A New Approach to DNS Security (DNSSEC)

Giuseppe Ateniese
Department of Computer Science and
JHU Information Security Institute
The Johns Hopkins University
3400 North Charles Street
Baltimore, MD 21218, USA
ateniese@cs.jhu.edu

Stefan Mangard
Institute for Applied Information
Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a
8010 Graz, Austria
stefan.mangard@iaik.at

ABSTRACT

The Domain Name System (DNS) is a distributed database that allows convenient storing and retrieving of resource records. DNS has been extended to provide security services (DNSSEC) mainly through public-key cryptography. We propose a new approach to DNSSEC that may result in a significantly more efficient protocol. We introduce a new strategy to build chains of trust from root servers to authoritative servers. The techniques we employ are based on symmetric-key cryptography.

Keywords

Domain Name System Security (DNSSEC), Authentication Protocols, Digital Signatures, Symmetric Encryption

1. INTRODUCTION

The Domain Name System (DNS) [14, 15, 16] is a hierarchically distributed database that provides information fundamental to Internet operations, such as translating between human readable host names and Internet Protocol (IP) addresses. Due to the importance of the information served by DNS, there is a strong demand for securing communication within the DNS system. The current (insecure) DNS does not prevent attackers from modifying or injecting DNS messages. Users accessing hosts on the Internet rely on the correct translation of host names to IP addresses by the DNS system. A typical attack, referred to as DNS spoofing, allows an attacker to manipulate DNS answers on their way to the users. If an attacker makes changes in the DNS tables of a single server, those changes will propagate across the Internet. Recently, the RSA Security web page was hijacked by spoofing the DNS tables [11]. In short, the attacker created a fake web page and then redirected to it all the legitimate traffic to the RSA Security's original page. Interestingly, the attacker didn't crack the DNS server of the company but rather the DNS server upstream in the network.

Increasingly, DNS is also being used to perform load distribution among replicated servers. For instance, companies such as Akamai have used DNS to provide Web content distribution. Moreover, there is consensus that, since DNS is a global and available database, it can be employed as a Public Key Infrastructure (PKI) which would help with enabling e-commerce applications by making public keys globally accessible.

Securing DNS means providing data origin authentication and integrity protection. Confidentiality is not required as the information stored in the DNS database is supposedly public. When communication requirements call for private channels, IP security (IPSEC) is the currently selected candidate system which could easily interface with DNS.

Existing proposals for securing DNS are mainly based on public-key cryptography. In this paper we describe a new approach based on symmetric (or secret-key) cryptographic techniques. Our solution enables a wide range of secure services previously believed to be impractical or too difficult to manage, such as mutual authentication.

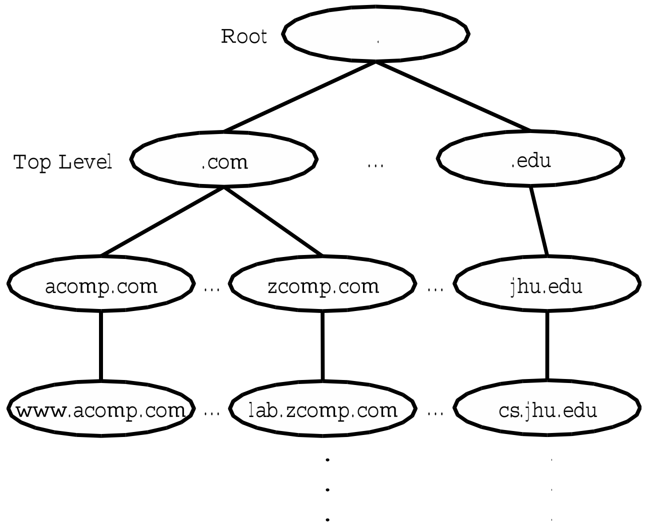

Figure 1: DNS Domains

2. PRELIMINARIES

We first give an overview of DNS and the current proposal for secure DNS (DNSSEC) as described in [20]. For a detailed discussion on DNS, we refer the readers to [1].

