

SECURITY SYSTEM FOR DNS USING CRYPTOGRAPHY

ABSTRACT

The mapping or binding of IP addresses to host names became a major problem in the rapidly growing Internet and the higher level binding effort went through different stages of development up to the currently used Domain Name System (DNS).

The DNS Security is designed to provide security by combining the concept of both the Digital Signature and Asymmetric key (Public key) Cryptography. Here the Public key is send instead of Private key. The DNS security uses Message Digest Algorithm to compress the Message(text file) and PRNG(Pseudo Random Number Generator) Algorithm for generating Public and Private key. The message combines with the Private key to form a Signature using DSA Algorithm, which is send along with the Public key.

The receiver uses the Public key and DSA Algorithm to form a Signature. If this Signature matches with the Signature of the message received, the message is Decrypted and read else discarded.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF TABLES	viii
	LIST OF FIGURES	viii
1.	INTRODUCTION	1
	1.1 Scope of the Project	1
	1.2 Problem Statement	2
2.	OVERVIEW OF DNS	3
	2.1. Fundamentals of DNS	4
	2.1.1 The Domain Name Space	5
	2.1.2. DNS Components	7
	2.2. DNS Transactions	11
	2.3 PROPOSED SYSTEM	12

3.	DNSSEC	13
	3.1. DNSSEC Objectives	14
	3.2. Performance Considerations	15
	3.3. DNSSEC Scope	15
	3.3.1. Key Distribution	15
	3.3.2. Data Origin Authentication	16
	3.3.3. DNS Transaction and Request Authentication	16
	3.4. DNSSEC Resource Records	18
	3.4.1. KEY RR	18
	3.4.2. SIG RR	20
	3.4.3. NXT RR	21
	3.5 Public Key Retrieval	26
4.	DEVELOPMENT ENVIRONMENT	28
	4.1 Hardware Environment	28
	4.2 Software Environment	28

5.	TESTING	29
	5.1 Testing Principles	29
	5.2 System Testing Requirements	30
	5.3 Phases Of Testing	30
	5.3.1 Unit Testing	30
	5.3.2 Integration Testing	31
	5.3.3 System Testing	31
	5.3.4 Performance Testing	32
	5.3.5 Validation Testing	32
6.	IMPLEMENTATION	33
	6.1 Source	33
	6.2 Domain-1	40
	6.3 Server	46
	6.4 Domain-2	50
	6.5 Destination	57
7.	SCREEN CAPTURES	64
8.	CONCLUSION	68

APPENDICES

REFERENCES

LIST OF TABLES

Table 1. Common DNS Resource Records	9
--------------------------------------	---

LIST OF FIGURES

Figure 1. Domain Name Space example	6
Figure 2. Example of inverse domains and the Domain Name Space	7
Figure 3. These three records are grouped into a RRSet.	10
Figure 4. Example of a DNS cross check that fails	17
<i>Figure 5. DNSSEC query & response messages</i>	
Figure 6. Authentication	64
Figure 7. Choice	65
Figure 8. Message	66
Figure 9. File	67

1. INTRODUCTION

1.1 SCOPE OF THE PROJECT

The Domain Name System(DNS) has become a critical operational part of the Internet Infrastructure, yet it has no strong security mechanisms to assure Data Integrity or Authentication. Extensions to the DNS are described that provide these services to security aware resolves are applications through the use of Cryptographic Digital Signatures. These Digital Signatures are included zones as resource records.

The extensions also provide for the storage of Authenticated Public keys in the DNS. This storage of keys can support general Public key distribution services as well as DNS security. These stored keys enables security aware resolvers to learn the authenticating key of zones, in addition to those for which they are initially configured. Keys associated with DNS names can be retrieved to support other protocols. In addition, the security extensions provide for the Authentication of DNS protocol transactions.

The DNS Security is designed to provide security by combining the concept of both the Digital Signature and Asymmetric key (Public key) Cryptography. Here the Public key is send instead of Private key. The DNS security uses Message Digest Algorithm to compress the Message(text file) and PRNG(Pseudo Random Number Generator) Algorithm for generating Public and Private key. The message combines with the Private key to form a Signature using DSA Algorithm, which is send along with the Public key.

The receiver uses the Public key and DSA Algorithm to form a Signature. If this Signature matches with the Signature of the message received, the message is Decrypted and read else discarded.

1.2 PROBLEM STATEMENT

Authenticity is based on the identity of some entity. This entity has to prove that it is genuine. In many Network applications the identity of participating entities is simply determined by their names or addresses. High level applications use mainly names for authentication purposes, because address lists are much harder to create, understand, and maintain than name lists.

Assuming an entity wants to spoof the identity of some other entity, it is enough to change the mapping between its low level address and its high level name. It means that an attacker can fake the name of someone by modifying the association of his address from his own name to the name he wants to impersonate. Once an attacker has done that, an authenticator can no longer distinguish between the true and fake entity.

2. Overview of the DNS

To connect to a system that supports IP, the host initiating the connection must know in advance the IP address of the remote system. An IP address is a 32-bit number that represents the location of the system on a network. The 32-bit address is separated into four octets and each octet is typically represented by a decimal number. The four decimal numbers are separated from each other by a dot character ("."). Even though four decimal numbers may be easier to remember than thirty-two 1's and 0's, as with phone numbers, there is a practical limit as to how many IP addresses a person can remember without the need for some sort of directory assistance. The directory essentially assigns host names to IP addresses.

The Stanford Research Institute's Network Information Center (SRI-NIC) became the responsible authority for maintaining unique host names for the Internet. The SRI-NIC maintained a single file, called `hosts.txt`, and sites would continuously update SRI-NIC with their host name to IP address mappings to add to, delete from, or change in the file. The problem was that as the Internet grew rapidly, so did the file causing it to become increasingly difficult to manage. Moreover, the host names needed to be unique throughout the worldwide Internet. With the growing size of the Internet it became more and more impractical to guarantee the uniqueness of a host name. The need for such things as a hierarchical naming structure and distributed management of host names paved the way for the creation of a new networking protocol that was flexible enough for use on a global scale [ALIU].

What evolved from this is an Internet distributed database that maps the names of computer systems to their respective numerical IP network address(es). This Internet lookup facility is the DNS. Important to the

concept of the distributed database is delegation of authority. No longer is one single organization responsible for host name to IP address mappings, but rather those sites that are responsible for maintaining host names for their organization(s) can now regain that control.

2.1. Fundamentals of DNS

The DNS not only supports host name to network address resolution, known as forward resolution, but it also supports network address to host name resolution, known as inverse resolution. Due to its ability to map human memorable system names into computer network numerical addresses, its distributed nature, and its robustness, the DNS has evolved into a critical component of the Internet. Without it, the only way to reach other computers on the Internet is to use the numerical network address. Using IP addresses to connect to remote computer systems is not a very user-friendly representation of a system's location on the Internet and thus the DNS is heavily relied upon to retrieve an IP address by just referencing a computer system's Fully Qualified Domain Name (FQDN). A FQDN is basically a DNS host name and it represents where to resolve this host name within the DNS hierarchy.

RELATED WORKS

2.1.1. The Domain Name Space

The DNS is a hierarchical tree structure whose root node is known as the root domain. A label in a DNS name directly corresponds with a node in the DNS tree structure. A label is an alphanumeric string that uniquely identifies that node from its brothers. Labels are connected together with a dot notation, ".", and a DNS name containing multiple labels represents its path along the tree to the root. Labels are written from left to right. Only one zero length label is allowed and is reserved for the root of the tree. This is commonly referred to as the root zone. Due to the root label being zero length, all FQDNs end in a dot [RFC 1034].

As a tree is traversed in an ascending manner (i.e., from the leaf nodes to the root), the nodes become increasingly less specific (i.e., the leftmost label is most specific and the right most label is least specific). Typically in an FQDN, the left most label is the host name, while the next label to the right is the local domain to which the host belongs. The local domain can be a subdomain of another domain. The name of the parent domain is then the next label to the right of the subdomain (i.e., local domain) name label, and so on, till the root of the tree is

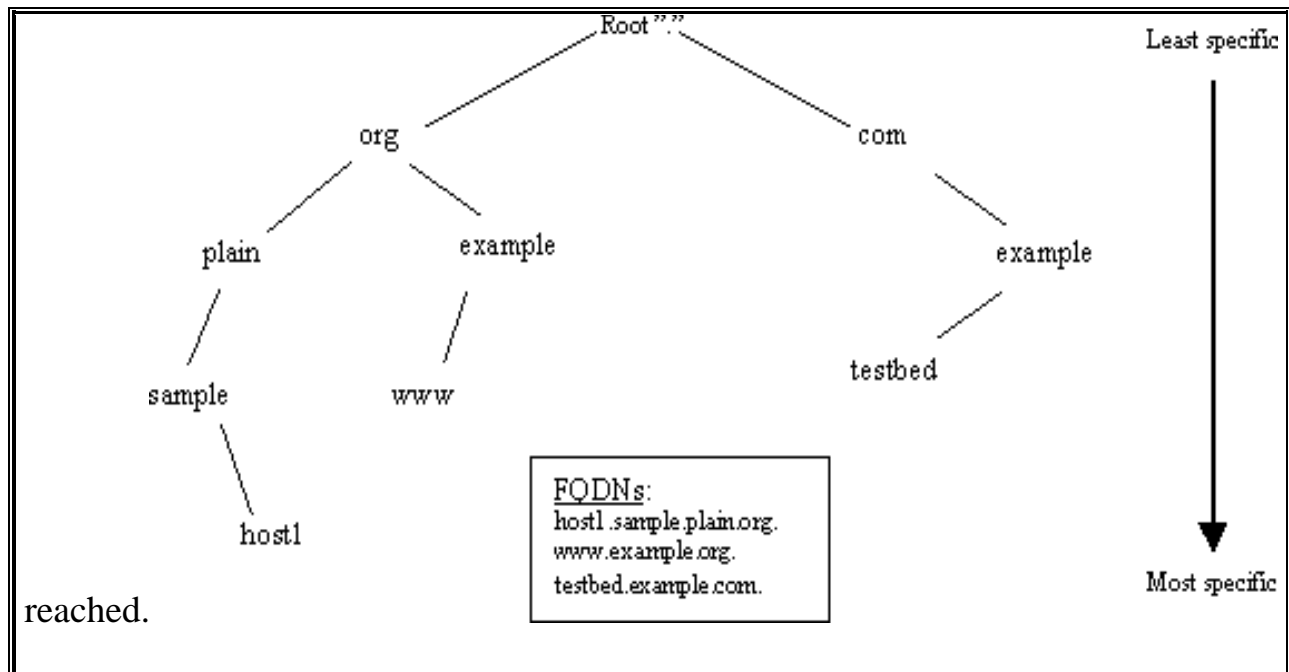


Figure 1. Domain Name Space example

When the DNS is used to map an IP address back into a host name (i.e., inverse resolution), the DNS makes use of the same notion of labels from left to right (i.e., most specific to least specific) when writing the IP address. This is in contrast to the typical representation of an IP address whose dotted decimal notation from left to right is least specific to most specific. To handle this, IP addresses in the DNS are typically represented in reverse order. IP addresses fall under a special DNS top level domain (TLD), known as the in-addr.arpa domain. By doing this, using IP addresses to find DNS host names are handled just like DNS host name lookups to find IP addresses.

