

Function:`int DBI_deleteEntry`

Description: This procedure deletes an entry in the database specified by the conn and rec_type argument. The entry to be deleted in the databse is specified by the seq argument. It should identify one entry from the database to be deleted.

Arguments: conn - Pointer returned from DBI_OPEN, where_buffer - specifies the qualification clause, rec_type - a cert, CRL, AA, cpath record (valid values are CERT_REC, CRL_REC, AA_REC, CPATH_REC)

Return Value: SUCCESS or FAILURE

Function:`int DBI_rec_exists`

Description: This procedure determines if a entry exists for the key specified.

Arguments: conn - Pointer returned from DBI_OPEN, where_buffer - specifies the qualification clause, rec_type - a cert, CRL, AA, cpath record (valid values are CERT_REC, CRL_REC, AA_REC, CPATH_REC)

Return Value: SUCCESS - record exists, FAILURE - on errors, NO_MATCH - no record found

B.3 Other Database Functions

Functions: `int make_dn_dns_cert` and `int make_dn_dns_crl`

Description: `make_dn_dns_cert` and `make_dn_dns_crl` is called by `write_cert_to_database` and `write_crl_to_database` respectively to parse the ufn notation of an issuer DN to get it in the dns format to write to DB_cert.iDns and DB_crl.iDns field.

Arguments: dnP - issuer DN in ufn notation

Return Value: 0 for SUCCESS or appropriate negative value for failure

Outputs: DB_cert.iDns and DB_crl.iDns - DN string to put into database record

Function:`int check_dup_cert`

Description: This routine is called to determine if a entry already exists in the certificate table for the keys specified.

Arguments: conn - Pointer returned from DBI_OPEN, hashIssuer - hash value of issuer DN and sn_buf - serial number (key of the record), cert - Certificate passed in globally

Return Value: DBI_ENTRYEXISTS if object is found in the db, NO_MATCH if no record found; FAILURE for other errors

Outputs: `seq` of the record

Function:`int check_dup_crl`

Description: This routine is called to determine if a entry already exists in the CRL table for the keys specified.

Arguments: conn - Pointer returned from DBI_OPEN, hashIssuer - hash value of issuer DN (key of the record), crl - CRL passed in globally

Return Value: Same as for `int check_dup_cert`

Outputs: `seq` of the record

Function:`int check_dup_aa`

Description: This routine is called to determine if a entry already exists in the AA table for the keys specified.

Arguments: conn - Pointer returned from DBI_OPEN, aaHash - hash value of aa (key of the record), aa - AA passed in globally

Return Value: Same as for `int check_dup_cert`

Outputs: `seq` of the record

Function:`int check_dup_cpath`

Description: This routine is called to determine if a entry already exists in the certification path table for the keys specified.

Arguments: conn - Pointer returned from DBI_OPEN, cpathHash - hash value of cpath, key of the record, cpath - certification path passed in globally

Return Value: Same as for `int check_dup_cert`

Outputs: `seq` of the record, `count` of record