2.1 Overview of DNS

As already mentioned, DNS is a global, hierarchical and distributed database. This database associates names, which are referred to as domain names, with certain data contained in resource records (RRs). Records linked to a domain name can be of different types, but the *address* type is the most common one. There can be multiple RRs of the same type for one domain name. The set of resource records of the same type is called a resource record set (RRset).

Since domain names need to be globally unique, a hierarchical naming scheme is used. A domain name refers to a node in a tree (Figure 1) which is called the domain name space. This tree of domain names is very similar to the structure of a UNIX file system. Each subtree is called a domain. For example, the subtree rooted on the `.com` node is called the *.com domain* and includes all domain names ending with `.com`. The nodes that are direct children of the root node are called top level domains.

Communication with the DNS database follows the client/server paradigm. The domain name tree is divided into zones, which usually are contiguous parts of the tree. Zones are defined by the process of delegation which assigns to some organization the responsibility of managing particular subdomains. A zone may contain information about a domain and its subdomains. Top-level zones, such as `.edu`, would mostly contain delegation information.

For each zone, there are authoritative servers (name servers) answering all queries concerning domain names in that zone. Name servers can be authoritative for multiple zones, too. A DNS client program is called a resolver. There are two kinds of resolvers: *real* resolvers and *stub* resolvers (Figure 2).

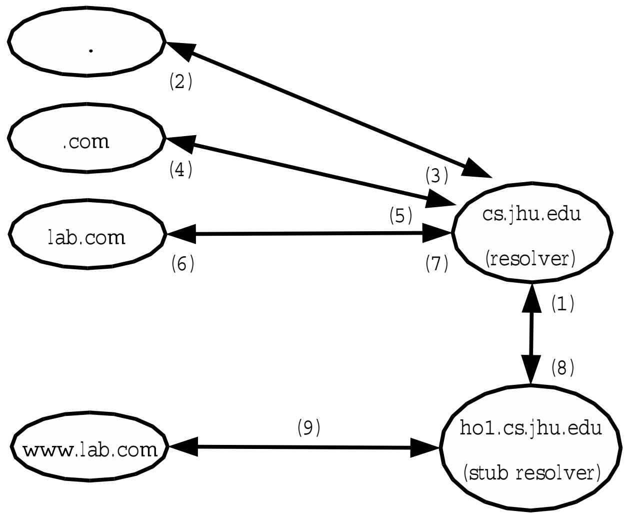

Figure 2: A resolver querying for www.lab.com

- A *stub resolver* is basically a library that needs to be installed on every host that wants to access the DNS database. Every time a query needs to be sent, functions of this library are called and the process of retrieving the desired information is run. Specifically, the stub resolver sends a *recursive query* to a *resolver* which will reply with the information needed.
- A *resolver* is generally located on a DNS server and serves a group of stub resolvers. When a recursive query is received, the resolver usually sends an *iterative query* to one of the root DNS servers serving the root domain. Iterative queries allow a DNS server, which does not have the requested mapping, to indicate the next server in the chain which is "closer" to the authoritative server for those queries.

In the example in Figure 2, the resolver `cs.jhu.edu` receives a recursive query for the IP address of the server `www.lab.com` from host `ho1.cs.jhu.edu`. The resolver then sends an iterative query to a root DNS server, which returns the IP address of the DNS server authoritative for the `.com` zone. The resolver will then query the name server authoritative for `.com` which will return the IP address of the name server authoritative for `lab.com`. Finally, the DNS server of `lab.com` is queried by the resolver and returns the IP address of `www.lab.com` for which it is authoritative. This answer is then forwarded by the resolver to the stub resolver `ho1.cs.jhu.edu`. The entire process is called *resolving.*

Root servers are essential to the functionality of the DNS system. There are currently 13 root DNS servers distributed all over the planet. Caching techniques are employed to reduce the number of requests in order to speed up the resolving process and to reduce network traffic. Consequently, each RR that is returned from a DNS server has a certain time-to-live (TTL) which is the time the RR can be cached.