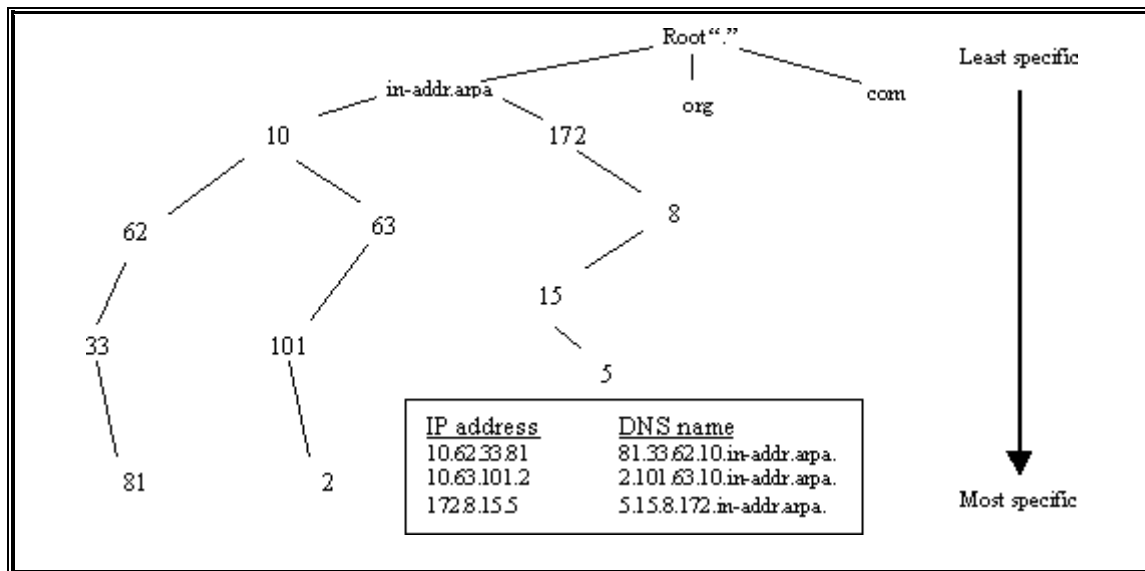


Figure 2. Example of inverse domains and the Domain Name Space

2.1.2. DNS Components

The DNS has three major components, the database, the server, and the client [RFC 1034]. The database is a distributed database and is comprised of the Domain Name Space, which is essentially the DNS tree, and the Resource Records (RRs) that define the domain names within the Domain Name Space. The server is commonly referred to as a name server. Name servers are typically responsible for managing some portion of the Domain Name Space and for assisting clients in finding information within the DNS tree. Name servers are authoritative for the domains in which they are responsible. They can also serve as a delegation point to identify other name servers that have authority over subdomains within a given domain.

The RR data found on the name server that makes up a domain is commonly referred to as zone information. Thus, name servers have zones of authority. A single zone can either be a forward zone (i.e., zone information that

pertains to a given domain) or an inverse zone (i.e., zone information that maps IP addresses into DNS host names). DNS allows more than one name server per zone, but only one name server can be the primary server for the zone. Primary servers are where the actual changes to the data for a zone take place. All the other name servers for a zone basically maintain copies of the primary server's database for the zone. These servers are commonly referred to as secondary servers.

A DNS RR has 6 fields: NAME, TYPE, CLASS, TTL, RD Length, and RDATA. The NAME field holds the DNS name, also referred to as the owner name, to which the RR belongs. The TYPE field is the TYPE of RR. This field is necessary because it is not uncommon for a DNS name to have more than one type of RR. The more common types of RR are found in

Table 1.

RECORD TYPE	DESCRIPTION	USAGE
A	An address record	Maps FQDN into an IP address
PTR	A pointer record	Maps an IP address into FQDN

NS	A name server record	Denotes a name server for a zone
SOA	A Start of Authority record	Specifies many attributes concerning the zone, such as the name of the domain (forward or inverse), administrative contact, the serial number of the zone, refresh interval, retry interval, etc.
CNAME	A canonical name record	Defines an alias name and maps it to the absolute (canonical) name
MX	A Mail Exchanger record	Used to redirect email for a given domain or host to another host

Table 1. Common DNS Resource Records

The CLASS in this case is "IN" which stands for Internet. Other classes exist but are omitted for brevity. The TTL is the time, in seconds, that a name server can cache a RR. A zero time to live means that a server is not to cache the RR. RD Length is the RDATA field's length in octets. The RDATA field is the resource data field and is uniquely defined for each TYPE of RR, but in general it can be thought of as the value into which the entity specified in the NAME field maps. The NAME field can be thought of as the subject

of a query, although this is not always the case, and the answer is the data contained in the RDATA field (even though the entire RR is returned in a DNS response) [RFC 1035].

RRs are grouped into resources records sets (RRSets). RRSets contain 0 or more RRs [RFC 2136] that have the same DNS name, class, and type, but the data (i.e., RDATA) is different. If the name, class, type, and data are the same for two or more records then duplicate records exist for the same DNS name. Name servers should suppress duplicate records [RFC 2181]. The Figure 3 shows an example of a RRSet.

example.com.
example.com.
example.com.
IN
IN
IN
NS
NS
NS
ns1.example.com.
ns2.example.com.
ns.plain.org.

Figure 3. These three records are grouped into a RRSet.

The client component of the DNS typically contains software routines, known as functions, which are responsible for requesting information from the Domain Name Space on behalf of an application. These functions are bundled together into a software library that is commonly referred to as the resolver library. For this reason, clients are often called resolvers. The resolver library functions are responsible for sending a query to a name server requesting information concerning a DNS name and returning the answer to the query back to the requestor.

2.2. DNS Transactions

DNS transactions occur continuously across the Internet. The two most common transactions are DNS zone transfers and DNS queries/responses. A DNS zone transfer occurs when the secondary server updates its copy of a zone for which it is authoritative. The secondary server makes use of information it has on the zone, namely the serial number, and checks to see if the primary server has a more recent version. If it does, the secondary server retrieves a new copy of the zone.

A DNS query is answered by a DNS response. Resolvers use a finite list of name servers, usually not more than three, to determine where to send queries. If the first name server in the list is available to answer the query, then the others in the list are never consulted. If it is unavailable, each name server in the list is consulted until one is found that can return an answer to the query. The name server that receives a query from a client can act on behalf of the client to resolve the query. Then the name server can query other name servers one at a time, with each server consulted being

presumably closer to the answer. The name server that has the answer sends a response back to the original name server, which then can cache the response and send the answer back to the client. Once an answer is cached, a DNS server can use the cached information when responding to subsequent queries for the same DNS information. Caching makes the DNS more efficient, especially when under heavy load. This efficiency gain has its tradeoffs; the most notable is in security.

2.3 PROPOSED SYSTEM

Taking the above prevailing system into consideration the best solution is using Pseudo Random Number Generator for generating KeyPair in a quick and more secured manner. We use MD5 (or) SHA-1 for producing MessageDigest and Compressing the message. Signature is created using Private Key and MessageDigest which is transmitted along with the Public Key. The transfer of the packets from each System to System is shown using Graphical User Interface (GUI). Each time the System get the message, it verifies the IPAddress of the sender and if no match is found it discards it. For verification, the Destination System generates Signature using PublicKey and DSA Algorithm and verifies it with received one. If it matches it Decrypts otherwise it discards.

The Following functions avoids the pitfalls of the existing system.

- Fast and efficient work
- Ease of access to system
- Manual effort is reduced

3. DNSSEC

In 1994, the IETF formed a working group to provide security extensions to the DNS protocol in response to the security issues surrounding the DNS. These extensions are commonly referred to as DNSSEC extensions. These security enhancements to the protocol are designed to be interoperable with non-security aware implementations of DNS. The IETF achieved this by using the RR construct in the DNS that was purposely designed to be extensible. The WG defined a new set of RRs to hold the security information that provides strong security to DNS zones wishing to implement DNSSEC. These new RR types are used in conjunction with existing types of RRs. This allows answers to queries for DNS security information belonging to a zone that is protected by DNSSEC to be supported through non-security aware DNS servers.

In order to gain widespread acceptance, the IETF DNSSEC WG acknowledged that DNSSEC must provide backwards compatibility and must have the ability to co-exist with non-secure DNS implementations. This allows for sites to migrate to DNSSEC when ready and allows less complexity when upgrading. This also means that client side software that are not DNSSEC aware can still correctly process RRsets received from a DNSSEC server [CHAR].

In March of 1997, the Internet Architecture Board (IAB) met to discuss the development of an Internet security architecture. This meeting identified existing security mechanisms and those that are under development, but have not yet become standards, that can play a part in the security

architecture. They also identified areas in which adequate protection using existing security tools could not be achieved. The results of this workshop include the identification of core security requirements for the Internet security architecture. Among those security protocols identified as core is DNSSEC. The protection that DNSSEC provides against injection of false cache information is crucial to the core security requirements of the Internet [RFC 2316].

3.1. DNSSEC Objectives

A fundamental principle of the DNS is that it is a public service. It requires correct and consistent responses to queries, but the data is considered public data. As such, the need for authentication and integrity exists, but not for access control and confidentiality. Thus, the objectives of DNSSEC are to provide authentication and integrity to the DNS. Authentication and integrity of information held within DNS zones is provided through the use of cryptographic signatures generated through the use of public key technology. Security aware servers, resolvers, and applications can then take advantage of this technology to assure that the information obtained from a security aware DNS server is authentic and has not been altered.

Although the DNSSEC WG chose not to provide confidentiality to DNS transactions, they did not eliminate the ability to provide support for confidentiality. Other applications outside of the DNS may choose to use the public keys contained within the DNS to provide confidentiality. Thus the DNS, in essence, can become a worldwide public key distribution mechanism. Issues such as cryptographic export are not, and may never be,

solved worldwide; however, the DNS provides mechanisms to have multiple keys, each from a different cryptographic algorithm for a given DNS name, as a means to help alleviate this problem.

3.2. Performance Considerations

Performance issues are a concern for the security extensions to the DNS protocol and several aspects in the design of DNSSEC are targeted to avoid the overhead associated with processing the extensions. For instance, formulating another query that asks for the signature belonging to the RRSet just retrieved is not necessarily the most efficient way to retrieve a signature for the RRSet. This additional query is avoided whenever possible by allowing information retrieved from secured zones to be accompanied by the signature(s) and key(s) that validate the information.

3.3. DNSSEC Scope

The scope of the security extensions to the DNS can be summarized into three services: key distribution, data origin authentication, and transaction and request authentication.

3.3.1. Key Distribution

The key distribution service not only allows for the retrieval of the public key of a DNS name to verify the authenticity of the DNS zone data, but it also provides a mechanism through which any key associated with a DNS name can be used for purposes other than DNS. The public key distribution service supports several different types of keys and several different types of key algorithms.

3.3.2. Data Origin Authentication

Data origin authentication is the crux of the design of DNSSEC. It mitigates such threats as cache poisoning and zone data compromise on a DNS server. The RRSets within a zone are cryptographically signed thereby giving a high level of assuredness to resolvers and servers that the data just received can be trusted.