2.2 Overview of DNSSEC

The first RFC on securing DNS was published in 1997 [18]. Since then, several documents (research papers, internet drafts, and RFCs) have been published on this topic. In this section, we summarize the basic concepts of secure DNS as described in [20, 25, 27].

The primary goal of DNSSEC is to provide authentication and integrity for data received from the DNS database. This is done via digital signature schemes based on public-key cryptography. A possible approach is to sign each DNS message. The general idea is that each node in the DNS tree is associated with a public key of some sort. Each message from DNS servers is signed under the corresponding private-key. It is assumed that one or more *authenticated* DNS root public keys are publicly known. These keys are used to generate *certificates* on behalf of the top level domains, i.e. these keys are used to generate a signature that binds the identity information of each top-level domain to the corresponding public key. The top level domains sign the keys of their subdomains and so on in a process where each parent signs the public keys of all its children in the DNS tree. Considering our example in Figure 2, the resolver, that owns an authentic copy of the root's public key, will receive the IP address of the DNS server of `.com` from the root along with its public key all signed via a pre-specified digital signature algorithm. The public key for the `.com` zone is trusted since it is signed by the root and will be used to sign the public key of the DNS server of `lab.com`. This process is repeated going down across the tree. To associate a domain name with a certain public key, a so called KEY RR [20] is used. Currently, DSA [21] and RSA [22] are the digital signature algorithms supported by DNSSEC. The Diffie-Hellman key agreement protocol is also supported [24, 26].

Two different kinds of signatures for DNS messages as a whole are currently defined: transaction signatures (TSIGs) [25] based on symmetric techniques, and public-key signatures which are abbreviated by SIG(0) [27]. TSIG signatures have been introduced mainly for transactions between local servers, for instance between the resolver and the stub resolver. It is convenient to use TSIG to secure dynamic updates or zone transfers between master and slave servers. SIG(0) is similar to TSIG but employs public-key signatures. SIG(0) may not be practical to use on a large scale but it is useful in case integrity protection and authentication of the message as a whole are desired. SIG(0) could be used to authenticate requests when it is necessary to check whether the requester has some required privilege.

A more efficient alternative employs digital signatures to sign each RRset as described in [20]. The basic idea is to cover each resource record set with a public-key signature which is stored as a resource record called SIG RR [20]. These SIG RRs are computed for every RRset in a zone file and stored therein. A DNS server adds the corresponding *pre-calculated* signature for each RRset in answers to DNS queries. It is imperative for DNS servers to include the entire RRset in a DNS answer as otherwise the resolver could not verify the signature.

For this scheme it is also necessary to introduce an additional type of resource records: NXT RRs [20]. The NXT resource record is associated with a domain name and indicates the types of RRs that are available for that domain name and additionally which domain name is next by dictionary order (the zone should be in canonical order). In order to build a closed chain of NXT records for a zone, the first and the last entry are considered to be next to each other. These NXT resource records are, like any other RRset, signed. If a resolver queries for a domain name or a type of data that does not exist, the corresponding NXT RR and a covering SIG RR are returned. The NXT records identify what does not exist in a zone to avoid generating signatures on general statements of nonexistence which could be replayed. However, notice that an attacker could query for the NXT record of a domain name to find the next domain name in canonical order and repeat the process to learn all the domain names in the zone.

3. OUR PROPOSAL

In this section we describe our approach to DNSSEC. As we employ mostly symmetric (or secret-key) cryptography, we will refer to our proposal as SK-DNSSEC. The proposals of DNSSEC using SIG RRs [20] and SIG(0) [27] make use of public-key cryptography and will thus be referred to as PK-DNSSEC.

3.1 Symmetric Certificates

The system we propose makes use of symmetric ciphers such as AES or Blowfish in CBC mode, and symmetric signatures implemented via Message Authentication Code (MAC) functions. A practical construction for MAC functions is described in [3, 19], called HMAC. To achieve private and authenticated channels, we can combine encryption techniques with MAC functions. In our system, we would encrypt a message m as follows:

$$\mathrm{E}_k\left[m, \mathrm{MAC}_\ell(m)\right] = \mathrm{E}_k\left[\mathrm{MAC}_\ell(m)||m\right], [1]$$

where E_k is a symmetric encryption algorithm, and k and ℓ are independent random session keys. A careful analysis of the building block $\mathrm{E}_k(m, MAC_\ell(m))$, in the context of secure channels, can be found in [9]. We assume that the symmetric encryption algorithm, $\mathrm{E}_k(\cdot)$, is computed in CBC mode and secure against chosen-plaintext attacks. Furthermore, we assume that the message authentication code is secure against chosen-message attacks.

There are several other secure methods that provide authentication and privacy [4, 9], which can be employed in our system. In particular, encrypting first the message and then computing the MAC function over the ciphertext is usually preferable since the encryption function does not have to be computed in CBC mode [9] and invalid ciphertexts can be discarded without the overhead of decryption. However, direct authentication of the plaintext is a desirable property, and also notice that we employ the MAC to authenticate messages that are not necessarily private and that can be predictable or induced. Moreover, in the scheme we are proposing, it is impossible to apply a chosen plaintext attack as DNS servers encrypt exclusively freshly self-generated session keys.

It is assumed that each message sent contains a *nonce*. A nonce is typically a random number which helps preventing several kinds of attacks, such as replay attacks. We will consider a nonce as a pair of values, a random number and a timestamp, respectively. Timestamps provide message *freshness* but must be used carefully in asynchronous systems such as the Internet. However, the upcoming secure network time protocols, such as secure NTP [12], should encourage a more consistent use of timestamps.

We introduce the concept of *DNS symmetric certificates* that will be used to efficiently build a chain of trust from a DNS root to the authoritative server. Our strategy is very similar to the one introduced by Davis and Swick [6]. A symmetric certificate binds the owner's identity to a secret key and it is generated by using symmetric cryptographic techniques.

3.2 Motivation

Any public-key cryptographic algorithm requires some form of authentication. Usually, this is achieved via trusted certification authorities (CAs) which generate certificates on the users' behalf. A certificate binds the user's identity to a public key and contains other pertinent information such as inception and expiration dates. It is assumed that a method to verify the validity of certificates is known a priori.

Secure communication based on public-key cryptography has several positive aspects and in particular: It requires only a *functional* Trusted Third Party (TTP) and the TTP has the ability to operate *off-line* in the sense that it does not have to be involved during secure transactions. A functional TTP does not have access to the secret keys corresponding to the public keys, hence, in some scenarios, it does not have to be unconditionally trusted. However, this does not apply to the case of certified public keys. Indeed, the CA (i.e., the TTP) could generate a fake certificate and pretend to be someone else. It could then read encrypted messages or sign

[1] The symbol || denotes the concatenation operation.

K_{XY}	secret-key pair (K^1_{XY}, K^2_{XY}) shared by X and Y (Y's master key)
K^1_{XY}	secret key shared by X and Y used for encryption
K^2_{XY}	secret key shared by X and Y used for MAC functions
K_{R_0}	root's (R_0) key pair $(K^1_{R_0}, K^2_{R_0})$ (root's master key)
P_{XY}	*symmetric certificate* shared by X and Y
$Info(P_{XY})$	contains relevant information about P_{XY}
PE_{R_0}	public-key encryption under root's public key
$Nonce_i$	$r\|\|t$, where r is a random number and t a timestamp
I_X	identity information about X

Table 1: Notation

arbitrary messages under the stolen identity. In this case, the CA must be fully trusted. On the other hand, secret-key cryptography requires unconditionally trusted TTPs which, in addition, have to be operated on-line. However, secret-key algorithms are extremely fast and make use of relatively short keys.

The PK-DNSSEC system [20, 27] uses the entire DNS tree of domains as an on-line certification authority that, for each request of host mapping information, returns the public key of the authoritative server responsible for that host. The DNS tree builds a chain of trust from the root to the authoritative server and each node that is passed through acts as a CA for its child nodes. In addition, the information retrieved from the DNS database is signed by DNS servers which must be then unconditionally trusted.