DNSSEC makes use of digital signature technology to sign DNS RRSets. The digital signature contains the encrypted hash of the RRSets. The hash is a cryptographic checksum of the data contained in the RRSets. The hash is signed (i.e., digitally encrypted) using a private key usually belonging to the originator of the information, known as the signer or the signing authority. The recipient of the RRSets can then check the digital signature against the data in the RRSets just received. The recipient does this by first decrypting the digital signature using the public key of the signer to obtain the original hash of the data. Then the recipient computes its own hash on the RRSets data using the same cryptographic checksum algorithm, and compares the results of the hash found in the digital signature against the hash just computed. If the two hash values match, the data has integrity and the origin of the data is authentic [CHAR].

3.3.3. DNS Transaction and Request Authentication

DNS transaction and request authentication provides the ability to authenticate DNS requests and DNS message headers. This guarantees that the answer is in response to the original query and that the response came from the server for which the query was intended. Providing the assurance for both is done in one step. Part of the information, returned in a response to

a query from a security aware server, is a signature. This signature is produced from the concatenation of the query and the response. This allows a security aware resolver to perform any necessary verification concerning the transaction.

Another use of transaction and request authentication is for DNS Dynamic Updates. Without DNSSEC, DNS Dynamic Update does not provide a mechanism that prohibits any system with access to a DNS authoritative server from updating zone information. In order to provide security for such modifications, Secure DNS Dynamic Update incorporates DNSSEC to provide strong authentication for systems allowed to dynamically manipulate DNS zone information on the primary server [RFC 2137].

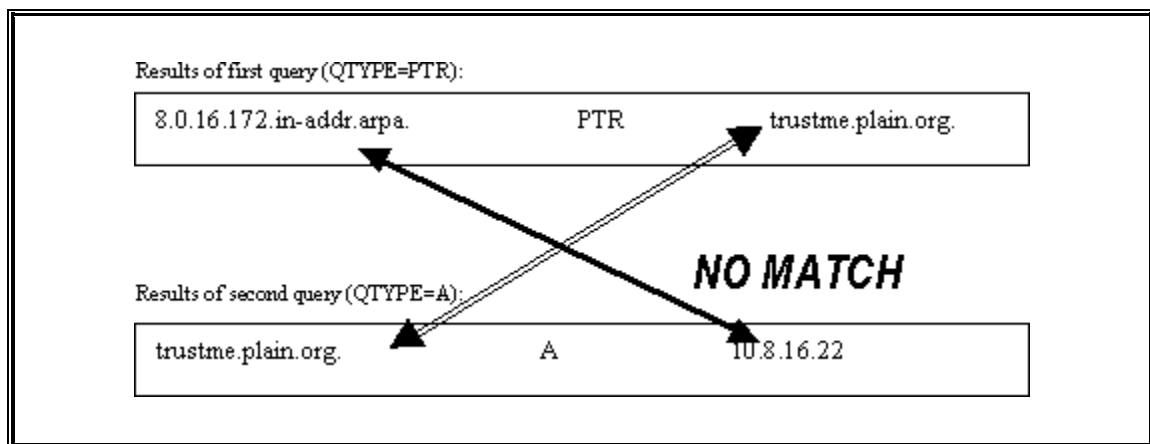


Figure 4. Example of a DNS cross check that fails

3.4. DNSSEC Resource Records

The IETF created several new DNS RRs to support the security capabilities provided by DNSSEC extensions. The RRs pertinent to the DNS are the KEY RR, SIG RR, and the NXT RR. DNSSEC utilizes the KEY RR for

storing cryptographic public keys, one public key per KEY RR. It is the KEY RR that is used for verification of a DNS RRSset's signature. The signature for a RRSset is stored in the SIG RR. The signature is used to prove the authenticity and integrity of the information contained in the RRSset. The NXT RR is the nonexistent RR and is used to cryptographically assert the nonexistence of a RRSset. Another RR exists, known as the CERT RR, that does not bring any additional security functions to the DNS, but is provided so that public key certificates can be kept within the DNS for use in applications outside of the DNS [RFC 2538]. In much the same way an application wishing to communicate with a remote IP host generates an A query to resolve the host name, a security application wishing to perform encryption with another entity generates a CERT query to retrieve the entity's public key certificate. For further explanation on KEY, SIG, and NXT RRs and their RDATA fields and flags not contained herein, please reference RFC 2535 and related documents.

3.4.1. KEY RR

The key for a DNS name is held in a KEY RR. Any type of query for a DNS name, found in a secured zone, results in a response that contains the answer to the query. The KEY RR associated with the DNS name can accompany this response. The resolver that generated the query can then validate the data using the KEY RR without having to send another query for the Key RR. This minimizes the number of queries needed for any given DNS name found in a secured zone.

DNSSEC utilizes the KEY RR for storing cryptographic public keys; however, this is not a public key certificate. Instead, the CERT RR is used to

store public key certificates. The key found in the RDATA section of the KEY RR belongs to the DNS name first listed in the KEY RR (i.e., the owner name). The owner name can represent a zone, a host, a user, et al.

The Key RR contains information denoting the security characteristics of the key and its allowed usage for the given owner name. It provides security information such as the public key, algorithm type, protocol type, and flags that specify such things as to whether or not the DNS name has a public key. The public key algorithm determines the actual format of the public key found in the RDATA section of the KEY RR. Several key algorithms are already supported and are defined in RFC 2535 as RSA/MD5, Diffie-Hellman, and Digital Signature Algorithm (DSA), and the elliptic curve algorithm. Only DSA support is mandatory. Another field is known as the protocol octet. It indicates for which protocol the public key is valid. Some already assigned protocols are TLS, email, DNSSEC, and IPsec. Since both the public key algorithm field and the protocol octet is an 8-bit field, theoretically up to 255 different algorithms and 255 different protocols can be used in conjunction with the public key.

Two bits out of the sixteen bits used for setting various flags are known as the type bits. All four combinations of the type bits indicate how the KEY RR can be used. They are confidentiality, authentication, confidentiality and authentication, or none. The latter indicates a key does not exist for the DNS name. In this way, one can cryptographically assert that the given owner name does not possess a key even though it is in a secure zone. Another two bits are used to identify three kinds of entities for which this key belongs, such as user, zone, or something that is not a zone. Indicating a host with

these flags is actually done by using the flags to indicate that the DNS name is not a zone. Thus a host is implied rather than specified by the flags.

3.4.2. SIG RR

A signature is held in another resource record type known as a SIG RR. The SIG RR provides authentication for a RRSset and the signature's validity time. In a secure zone, a RRSset has one or more SIG RR associated with it. The situation of having more than one SIG RR for a given RRSset may arise when more than one cryptographic algorithm is being used for signing the RRSset. Some sites may choose to do this for issues such as cryptographic export restrictions.

A number of fields are also found in the RDATA section of a SIG RR. The signature field holds the signature belonging to a specific RRSset. To indicate the RR type of the RRSset (i.e., NS, PTR, MX, etc.), a "type covered" field is used. In order to verify the signature, a resolver or server must know the signer's name. This is specified in the signer's field. The SIG RR has an algorithm field identical to that in the KEY RR. Since signatures have expiration times, as do individual RRs, the SIG RR has several time fields. This is further discussed later in this paper, [see "Security Aware Servers"].

Except for the SIG RRs used for transaction and request authentication and for the SIG RRs which are specifically the target of a query, security aware servers try to include in the response the SIG RRs needed to authenticate the RRSset. Thus, a resolver may still receive an answer for a RRSset belonging to a secure zone that does not have the SIG RR. This situation can typically occur when a size limitation is exceeded due to the SIG RR or when the response comes from a non-security aware server. Under these

circumstances, the security aware resolver is required to form another query specifically requesting any missing SIG RRs needed to complete the verification process.

3.4.3. NXT RR

The DNS provides the ability to cache negative responses. A negative response means that a corresponding RRSet does not exist for the query. DNSSEC provides signatures for these nonexistent RRSets so that their nonexistence in a zone can be authenticated. It does this through the use of the NXT RR. NXT RRs are used to indicate a range of DNS names that are unavailable or a range of RR types that are unavailable for an existing DNS name.

Two possibilities exist for nonexistent DNS names. One is that the DNS name itself does not have any RRs; it simply does not exist. The other is that the DNS name does exist (i.e., has at least one type of RR), but the RR type in the query for that name does not exist. To handle proof of nonexistence of a DNS name, all the records in a zone are sorted in a manner that is similar in some ways to alphabetical order. The technique used is known as canonical order and is defined in RFC 2535. Then when a query is received for a nonexistent name, a NXT RR is sent back containing the DNS name of the next DNS RRSet occurring "alphabetically", or rather canonically, after the name in the query. To handle proof of nonexistence of a RR type for an existing DNS name, a NXT record is sent back with the DNS name and the RR types that the name does in fact have. Whenever SIG RRs are generated for a zone, all NXT RRs for a zone should be generated.

Security aware DNS servers are the source of all security-related information within the DNS. Any given primary DNS server has three main functions: manage authoritative zone information, manage the caching of DNS information, and respond to client queries. A primary DNS server that is security aware has added responsibilities to each of these functions. Authoritative zone information management for a security aware server includes the addition of SIG, KEY, and NXT RRs in a zone's master database file. The SIG RRs are generated for the RRSets belonging to a zone. The private key used to generate the SIG belongs to the zone itself. Since private keys of servers are more than likely found on-line, it is possible that these keys could be compromised. The zone's private key, in contrast, is kept off-line for most purposes, so its compromise is less likely and the validity of the data is more assured. The zone's private key is retrieved periodically to re-sign all the records found within the zone. Once the new SIG RRs are generated they are included with the rest of the information in the zone's master file. NXT RRs also should be generated on the server and placed into a zone's master file whenever SIG RRs are generated.

On-line signing also takes place at the server. For transaction and request authentication for DNS queries, the server formulating the reply must use its private key for signing, rather than the zone key since it is kept off-line. Another instance in which a zone key is not used for signing is for transaction and request authentication for dynamic updates. The private key of the host formulating the request must be used. Because DNS queries and dynamic update requests can occur quite frequently, the signer's private keys must be maintained on-line. The protection of these on-line private keys is of

utmost importance; however, the means in which they are protected is beyond the scope of this paper. RFC 2541 discusses the operational considerations of KEY and SIG RR.

To perform caching, a security aware server must properly manage the caching of all security related RRs. The additional responsibility in caching of a security aware server begins with the maintaining of four cache states. One state, which has a corresponding state in a non-security aware server, is "Bad". In a non-security aware server, when a bad response is received in that the information contained is in some way corrupt, a non-security aware server throws away the response message without caching it (and typically logs the event). In much the same way, a security aware server can throw away a bad response, but in this case, a bad response means that the SIG RR verifications failed on the data. Even though the RRSets in the response may look legitimate, the failing of the data checks with the corresponding signature is a fatal condition.

The three other states are Insecure, Authenticated, and Pending. Insecure means that there isn't any available data to use to check the authenticity of the RRSets. It does not mean the data is bad, just that it cannot be authenticated. This commonly occurs for RRSets from non-secured zones. Authenticated means the RRSets cached has been fully validated through the use of the SIG RRs and KEY RRs. Pending means the cached data is still in the process of being checked.