Let us emphasize these two points:

1. the DNS system is involved in any request, thus acting as an *on-line* certification authority;

2. the information retrieved from the database is signed by name servers which implies that each DNS server must be unconditionally trusted.

From the previous two points, it is clear that the network configuration and the trust model of PK-DNSSEC would not change if secret-key cryptography were employed in place of public-key cryptography.

In the next sections, we investigate a new approach to DNS security based on secret-key cryptography. We propose a new system which addresses some open issues of the current DNSSEC proposals.

3.3 The New Protocol

The notation used throughout the paper is shown in Table 1. Given the DNS tree of domains, it is assumed that each node shares a key with its parent, called *master key*. For instance, the node `serv.com` (subdomain) shares a key with `.com` (domain) that is referred to as the master key of `serv.com`. The root domain has an asymmetric key pair (public and secret keys) as well as its own master key that is not shared with any other node. The root's master key is used to start the process of building the chain of trust from the root to the authoritative servers.

The general idea behind our proposal is quite simple. As a viable example, suppose that a local name server (acting as a resolver) U queries the root domain server for the IP address of `host.company.com`. The root is not authoritative for this query and thus will refer the resolver to the DNS server `.com`. (It is assumed that U has an authentic copy of the root's public key.) The root server generates a secret key K_a which is sent (encrypted) to U along with a symmetric certificate for `.com`. The key K_a will be shared by U and the server `.com`. The symmetric certificate is an encryption under the master key of `.com` of the key K_a and information about U. The name server U queries `.com` by sending the original DNS request along with the symmetric certificate generated by the root server. The `.com` server will retrieve the key K_a from the certificate and use it to encrypt a freshly generated key K_b. To safely communicate such a key to `.company.com`, the server `.com` inserts the key K_b into another symmetric certificate created for `.company.com`. The key K_b will be shared by U and the server `.company.com`. Finally, the server `.company.com` will send the IP address of `host.company.com` to U symmetrically signed with K_b.

The entire process is done to create a trusted path from the root to `.company.com`. Symmetric certificates can be seen as a sort of *tickets* in the Kerberos system [10, 17] and the trusted path from the root to the authoritative server is similar to the trusted path created in Kerberos from the authentication server to the destination server going through the ticket-granting server [17].

Master keys are used to generate symmetric certificates which allow safe transport of secret keys from the parent to the child in the DNS tree. Each node shares a master key with its node parent. This is usually achieved manually or via out-of-band cryptographic techniques (as it is done for transaction signatures (TSIGs) [25]).

Name servers acting as resolvers must have an authentic copy of the root's public key. (This is the same assumption made in the PK-DNSSEC proposal.) Such a public key can be recovered from trusted sources, bundled with the DNS server software distribution or could just be printed in some popular newspapers.

When a DNS resolver contacts the root server for the first time, it sends a request *DNS_RootCert_Req* encrypted under the public key of the root. Inside the encryption, the resolver includes two secret keys K_1, K_2 and a protocol header PH which should minimally contain the identities of both the resolver and the root server, lifetime of the encryption, and a nonce (i.e., a random number and a timestamp). The root server will generate a symmetric certificate for itself, the *DNS root certificate*, which is symmetrically signed and encrypted with the key K_2 and K_1, respectively. The encryption is then sent to the resolver. The symmetric certificate will be used to create an authenticated channel between the resolver and the root server.

Public-key cryptography is used only the first time the resolver communicates with the root server, since the root does not store (and will never store) locally any secrets shared with the resolvers. However, the next time the resolver contacts the root server, it will communicate securely via efficient symmetric-key protocols (see Remark 1).