Another server responsibility with caching is when to expire a cached RRSets. Once a RRSets is cached, a count down to zero from the original TTL is started and maintained for the cached record. Once zero is reached, the

RRSet is removed from the cache. For security aware servers, this has changed a little. The TTL cannot be the only time kept to determine when a cached RRSet is expired. Two new times are now used in addition to the TTL and these ultimately determine when to expire the RRSet from the cache. The new times are used to determine when the signature's validity time period for the authenticated RRSet expires, rather than just when the RRSet should be expired. These new times are kept in the SIG RR and are known as the signature inception time and the signature expiration time. For security aware clients and server this information is far more important on which to base expiration since it is cryptographically asserted. Although the signature expiration time seems have a correlation to the TTL, due to backward compatibility issues, the TTL field cannot be eliminated.

TTL aging is still incorporated for expiring authenticated RRsets. If the TTL expires prior to the signature expiration time, the TTL is decremented as normal and the RRSet is expired when the TTL hits zero. If the signature expiration time occurs prior to when the TTL expires, the TTL is adjusted to the signature expiration time and then the normal countdown of the TTL proceeds.

Responding to client queries now involves answering queries from both security aware and security unaware resolvers. When a non security aware resolver generates a query and sends it to a security aware server for information contained in a secured zone, the security aware servers can respond with either Authenticated or Insecure data. A security aware server can only send Pending data when the checking disabled (CD) flag is set. The security aware server knows not to send Pending data because a resolver not participating in DNSSEC never sets the CD flag in a DNS query. Since

sending Insecure data is the same as DNS without DNSSEC, the security unaware resolver processes the response message as usual. As far as receiving Authenticated data, the security unaware resolver basically ignores the additional security information and goes about processing the response as usual.

When queries are originating from security aware resolvers, it is strongly encouraged that the resolver set the CD flag. With the CD flag set in the query, security aware servers can send the Pending data. Sending Pending data accomplishes two things. It minimizes the response time freeing up server resources for handling queries and it allows a resolver to implement its policies on Pending data, independent of servers. If the answer to the query is already Authenticated data on the server, the server sets the authentic data flag (AD) to indicate to the resolver that the necessary checks have already been performed. In this way the resolver does not need to do any security verification checks.

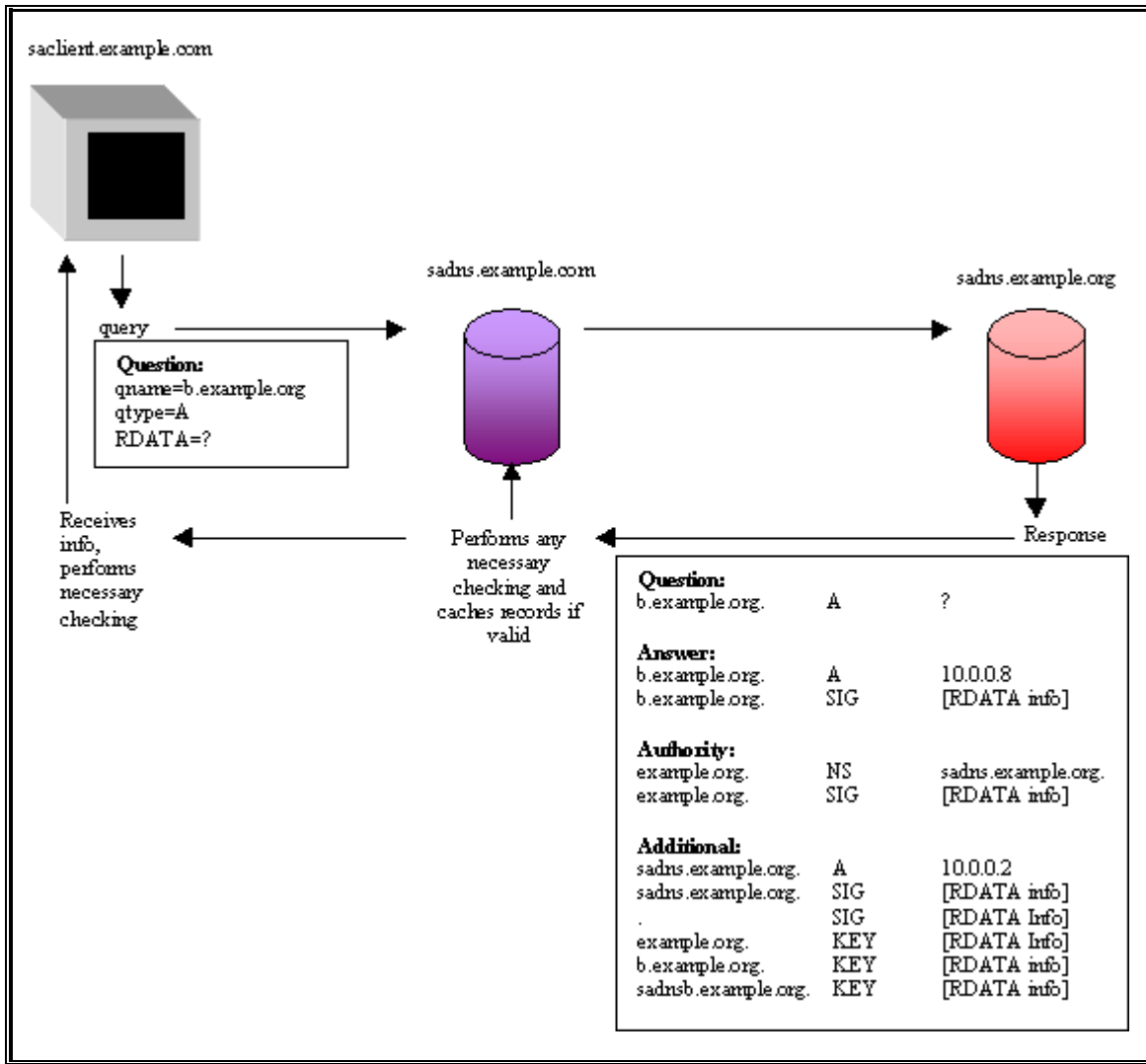


Figure 5. DNSSEC query & response messages

3.5 Public Key Retrieval

Resolvers can obtain public keys of zones in one of two ways. Resolvers can utilize the DNS to query for the public key or they can be statically configured with the key. Regardless of the method used, problems exist with both. In the case where keys are obtained through the DNS, the issue of trusting the key arises. In order to trust the retrieved key, it must be signed and this signature must be reliable. Providing the assurance that the

signature on the key is reliable means that the public key of the signing authority must also be obtained, be signed, and be found reliable and so on. The solution to ending this recursive chain of events is to configure the resolver with the public key that authenticates the signed keys below it. In other words, a trusted zone key can be used as a starting point for verifying all keys found below it. A likely set of trusted public keys with which a secure zone can be statically configured are those of the root zone.

The static configuration of a resolver with the public keys from many different zones has one advantage in that the compromise of one of the zone's private key does not result in the compromise of the keys for all the other zones. The disadvantage of statically configuring each resolver with keys for many different zones is that it does not scale well. If one key for one zone must change, then all the resolvers must be configured to reflect this change.

4. DEVELOPMENT ENVIRONMENT

4.1 HARDWARE ENVIRONMENT

The minimum configuration required to run this project are:

1. Main processor : Pentium III (or) IV
2. RAM : 128MB
3. Hard Disk : 4.2GB
4. Clock Speed : 550 MHZ
5. System Bus Speed : 400 MHz
6. Cache RAM : 256 KB

4.2 SOFTWARE ENVIRONMENT

Language : JDK1.3 (or) Higher.

Front End Design : Swings

Operating System : Windows

5. TESTING

Software Testing is a process of executing program within the intent of finding an error. Software testing is a critical element of software quality assurance and represents The ultimate review of system specification, design, coding. Testing is last chance to uncover the error defects in the software and facilities delivery of quality system,

5.1 TESTING PRINCIPLES

The basic principles for effective software testing are follows:

- A good test case is the one that has a high probability of finding an as –yet undiscovered error.
- A successful test is one that uncovers an as-yet undiscovered Error.
- All tests should net race able to the customer requirements.
- Tests should be planned long before testing begins.
- Testing should begin in the small “and progress towards testing “in the large.
- Exhaustive testing is not possible.

5.2 SYSTEM TESTING REQUIREMENTS

Software testing is not an activity to take up when the product is ready. An effective testing begins with a proper plan forms the user requirement stage itself. Software testability is the case with which a computer program is tested. Metrics can be used to measure the testability of a product.

5.3 PHASES OF THE TESTING

Several testing strategies and lead to the following generic characteristics:

- Testing begins then unit level and works “outward” toward the integration of the entire system.
- Different testing techniques are appropriate at different points of software development cycle.

5.3.1 UNIT TESTING

System security refers to the technical innovations and procedure applied to the hardware and operating system to protect against deliberate or accidental damage. Data security refers to the protection of data from loss, disclosure, modification and distraction. Privacy defines the rights of the users or organization to determine what information they willing to share with others and protect the information to minimize the possible invasion of privacy. To achieve all the above objectives.

5.3.2 INTERGRATION TESTING

System integrity refers to the proper functioning of hardware and software, Appropriate physical security and safety against external threats like wiretapping. Data integrity makes sure that data do not differ from their original form.

5.3.3 SYSTEM TESTING

After the Integration testing gets over the system as a whole is tested for validation. Here the testing is done by a complete tour of all the modules in a sequence.

In case of further development of the system in the future, the programmer has to know the logic involved. Documents to a programmer are like Road map to a traveler on the move.

Having the above facts in mind, a lot of care was taken in documenting at every stage of the project. By reading these documents the logic's involved in the programs will be crystal clear to the Programmer, in future. These documents will be of extensive use for debugging if any bugs are detected in the future.

5.3.4 PERFORMANCE TESTING

The system is very much user friendly and has a good user interface. This has been tested. Every user who needs to access this system is

given an user Id and password and no one else can access. This too has been tested. It has been tested whether the loading of the screens of the application is fast and the migration from one form to another took less time. The time taken for this had been calculated. The application is designed in such a way that it occupies less memory space; the database is also designed

in such a way that it avoids duplication of records -i.e. the database avoids redundancy in all possible ways. Redundancy in storing the same data multiple times leads to several problems. Due to this storage place is also wasted. The files that represent the same data may be inconsistent. All these problems are looked after and rectified for efficient execution of the application.

5.3.5 VALIDATION TESTING

Here in this validation testing, all the values entered in each and every module are tested for correctness and validation as it has been entered before updating to the back end system.

6. IMPLEMENTATION

SOURCE

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.util.zip.*;
import java.security.*;
import java.security.SecureRandom.*;
import java.security.spec.*;
import javax.swing.*;

public class SwingMes extends JFrame implements ActionListener
{
    JTabbedPane tp ;
    JButton send,browse,send1;
    JTextField tf1;
    JTextArea ta,ta1;
    JScrollPane sp,sp1;
    String g1=" ";
    byte str[];
    byte realSig[];
    String y="";
    String st="";
    Client ct;

    public SwingMes(Client ct)
    {
        super("Transaction");
        this.ct=ct;
        setSize(500,500);
        Container c = getContentPane();
        JPanel jp = new JPanel(new FlowLayout());
        JLabel jl = new JLabel("Enter the Text: ");
        ta = new JTextArea(20,40);
        sp = new JScrollPane(ta);
        send= new JButton("Send");
        jp.add(jl);
```

```

jp.add(sp);
jp.add(send);
jp.setVisible(true);
JPanel jp1 = new JPanel(new FlowLayout());
JLabel jl1 = new JLabel("Enter the File Name ",JLabel.LEFT);
tf1 = new JTextField(15);
browse = new JButton("Browse");
send1= new JButton("Send");
send.addActionListener(this);
browse.addActionListener(this);
send1.addActionListener(this);
jp1.add(jl1);
jp1.add(tf1);
jp1.add(browse);
jp1.add(sp1);
jp1.add(send1);
jp1.setVisible(true);
tp = new JTabbedPane();
tp.addTab("Message",null,jp,"Enter the Message here");
tp.addTab("File",null,jp1,"Enter the File Name here");
c.add(tp);
setDefaultCloseOperation(EXIT_ON_CLOSE);
setVisible(true);
setResizable(false);
}

```

```

/*      MESSAGE DIGEST ALOGORITHM      */

```

```

private byte[] content;
public byte[] digestValue(byte[] in_text1)
{
    byte[] in_text=in_text1;
    content=in_text;
    MessageDigest mg1=null;

    try
    {
        mg1=MessageDigest.getInstance("SHA1");
    }
}

```

```

        catch (Exception e)
        {
            System.out.println(e);
        }
        mg1.update(content);
        byte[] digest1=mg1.digest();
        System.out.println("Message Digest:"+digest1);
        return digest1;
    }
    public void keys()
    {
        try
        {
            KeyPairGenerator keygen =
KeyPairGenerator.getInstance("DSA","SUN");

            /* cryptographically strong pseudo-random number generator
(PRNG) */
            SecureRandom random =
SecureRandom.getInstance("SHA1PRNG","SUN");
            keygen.initialize(1024,random);
            KeyPair keypair = keygen.generateKeyPair();
            PrivateKey privatekey = keypair.getPrivate();
            PublicKey publickey = keypair.getPublic();
            FileOutputStream pubkeyfos = new
FileOutputStream("public.txt");
            pubkeyfos.write(publickey.getEncoded());

            System.out.println("PublicKey"+publickey.getEncoded());
            pubkeyfos.close();
            FileOutputStream prikeyfos = new
FileOutputStream("private.txt");
            prikeyfos.write(privatekey.getEncoded());
            System.out.println("PrivateKey:"+privatekey);
            prikeyfos.close();
            JOptionPane.showMessageDialog(null, "keys are
created", "Keys", JOptionPane.INFORMATION_MESSAGE);
        }
        catch(Exception e)
        {

```



```

        System.out.println(e);
    }
}
public void met2(String dis2)
{
    System.out.println("FileName:"+dis2);
    try
    {
        int i;
        String g=" ";
        char x[]=new char[50];
        char d,s;
        String s1;
        InputStream in = new FileInputStream(dis2);
        str=new byte[in.available()];
        in.read(str, 0, str.length);
        System.out.println(new String (str));
        ta1.setText(new String (str));
        FileOutputStream fos=new
FileOutputStream("input.txt");
        fos.write(str);
        int len=str.length;
        System.out.println("StringLength:"+len);
        int a[]=new int[len];
        System.out.print("HASH-CODE:");
        for(i=0;i<len;i++)
        {
            int aa= str[i];
            System.out.print("\t"+ aa);
            a[i]=aa;
        }
        System.out.print("HEX-CODE:");
        for(i=0;i<len;i++)
        {

            g=" ";
            b[i]=a[i]%16;
            a[i]=a[i]/16;
            g1=g1+g.trim();
        }
    }
}

```

```

        str=g1.getBytes();
        digestValue(g1.getBytes());
    }
    catch(IOException e)
    {
        System.out.println(e);
    }
    keys();
    genSig("input.txt");
    String inputValue = JOptionPane.showInputDialog("Enter the
System Name for Domain1");
    try
    {
        Socket s1 = new
Socket(InetAddress.getByName(inputValue),7878);
        PrintStream ps = new
PrintStream(s1.getOutputStream());
        ps.println(ct.SenderName.getText());
        System.out.println("SenderName
:"+ct.SenderName.getText());
        ps.println(ct.SenderPassword.getText());
        ps.println(ct.ReceiverName.getText());
        System.out.println("ReceiverName
:"+ct.ReceiverName.getText());
        System.out.println("Encrypted Data:"+ new String(str));
        ps.flush();
        s1.close();
    }
    catch(Exception ex)
    {
        System.out.println(ex);
    }
}
public void genSig(String fname)

{
    try
    {
        byte[] md;
        String args=fname;

```

```

        FileInputStream fin=new FileInputStream(args);
        byte[] in_text=new byte[fin.available()];
        fin.read(in_text);
        fin.close();
        md=digestValue(in_text);
        FileInputStream fins=new FileInputStream("private.txt");
        byte[] enc_priv=new byte[fins.available()];
        fins.read(enc_priv);
        fins.close();
        Signature
dsa=Signature.getInstance("SHA1withDSA","SUN");
        PKCS8EncodedKeySpec privKeySpec=new
PKCS8EncodedKeySpec(enc_priv);
        KeyFactory
keyFactory=KeyFactory.getInstance("DSA","SUN");
        PrivateKey
priv=(PrivateKey)keyFactory.generatePrivate(privKeySpec);
        FileOutputStream sigfos = new
FileOutputStream("realSign.txt");
        sigfos.write(realSig);
        sigfos.close();
        JOptionPane.showMessageDialog(null, "Signature are
Generated","Signature", JOptionPane.ERROR_MESSAGE);
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
public void actionPerformed(ActionEvent e)
{
    if(e.getSource() == browse)
    {
        JFileChooser fc = new JFileChooser();
        int option = fc.showOpenDialog(SwingMes.this);
        if(option == JFileChooser.APPROVE_OPTION)
        {

tf1.setText(fc.getSelectedFile().getAbsolutePath());
        }
    }
}

```

```

    }
    else if(e.getSource() == send)
    {
        if (ta.getText() == null)
        else
        {
            int i;
            int b[]=new int[5000];
            int a[]=new int[5000];
            String g=" ";
            char x[]=new char[50];
            char d,s;
            String h=ta.getText();
            System.out.println("Message Entered:"+h);
            int len=h.length();
            System.out.println("StringLength:"+len);
            for(i=0;i<len;i++)
            {
                int aa= h.charAt(i);
                System.out.print("\t"+ aa);
                a[i]=aa;
            }
            for(i=0;i<len;i++)
            {
                g=" ";
                b[i]=a[i]%16;
                a[i]=a[i]/16;
                g1=g1+g.trim();
            }
            System.out.println(g1);
            str=g1.getBytes();
            System.out.println(str);
            st=new String(str);
            digestValue(g1.getBytes());
            keys();
            genSig("input.txt");
            try
            {
                Socket s1 = new
Socket(InetAddress.getByName(inputValue),7878);

```

```

        PrintStream ps = new
PrintStream(s1.getOutputStream());
        ps.println(ct.SenderName.getText());
        ps.println(new String(str));
        ps.println("public.txt");
        ps.println("realSign.txt");
        ps.flush();
        BufferedReader br=new
BufferedReader(new InputStreamReader(s1.getInputStream()));
        {
            JOptionPane.showMessageDialog(null,"Valid
Password","Password",JOptionPane.ERROR_MESSAGE);
        }
    }
}
}

```

DOMAIN-1

```

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Domn12 implements ActionListener
{
    static String str90,str91,str92,str93,str94,str95;
    static String str1,str2,str3,str4,str5,str15,str21,str35;
    static String output;
    static String wq =new String();
    static JFrame jf;
    static JTextField t1,t2,t3,t4;
    static JLabel l1,l2,l3,l4,l5;
    static JTextArea ta;
    static JScrollPane js;
    static JButton Send,Exit;
    static boolean succ;

```

```

Domn12()
{
    String
inf="com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
    try
    {
        UIManager.setLookAndFeel(inf);
    }
    catch(Exception e){ }
    jf =new JFrame("Domain1");
    l = new JLabel("Domain-1 Contains the Sub-Domains");
    t1 = new JTextField(20);
    t2 = new JTextField(20);
    ta = new JTextArea();
    js = new JScrollPane(ta);
    Send= new JButton("Send");
    Exit= new JButton("Exit");
    p = new JPanel();
    p.add(l);
    p.add(l1);
    p.add(l2);
    p.add(l3);
    p.add(t1);
    p.add(t2);
    p.add(js);
    p.add(Send);
    p.add(Exit);
    Send.addActionListener(this);
    Exit.addActionListener(this);
    p.setLayout(null);
    l.setBounds(40,40,350,25);
    l1.setBounds(40,70,120,25);
    t1.setBounds(180,70,120,25);
    l2.setBounds(40,100,120,25);
    l3.setBounds(40,130,120,25);
    l5.setBounds(40,190,120,25);
    js.setBounds(40,220,300,200);
    Send.setBounds(100,430,75,25);
    Exit.setBounds(200,430,75,25);
    jf.setContentPane(p);

```

```

        jf.setSize(400,500);
    }
    public static void main(String args[])throws Exception
    {
        Domn12 d1 = new Domn12();
        succ=false;
    }
    public static boolean ServSoc()
    {
        boolean res1=false;
        try
        {
            ServerSocket ss = new ServerSocket(7878);
            Socket s1 = ss.accept();
            System.out.println("Domain-1 Contains the Sub-
Domains are:");
            BufferedReader ps2 = new BufferedReader(new
InputStreamReader(s1.getInputStream()));
            str1= ps2.readLine();
            System.out.println("SenderName    : "+str1);
            t1.setText(str1);
            str2 = ps2.readLine();
            System.out.println("SenderPassword : "+str2);
            str3 = ps2.readLine();
            System.out.println("ReceiverName  : "+str3);
            t2.setText(str3);
            str4 = ps2.readLine();
            System.out.println("Encrypted Data : "+str4);
            try
            {
                int len=str4.length();
                int k=16;
            }

            str5 = ps2.readLine();
            str5=str5.trim();
            System.out.println("Public Key    : "+str5);
            t3.setText(str5);
            t4.setText(str15);
            str35 = str1.substring(str1.lastIndexOf("."));
            System.out.println("SenderDomain  : "+str35);

```

```

        PrintStream ps7 = new
PrintStream(s1.getOutputStream());
        ps7.println(output);
        s1.close();
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
public void actionPerformed(ActionEvent e12)
{
    if(e12.getSource()==Send)
    {
        if (succ ==true)
        {
            Soc();
        }
    }
    else
    {
        System.out.println("Connection Refused");
    }
}
public static void Soc()
{
    try
    {
        Socket s2 = new
Socket(InetAddress.getByName(input),9999);
        PrintStream ps = new
PrintStream(s2.getOutputStream());
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
public static boolean pass(String temp)throws Exception
{

```