`DNS Root Certificate:`

$R_0 \longleftarrow U:\ PE_{R_0}(PH, K_1, K_2, \text{DNS\_RootCert\_Req});$

$R_0 \longrightarrow U:\ P_{R_0U}, E_{K_1}(K_{R_0U}, MAC_{K_2}(K_{R_0U}, P_{R_0U}));$

$$P_{R_0U} = Info(P_{R_0U}),\ E_{K^1_{R_0}}(K_{R_0U},\ \text{MAC}_{K^2_{R_0}}(Info(P_{R_0U}), K_{R_0U})).$$

The communication cost associated with the above request (two network messages) can be amortized. Indeed, the request for the root symmetric certificate can be sent along with an ordinary DNS request.

The value $Info(P_{XY})$ contains relevant information about the certificate P_{XY}, shared by the servers X and Y, which is similar to the information contained in standard public-key certificates. In particular, $Info(P_{XY})$ has to minimally contain the identity strings I_X and I_Y, inception and expiration dates, details about the encryption and authentication algorithms employed, certificate and key unique identifiers (in case of multiple keys), and the identity of the creator of the certificate.

The resolver U may send a DNS query to the root domain server which is not authoritative for it. The root does not need to store any information about U as all it needs is stored into the symmetric certificate enclosed with the request. In particular, the certificate contains the necessary secret keys used to create private and authenticated channels between the root and the host U.

`DNS Request to Root with Symmetric Certificate:`

$R_0 \longleftarrow U:\ P_{R_0U}, \text{DNS\_Req}, Nonce_0;$

$R_0 \longrightarrow U:\ P_{R_1U}, \text{DNS\_Ans}_0, E_{K^1_{R_0U}}(K_{R_1U},\ \text{MAC}_{K^2_{R_0U}}(\text{DNS\_Ans}_0, Nonce_0, K_{R_1U}));$

$$P_{R_1U} = Info(P_{R_1U}),\ E_{K^1_{R_0R_1}}(K_{R_1U},\ \text{MAC}_{K^2_{R_0R_1}}(Info(P_{R_1U}), K_{R_1U})).$$

The response from the root is signed using the MAC function whose output is also encrypted under the key shared by the root and the server U. The field DNS_Ans contains the answer to U's query in accordance with the DNS protocol. In particular, DNS_Ans contains information that undeniably identifies it as the answer to the original query DNS_Req (e.g., DNS_Ans may simply include DNS_Req).

The symmetric signature contains also the nonce generated by U and a secret key that will be shared by U and R_1. Furthermore, in case of multiple keys being used, an additional piece of information containing key identifiers should be included. The symmetric certificate of R_1, P_{R_1U}, is sent along with the signature. It contains the secret key K_{R_1U} shared by R_1 and U. Finally, the encryption is computed under R_1's master key and contains a MAC function which acts as symmetric signature and precludes malleability attacks. Iterative requests to intermediated servers have the same structure:

`DNS Request to Intermediate Server:`

$R_i \longleftarrow U:\ P_{R_iU}, \text{DNS\_Req}, Nonce_i;$

$R_i \longrightarrow U: P_{R_{i+1}U}, \text{DNS\_Ans}_i, E_{K^1_{R_iU}}(K_{R_{i+1}U},\ \text{MAC}_{K^2_{R_iU}}(\text{DNS\_Ans}_i, Nonce_i, K_{R_{i+1}U}));$

$$P_{R_{i+1}U} = Info(P_{R_{i+1}U}),\ E_{K^1_{R_iR_{i+1}}}(K_{R_{i+1}U},\ \text{MAC}_{K^2_{R_iR_{i+1}}}(Info(P_{R_{i+1}U}),\ K_{R_{i+1}U})).$$

Symmetric certificates can be cached and used later by U for similar requests.

Once the authoritative server is queried by U, it will send the answer signed using the MAC function. No other symmetric certificates are generated in this final phase.

`DNS Request to Authoritative Server:`

$R_n \longleftarrow U:\ P_{R_nU}, \text{DNS\_Req}, Nonce_n;$

$R_n \longrightarrow U:\ \text{DNS\_Ans}_n, \text{MAC}_{K^2_{R_nU}}(\text{DNS\_Ans}_n, Nonce_n);$

DNS_Ans_n is the authoritative answer.