```

        int Result;
        String Response;
        String Rname;
        Rname=temp;
        FileReader fr = new FileReader("password.txt");
        BufferedReader br = new BufferedReader(fr);
        String s="";
        String str[];
        while((s=br.readLine())!=null)
        {
            if(Rname.equalsIgnoreCase(str[0]))
            {
                Result=1;
                output=str[1];
            }
            else
            {
                output="password is incorrect";
                System.out.println(output);
            }
        }

        JOptionPane.showMessageDialog(null,"Invalid
        Password","Password",JOptionPane.ERROR_MESSAGE);

    }

}

}

public static void sen1(String temp)throws Exception
{
    int Result;
    String output;
    Result=0;
    Rname=temp;
    FileReader fr = new FileReader("send1.txt");
    BufferedReader br = new BufferedReader(fr);
    String s="";
    String str[];
}

public static void ServSoc1()
{

```

```

        try
        {
            ServerSocket ss = new ServerSocket(8898);
            Socket s11 = ss.accept();
            BufferedReader ps2 = new BufferedReader(new
InputStreamReader(s11.getInputStream()));
            str93 = ps2.readLine();
            str93=str93.trim();
            System.out.println("ReceiverName      : "+str93);
            str95 = ps2.readLine();
            str95=str95.trim();
            System.out.println("RealSign          : "+str95);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
    public static void Soc1()
    {
        try
        {
            String input1 =
JOptionPane.showInputDialog("Enter the System Name for Receiver");
            System.out.println("str92:"+str92);
            ps.println(str93);
            System.out.println("str95:"+str95);
            soc.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
    public static void Dom2(String temp)throws Exception
    {
        int Result;
        Rname="";
        output="";
        Rname=temp;
    }

```

```

        try
        {
            System.out.println("Domain-2
Contains the Sub-Domains are:");
            FileReader fr = new
FileReader("Send1.txt");

            String str[];
            while((s=br.readLine())!=null)
            {

                str=s.split(" ");

            }
        }
        catch(Exception w)
        {
            System.out.println(w);
        }
    }
}

```

SERVER

```

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class DnsServer12 implements ActionListener
{
    static String str2,str21,str31,str41,str51,str71,str72;
    static JFrame jf;
    static JTextField t1,t2,t3,t4,t5,t6;
    static JLabel l,l1,l2,l3,l4,l5,l6,l7;
    static JPanel p;
    static Socket s2;
    static String subs1;
    static String sub="";
    DnsServer12()

```

```

        {
            String
inf="com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
            try
            {
                UIManager.setLookAndFeel(inf);
            }
            catch(Exception e){ }
            jf=new JFrame("DNSServer");
            l = new JLabel("DNS-Server Contains the Domains");
            l1 = new JLabel("SenderName:");
            l6 = new JLabel("SenderDomain:");
            l7 = new JLabel("ReceiverDomain:");
            t1 = new JTextField(20);
            t2 = new JTextField(20);
            ta = new JTextArea();
            l.setBounds(40,40,350,25);
            l1.setBounds(40,70,120,25);
            l5.setBounds(40,250,120,25);
            js.setBounds(40,270,300,180);
        }
        public void actionPerformed(ActionEvent e12)
        {
            if(Exit== e12.getSource())
            {
                System.exit(0);
            }
        }

        public static void ServSoc1()
        {
            try
            {
                ServerSocket ss = new ServerSocket(9999);
                Socket s1 = ss.accept();
                t1.setText(str31);
                str71 = ps2.readLine();
                str71=str71.trim();
            }
        }
    }
}

```

```

        System.out.println("Public Key   :"+str71);
        t3.setText(str71);
        t4.setText(str72);
        System.out.println("Server Checking the List of
Domains");
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}

public static void Soc1()
{
    try
    {
        String inputValue =
JOptionPane.showInputDialog("Enter the System Name for Domain2");
        Socket s2 = new
Socket(InetAddress.getByName(inputValue),9989);
        PrintStream ps = new
PrintStream(s2.getOutputStream());
        ps.println(str2);
        ps.println(str21);
        ps.flush();
        s2.close();
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}

public static void Dom1(String temp)throws Exception
{
    int Result;
    String output;
    FileReader fr = new FileReader("file1.txt");
    BufferedReader br = new BufferedReader(fr);
    String s="";
    String str[];

```

```

    }

    public static void Dom2(String temp)throws Exception
    {
        int Result;
        String output;
        String Response;
        String s="";
        String str[];
    }
    public static void ServSoc11()
    {
        try
        {
            ServerSocket ss = new ServerSocket(9988);
            Socket s11 = ss.accept();
            BufferedReader ps2=new BufferedReader(new
InputStreamReader(s11.getInputStream()));
            Dom11(str80);
            Dom21(str81);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
    public static void Soc11()
    {
        try
        {
            Socket s21 = new
Socket(InetAddress.getByName(input),8898);
            PrintStream ps3 = new
PrintStream(s21.getOutputStream());
            ps3.println(str80);
            ps3.println(str81);
            ps3.flush();
            s21.close();
        }
        catch(Exception e)

```

```

        {
            System.out.println(e);
        }
    }
    public static void Dom11(String temp)throws Exception
    {
        int Result;
        String output;
        String Response;
        String s="";
        String str[];
    }
    public static void Dom21(String temp)throws Exception
    {
        int Result;
        String output;
        String Response;
        String Rname;
        Result=0;
        String str[];
    }
}

```

DOMAIN-2

```

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

```

class Domn22 implements ActionListener

```

{
    static String str21,str2,str31,str41,str51,str71,str72;
    static JFrame jf;
    static JTextField t1,t2,t3,t4;
    static JPanel p;
    static Socket s2;
    static String subs1,subs;

```

```

Domn22()
{
    String
inf="com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
    try
    {
        UIManager.setLookAndFeel(inf);
    }
    catch(Exception e){ }
    jf =new JFrame("Domain2");
    l = new JLabel("Domain-2 Contains the Sub-Domains");
    t2 = new JTextField(20);
    js = new JScrollPane(ta);
    Send= new JButton("Send");
    Exit= new JButton("Exit");
    Font f = new Font("Bold Italic",Font.PLAIN,20);
    l.setFont(f);
    p = new JPanel();
    p.add(l);
    p.add(l1);
    p.add(l2);
    p.add(js);
    p.add(Send);
    p.add(Exit);
    l.setBounds(40,40,350,25);
    l1.setBounds(40,70,120,25);
    jf.setContentPane(p);
    jf.setSize(400,500);
}
public void actionPerformed(ActionEvent e12)
{
    if(e12.getSource()==Send)
    {
        Soc1();
    }
    else
    {
        System.out.println("Connection Refused");
    }
}

```



```

public static void ServSoc1()
{
    try
    {
        ServerSocket ss = new ServerSocket(9989);
        Socket s1 = ss.accept();
        BufferedReader ps2 = new BufferedReader(new
InputStreamReader(s1.getInputStream()));

        str21 = ps2.readLine();
        str21=str21.trim();
        System.out.println("SenderDomainName  : "+str21);
        str2 = ps2.readLine();
        str2=str2.trim();
        System.out.println("ReceiverDomainName : "+str2);
        t1.setText(str31);
        str41 = ps2.readLine();
        str41=str41.trim();
        System.out.println("ReceiverName      : "+str41);
        t2.setText(str41);
        {
            int len=str51.length();
            int len1=len/16;
            int len2=len%16;
            int k=16;
            int c=0;
            ta.setText("Source
:"+str31+"\n"+sub+"\n"+"Destination :"+str41);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        t3.setText(str71);
        t4.setText(str72);
        Dom2(str41);
        s1.close();
    }
    catch(Exception e)
    {

```

```

        System.out.println(e);
    }
}

public static void Soc1()
{
    try
    {
        String inputValue =
JOptionPane.showInputDialog("Enter the System Name for Receiver");
        Socket soc = new
Socket(InetAddress.getByName(inputValue),5555);
        PrintStream ps = new
PrintStream(soc.getOutputStream());
        ps.println(str31);
        ps.println(str41);
        soc.close();
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}

public static void Dom2(String temp)throws Exception
{

    int Result;
    String output;
    String Response;
    String Rname;
    try
    {
        System.out.println("Domain-2 Contains the
Sub-Domains are:");
        FileReader fr = new
FileReader("Send1.txt");
        BufferedReader br=new BufferedReader(fr);
        String s="";
        String str[];

```

```

        }
        catch(Exception w)
        {
            System.out.println(w);
        }
    }
    public static void ServSoc()
    {
        try
        {
            ServerSocket ss = new ServerSocket(8787);
            Socket s1 = ss.accept();
            System.out.println("Domain-1 Contains the Sub-
Domains are:");
            BufferedReader ps2 = new BufferedReader(new
InputStreamReader(s1.getInputStream()));
            str61= ps2.readLine();
            str61=str61.trim();
            System.out.println("SenderName   : "+str61);
            str62 = ps2.readLine();
            str62=str62.trim();
            System.out.println("SenderPassword : "+str62);
            s1.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
    public static void Soc()
    {
        try
        {
            String input =
JOptionPane.showInputDialog("Enter the System Name for DNS-Server");
            Socket s2 = new
Socket(InetAddress.getByName(input),9988);
            PrintStream ps = new
PrintStream(s2.getOutputStream());
            ps.println(str66);

```

```

        ps.println(str67);
        ps.println(str61);
        ps.flush();
        s2.close();
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
public static void pass(String temp)throws Exception
{
    int Result;
    String output;
    String Response;
    String Rname;
    Rname=temp;
    FileReader fr = new FileReader("password.txt");
    BufferedReader br = new BufferedReader(fr);

    while((s=br.readLine())!=null)
    {
        str=s.split(" ");
        if(Rname.equalsIgnoreCase(str[0]))
        {
            Result=1;
            output=str[1];
        }
    }
}

public static void sen1(String temp)throws Exception
{
    int Result;
    String output;
    String Response;
    String Rname;
    Result=0;
    Rname=temp;

```

```

        FileReader fr = new FileReader("send1.txt");
        BufferedReader br = new BufferedReader(fr);
        String s="";
        String str[];

        while((s=br.readLine())!=null)
        {

            str=s.split(" ");

            if(Rname.equalsIgnoreCase(str[0]))
            {

                Result=1;
                output=str[1];
                System.out.println("SenderIP : "+output);
                break;

            }

            else
            {

                output="Server Protected";
                System.out.println("SenderIP : "+output);

            }

        }

    }
}

```

DESTINATION

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.util.zip.*;

```

```

import java.security.*;
import java.security.SecureRandom.*;
import java.security.spec.*;
import javax.swing.*;

public class SwingRMes extends JFrame implements ActionListener
{
    JTabbedPane tp ;
    JLabel jl,jl1,jl2,jL;
    JButton jb,jb1,jb2;
    JTextField tf1,tf2,tfd,tfd1;
    JTextArea tf,ta1;
    JScrollPane sp,sp1;
    static String g1=" ";
    static String n,n1,n2;
    static String y="";
    static String str6,str61,str2,str12,str13;
    static String t,st="",st1="",vj="";

    public SwingRMes()
    {
        super("Reception");
        String
inf="com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
        try
        {
            UIManager.setLookAndFeel(inf);
        }
        catch(Exception e){ }
        setSize(400,450);
        Container c = getContentPane();
        JPanel jp = new JPanel();
        jp.setLayout(null);
        jl = new JLabel("Sender Name ",JLabel.LEFT);
        jL = new JLabel("Receiver Name ",JLabel.LEFT);
        tfd = new JTextField(15);
        tfd1 = new JTextField(15);
        tf = new JTextArea(20,40);
        jb= new JButton("OPEN");
        jp.add(jl);