The above is a high-level description and it does not include implementation details needed for a robust and efficient execution of the protocol. We refer the reader to [2] for technical details about the implementation of the system.

If mutual authentication and protection for DNS requests are needed (see section 4), then, for any $0 \leq i \leq n$, the first message

$$R_i \longleftarrow U:\ P_{R_iU}, \text{DNS\_Req}, Nonce_i$$

becomes:

$$R_i \longleftarrow U:$$

$$P_{R_iU}, \text{DNS\_Req}, Nonce_i, \text{MAC}_{K^2_{R_iU}}(\text{DNS\_Req}, Nonce_i).$$

The timestamp in $Nonce_i$ should be verified by R_i, and U should set a timeout period after which rejecting signed messages which contain $Nonce_i$. Obviously, in case of mutual authentication, the server U should authenticate itself first. This needs to be done only once, for instance it can be done together with the request for the DNS root symmetric certificate. The server U would sign the public-key encryption (the first message) by computing:

$$SIGN_U(PH_1, PE_{R_0}(PH, K_1, K_2, \text{DNS\_RootCert\_Req})),$$

where PH_1 is a protocol header similar to PH which also contains a sentence clearly stating that the signature is computed over an encryption in accordance with the DNS protocol. The public-key of U may be embedded in a certificate signed by some certification authority recognized by the root server. Finally, the root will set an appropriate flag inside the symmetric certificate to inform other nodes that the resolver was indeed authenticated. Similarly, this entire task can be performed by any downward name server and not necessarily by the root.

The resolving process will end once the resolver U sends the information retrieved from the authoritative server to the stub resolver H, which requested it. The host H could share a key with the local name server U so they could communicate securely via mechanisms that may be already in place, such as TSIG [25]. The secret key may be generated by the system administrators and stored manually in both U and H.

Remark 1. (Hybrid Approach). Notice that once a name server becomes operative for the first time, it has to query one of the root servers for a root symmetric certificate. This forces the root server to decrypt a message each time a name server needs to use public-key encryption. This, however, is not an issue since the name server will retrieve the root symmetric certificate and use it thereafter. Moreover, a public-key encryption function is never applied on the entire message but rather on a short symmetric key[2].

Observe that the problem in DNSSEC is not that root servers cannot compute cryptographic functions fast enough. Indeed, root servers are very powerful (and expensive) machines [3] and, if necessary, they can be equipped with specialized hardware, such as high speed encryption accelerators. The real problems in DNSSEC are the potential increase of network traffic due to larger DNS messages and the high cost of cryptographically **verifying**, at the resolver side, public-key digital signatures computed over zone data.

Alternatively, however, it is possible to implement a hybrid system which uses both PK-DNSSEC and SK-DNSSEC. The idea is to let some servers sign answers via public-key signatures which will include, at a certain point, subdomains' public keys used by SK-DNSSEC. Suppose, for instance, that root servers are configured to use PK-DNSSEC whereas top-level domains use SK-DNSSEC. A resolver may query one of the roots for the IP address of `comp.ccc.com`. Since the root is not authoritative, it will send a referral which will be signed via a public-key signature. The signature contains the SK-DNSSEC public-key of `.com` used for encryption. The resolver can then start using SK-DNSSEC by requesting a symmetric certificate from `.com` via the protocol, introduced earlier, used for requesting DNS root certificates.

This modus operandi may be useful to relieve certain name servers (root servers, for instance) from computing public-key decryptions.

[2]A fast Pentium III (800Mhz) server can compute about 200 RSA decryptions of a short key (128 bits) per second, when the modulus is 768 bits long.

[3]The root server F.root-servers.net [8] is a virtual parallel machine with currently 8 processors and a total of 8 Gb of memory.