```

```

        jp.add(tfd);
        jp.add(jL);
        jl.setBounds(25,10,150,25);
        tfd.setBounds(150,10,150,25);
        jL.setBounds(25,40,150,25);
        tfd1.setBounds(150,40,150,25);
        JPanel jp1 = new JPanel();
        jp1.setLayout(null);
        jl1 = new JLabel("Sender Name ",JLabel.LEFT);
        jl2 = new JLabel("Receiver Name ",JLabel.LEFT);
        tf1 = new JTextField(15);
        sp1 = new JScrollPane(ta1);
        jb2= new JButton("Retrive");
        jl1.setBounds(25,10,150,25);
        tf1.setBounds(150,10,150,25);
        jl2.setBounds(25,40,150,25);
        tf2.setBounds(150,40,150,25);
        jp1.add(jl1);
        jp1.add(tf1);
        jp1.add(jl2);
        jp1.setVisible(true);
        tp = new JTabbedPane();
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public static void ServSoc()
    {
        try
        {
            ServerSocket ss = new ServerSocket(5555);
            Socket s11 = ss.accept();
            System.out.println(" Server Started..... ");
            BufferedReader ps2=new BufferedReader(new
InputStreamReader(s11.getInputStream()));
            str2 = ps2.readLine();
            str2=str2.trim();
            System.out.println("File Name   : " +str2);
            char a[]=new char[50000];
            char res[]=new char[9];
            char s1[]=new char[5];
            String w =new String();

```

```

String r =new String();
System.out.println("Verify Signature are
Generated");

        try
        {
            verifySig(str12);
        }
        catch(Exception e)
        {

            System.out.println(e);
        }
        InputStream in3 = new
FileInputStream(str12);
        FileOutputStream out2=new
FileOutputStream("Sign1.txt");
        byte[] str3=new byte[512];
        while(in3.read(str3)!=-1)
        {
            String r1=new String(str3);
            out2.write(str3);
        }

        /* Converting Hexcode into Bits */

        for(i=0;i<len;i++)
        {
            char aa=
((str2).charAt(i));

            a[i]=aa;
            if(a[i]=='A')
            {
                st="1010";
                st1=st;
            }
            else if(a[i]=='B')
            {
                st="1011";
                st1=st;
            }
        }
    }
}

```



```

}
else if(a[i]=='C')
{
    st="1100";
    st1=st;
}
else if(a[i]=='D')
{
    st="1101";
    st1=st;
}
else if(a[i]=='E')
{
    st="1110";
    st1=st;
}
else if(a[i]=='F')
{
    st="1111";
    st1=st;
}
else if(a[i]=='0')
{
    st="0000";
    st1=st;
}
else if(a[i]=='1')
{
    st="0001";
    st1=st;
}
else if(a[i]=='2')
{
    st="0010";
    st1=st;
}
else if(a[i]=='3')
{
    st="0011";
    st1=st;
}

```

```

    }
    else if(a[i]=='4')
    {
        st="0100";
        st1=st;
    }
    else if(a[i]=='5')
    {
        st="0101";
        st1=st;
    }
    else if(a[i]=='6')
    {
        st="0110";
        st1=st;
    }
    else if(a[i]=='7')
    {
        st="0111";
        st1=st;
    }
    else if(a[i]=='8')
    {
        st="1000";
    }
    else if(a[i]=='9')
    {
        st="1001";
    }
    w=w+st;
}

```

char

w_ch[]=w.toCharArray();

```

String x=w.toString();
System.out.print(x);

```

System.out.println("Length:"+len);

public static boolean verifySig(String fname) throws Exception

```

        {

            byte[] md;
            byte[] sign;
            byte[] realSig;
            String args=fname;
            FileInputStream fin=new FileInputStream(args);
            byte[] in_text=new byte[fin.available()];
            fin.read(in_text);
            fin.close();
            //SHA sha=new SHA(in_text);
            md=digestValue(in_text);

            FileInputStream finPublic=new
FileInputStream(str12);
            byte[] enc_pub=new byte[finPublic.available()];
            finPublic.read(enc_pub);
            finPublic.close();

            FileInputStream sigfis=new
FileInputStream(str13);
            sign=new byte[sigfis.available()];
            sigfis.read(sign);

            Signature dsa1 =
Signature.getInstance("SHA1withDSA","SUN");
            PublicKey
pub=keyFactoryPub.generatePublic(pubKeySpec);
            dsa1.initVerify(pub);
            dsa1.update(md);
        }
        public void actionPerformed(ActionEvent e)
        {
            if(e.getSource() == jb2)
            {
                tf1.setText(str6);
                tf2.setText(str61);
                ta1.setText(vj);
            }
        }
    }
}

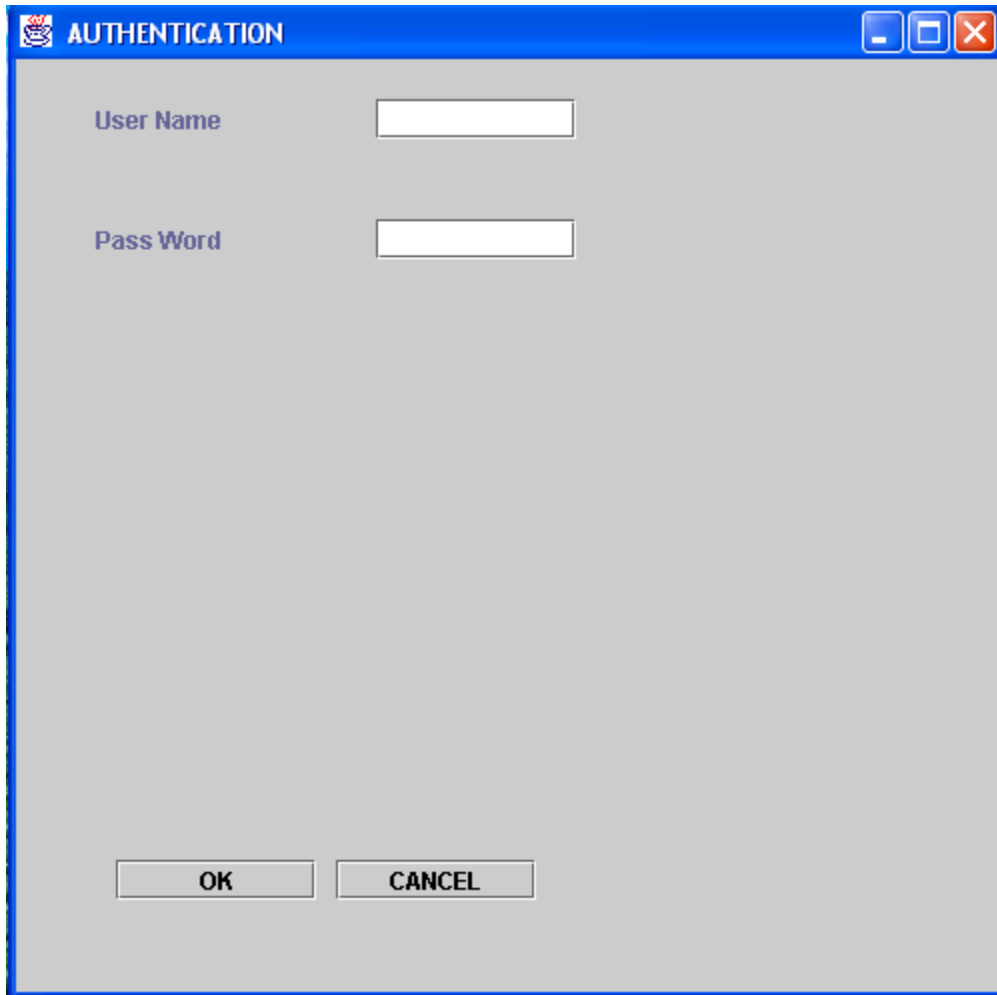
```

```

    }
    else if(e.getSource() == jb)
    {
        JOptionPane.showMessageDialog(null, "Signature are
Verified","Verify Sign", JOptionPane.INFORMATION_MESSAGE);
        tfd1.setText(str61);
        tf.setText(vj);
    }
}

```

7. Screen Captures



The image shows a standard Windows-style dialog box titled "AUTHENTICATION". The title bar is blue and contains a small icon on the left and three control buttons (minimize, maximize, and close) on the right. The main area of the dialog has a light gray background. It contains two labels, "User Name" and "Pass Word", both in a dark blue font. Each label is followed by a white rectangular text input field. At the bottom of the dialog, there are two buttons: "OK" and "CANCEL", both with a light gray background and black text.

Figure 6. Authentication

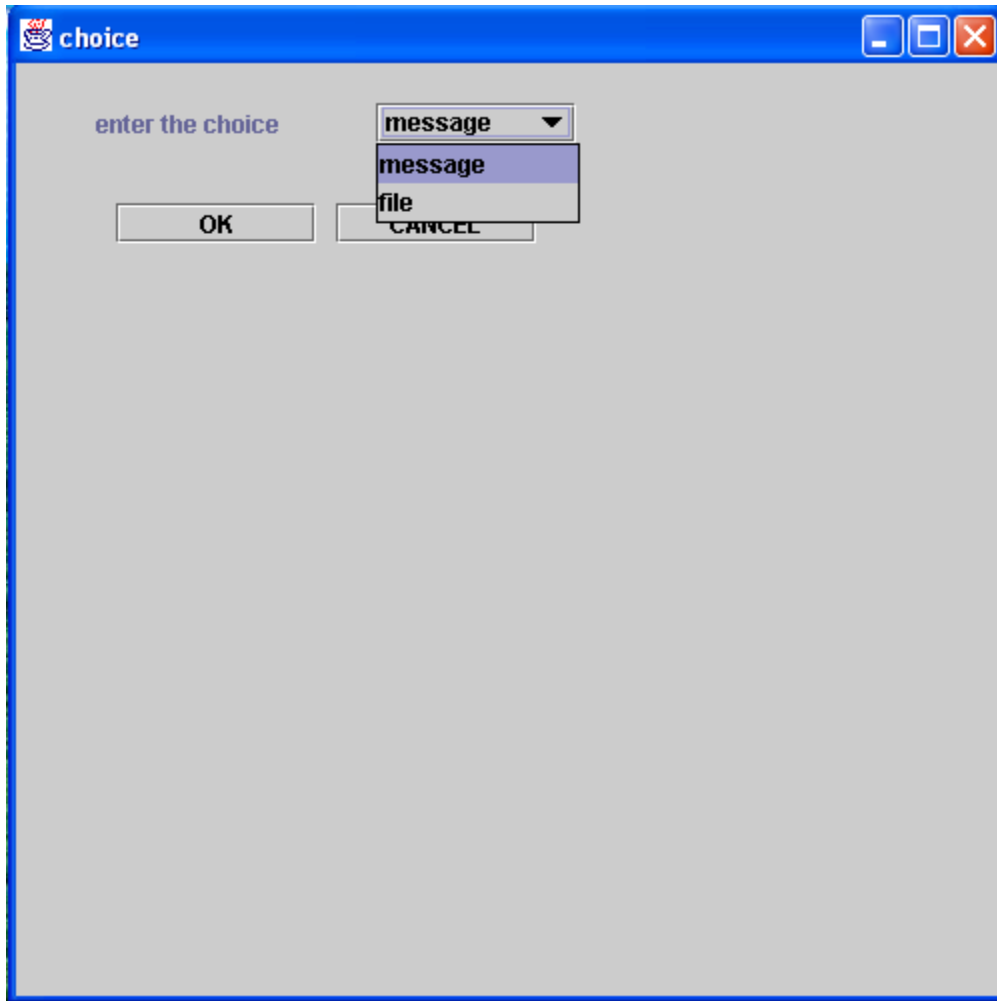


Figure 7. Choice

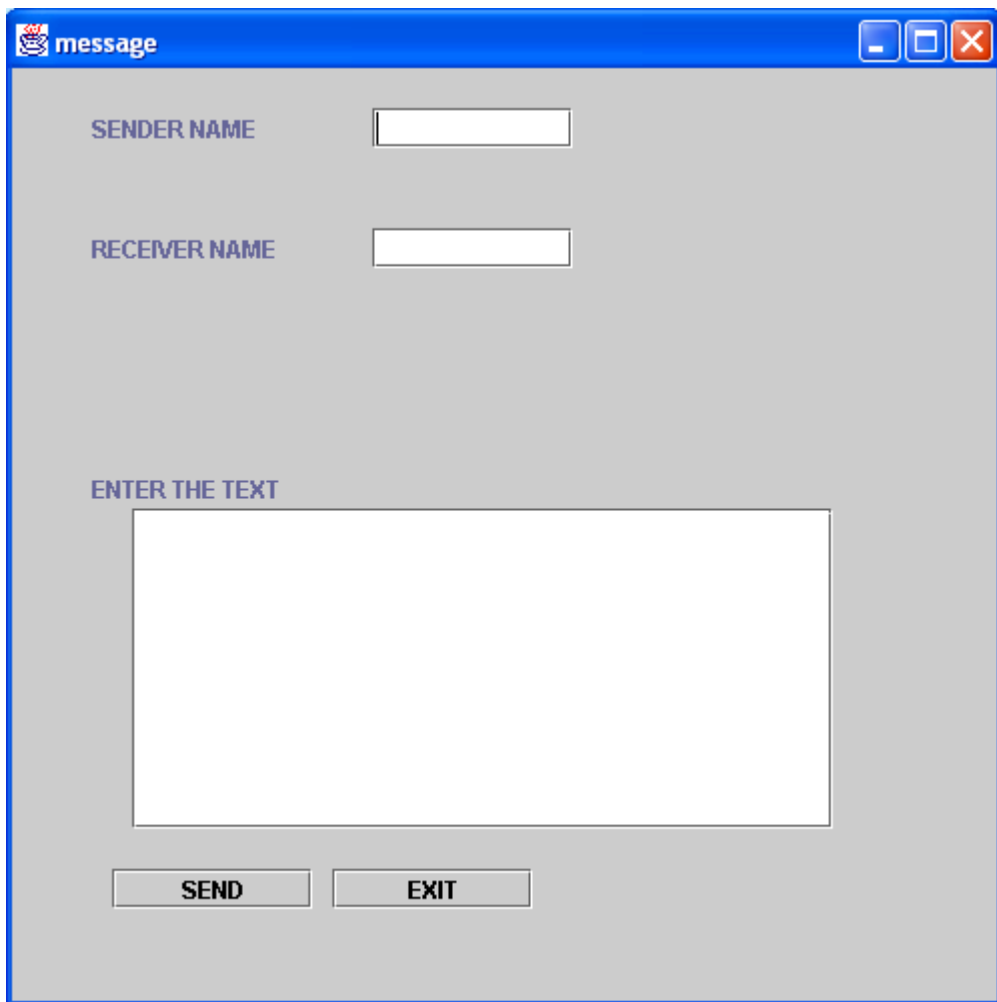


Figure 8. Message

FILE

SENDER NAME

RECEIVER NAME

FILE PATH

Figure 9. File

8. Conclusion

The DNS as an Internet standard to solve the issues of scalability surrounding the hosts.txt file. Since then, the widespread use of the DNS and its ability to resolve host names into IP addresses for both users and applications alike in a timely and fairly reliable manner, makes it a critical component of the Internet. The distributed management of the DNS and support for redundancy of DNS zones across multiple servers promotes its robust characteristics. However, the original DNS protocol specifications did not include security. Without security, the DNS is vulnerable to attacks stemming from cache poisoning techniques, client flooding, dynamic update vulnerabilities, information leakage, and compromise of a DNS server's authoritative files.

In order to add security to the DNS to address these threats, the IETF added security extensions to the DNS, collectively known as DNSSEC. DNSSEC provides authentication and integrity to the DNS. With the exception of information leakage, these extensions address the majority of problems that make such attacks possible. Cache poisoning and client flooding attacks are mitigated with the addition of data origin authentication for RRSets as signatures are computed on the RRSets to provide proof of authenticity. Dynamic update vulnerabilities are mitigated with the addition of transaction and request authentication, providing the necessary assurance to DNS servers that the update is authentic. Even the threat from compromise of the DNS server's authoritative files is almost eliminated as the SIG RR are

created using a zone's private key that is kept off-line as to assure key's integrity which in turn protects the zone file from tampering. Keeping a copy of the zone's master file off-line when the SIGs are generated takes that assurance one step further.

DNSSEC can not provide protection against threats from information leakage. This is more of an issue of controlling access, which is beyond the scope of coverage for DNSSEC. Adequate protection against information leakage is already provided through such things as split DNS configuration.

DNSSEC demonstrates some promising capability to protect the Internet infrastructure from DNS based attacks. DNSSEC has some fairly complicated issues surrounding its development, configuration, and management. Although the discussion of these issues is beyond the scope of this survey, they are documented in RFC 2535 and RFC 2541 and give some interesting insight into the inner design and functions of DNSSEC. In addition to keep the scope of this paper down, many topics such as secure zone transfer have been omitted but are part of the specifications in RFC 2535. The first official release of a DNSSEC implementation is available in BIND version 8.1.2.

APPENDICES

JAVA

Java was designed to meet all the real world requirements with its key features, which are explained in the following paragraph.

SIMPLE AND POWERFUL

Java was designed to be easy for the professional programmer to learn and use efficiently. Java makes itself simple by not having surprising features. Since it exposes the inner working of a machine, the programmer can perform his desired actions without fear. Unlike other programming systems that provide dozens of complicated ways to perform a simple task, Java provides a small number of clear ways to achieve a given task.

SECURE

Today everyone is worried about safety and security. People feel that conducting commerce over the Internet is as safe as printing the credit card number on the first page of a Newspaper. Threatening of viruses and system hackers also exists. To overcome all these fears java has safety and security as its key design principle.

Using Java Compatible Browser, anyone can safely download java applets without the fear of viral infection or malicious intent. Java achieves this protection by confining a java program to the java execution environment and by making it inaccessible to other parts of the computer. We can download applets with confidence that no harm will be done and no security will be breached.

PORTABLE

In java, the same mechanism that gives security also helps in portability. Many types of computers and operating systems are in use throughout the world and are connected to the internet. For downloading programs through different platforms connected to the internet, some portable, executable code is needed. Java's answer to these problems is its well designed architecture.

OBJECT-ORIENTED

Java was designed to be source-code compatible with any other language. Java team gave a clean, usable, realistic approach to objects. The object model in java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non -objects.

DYNAMIC

Java programs carry with them extensive amounts of run-time information that is used to verify and resolve accesses to objects at run-time. Using this concept it is possible to dynamically link code. Dynamic property

of java adds strength to the applet environment, in which small fragments of byte code may be dynamically updated on a running system.

NEWLY ADDED FEATURES IN JAVA 2

- **SWING** is a set of user interface components that is entirely implemented in java the user can use a **look and feel** that is either specific to a particular operating system or uniform across operating systems.
- Collections are a group of objects. Java provides several types of collection, such as linked lists, dynamic arrays, and hash tables, for our use. Collections offer a new way to solve several common programming problems.
- Various tools such as javac, java and javadoc have been enhanced. Debugger and profiler interfaces for the JVM are available.
- **Performance** improvements have been made in several areas. A **JUST-IN-TIME (JIT) compiler** is included in the JDK.
- **Digital certificates** provide a mechanism to establish the identity of a user, which can be referred as electronic passports.
- Various security tools are available that enable the user to create the user to create and store cryptographic keys and digital certificates, sign Java Archive(JAR) files, and check the signature of a JAR file.

SWING

Swing components facilitate efficient graphical user interface (GUI) development. These components are a collection of lightweight visual components. Swing components contain a replacement for the heavyweight AWT components as well as complex user interface components such as Trees and Tables.

Swing components contain a pluggable look and feel (PL & F). This allows all applications to run with the native look and feel on different platforms. PL & F allows applications to have the same behaviour on various platforms. JFC contains operating system neutral look and feel. Swing components do not contain peers. Swing components allow mixing AWT heavyweight and Swing lightweight components in an application.

The major difference between lightweight and heavyweight components is that lightweight components can have transparent pixels while heavyweight components are always opaque. Lightweight components can be non-rectangular while heavyweight components are always rectangular.

Swing components are JavaBean compliant. This allows components to be used easily in a Bean aware application building

program. The root of the majority of the Swing hierarchy is the **JComponent** class. This class is an extension of the AWT Container class.

Swing components comprise of a large percentage of the JFC release. The Swing component toolkit consists of over 250 pure Java classes and 75 Interfaces contained in about 10 Packages. They are used to build lightweight user interfaces. Swing consists of User Interface (UI) classes and non- User Interface classes. The non-User Interface classes provide services and other operations for the UI classes.

Swing offers a number of advantages, which include

- Wide variety of Components
- Pluggable Look and Feel
- MVC Architecture
- Keystroke Handling
- Action Objects
- Nested Containers
- Virtual Desktops
- Compound Borders
- Customized Dialogues
- Standard Dialog Classes
- Structured Table and Tree Components
- Powerful Text Manipulation
- Generic Undo Capabilities
- Accessibility Support

References

1. Albitz, P. and Liu, C., (1997) 'DNS and Bind', 2nd Ed., Sebastopol, CA, O'Reilly & Associates, pp.1-9.
2. Herbert Schildt, Edition (2003) 'The Complete Reference JAVA 2' Tata McGraw Hill Publications
3. IETF DNSSEC WG, (1994) 'DNS Security (dnssec) Charter', IETF.
4. Michael Foley and Mark McCulley, Edition(2002) 'JFC Unleashed' Prentice-Hall India.
5. Mockapetris, P., (1987) 'Domain Names - Concepts and Facilities'.