Operation	key len.	Ops per sec.
Create SK authoritative ans	128	52083.3
Verify SK authoritative ans	128	98039.2
Create SK referral	128	33444.8
Verify SK referral	128	84033.6
Create DSA signature	768	349.5
Verify DSA signature	768	286.2
Create RSA signature	768	198.8
Verify RSA signature	768	1990.0
Create DSA signature	1024	234.5
Verify DSA signature	1024	192.7
Create RSA signature	1024	114.4
Verify RSA signature	1024	1355.6

Table 2: Ops per sec. over a 500-byte message on a fast Pentium III

Remark 2. The system described so far considers only iterative queries. A resolver, however, may send a recursive query to a name server for information about a particular domain name. The queried server is then forced to interact with other name servers in order to find the answer and respond with the requested data. (Stub resolvers always send recursive queries.) Recent versions of BIND allow configuration of servers to ignore or refuse recursive queries. In particular, root and top-level name servers generally do not accept recursive queries given the burden associated with them.

Handling authenticated recursive requests in SK-DNSSEC is natural since we assume the existence of master keys shared between child nodes and their parent domains. It is sufficient to employ the existing transaction signatures (TSIGs) [25] to secure the exchange of information between a child node and its parent[4].

Remark 3. A queried name server may return several servers authoritative for the same zone. Resolvers based on BIND use the *roundtrip time*, or RTT, to choose between the name servers authoritative for the same zone. The RTT, which measures the time necessary to a name server to respond to queries, ensures that the resolver will, most likely, query the *closest* authoritative name server. In SK-DNSSEC, we have currently adopted a single strategy: Name servers, authoritative for the same zone, store the same secrets. Only one symmetric certificate needs to be generated and the resolver would just select the name server with the lowest RTT.

4. ANALYSIS

We believe SK-DNSSEC addresses some interesting issues:

Performance. In PK-DNSSEC, it is possible to reuse digital signatures in order to save time. However, signatures still have to be verified each time a response is received from DNS servers. Even when using RSA, which is faster in verification than DSA, the verification of many digital signatures would require substantial computational resources. In contrast, symmetric signatures can be verified very efficiently.

[4]For instance, the master key may be stored in both servers via the TSIG *key* statement. The TSIG *server* statement may be extended so that each server is instructed to sign recursive queries and answers sent to a direct child or parent domain.

A fast Pentium machine, for instance, can verify more than $98,000$ symmetric signatures per second. On the other hand, SK-DNSSEC requires queried servers to sign each response. However, as shown in Table 2, computing SK-DNSSEC authenticated authoritative answers and referrals can be extremely rapid. Table 2 reports the number of SK-DNSSEC (which is abbreviated to the shorter "SK") operations that can be computed per second on a Pentium III (800Mhz) machine. In particular, creating a SK authoritative answer requires the name server to first open and verify the symmetric certificate sent by the resolver and then compute a MAC over the answer. To verify an authoritative answer, the resolver simply checks the MAC. Computing a SK referral is more complicated. It requires the name server to open and verify the certificate, sign the answer and create a symmetric certificate for its child node. Verifying such an answer requires decrypting a session key and checking the MAC. Details on these operations can be found in Section 6.3. On reading these numbers, it should be taken into account that the F.root-servers.net, often the busiest root server on the Internet according to [8], receives about $3,150$ requests per second (more than 272 million requests per day).

Table 2 also reports similar estimates for public-key signatures, computed over the hash of a 500-byte message, with comparable level of security (consider that applying a brute-force attack against a 128-bit-key symmetric cipher is roughly as difficult as factoring a public-key modulus of approximately $2,300$ bits).

Network traffic. DNS usually runs over UDP. One of the major problems of PK-DNSSEC is that authenticated queries and responses do not fit into a 512-byte UDP datagram [13]. Moreover, if a name server based on PK-DNSSEC with SIG RR is queried for n different types of RRs, it would return n public-key signatures (one for each RRset). SK-DNSSEC does not require to sign entire RRsets and, as shown in Section 6.2, the final authenticated DNS message is very short and may well fit into a 512-byte UDP datagram.

Storage. SK-DNSSEC uses very short certificates. With the same amount of cache it is possible to store more symmetric certificates which, in principle, should improve the delay performance and reduce the number of DNS messages in the network. In addition, there is no need for NXT RRs which authenticatably deny the nonexistence of requested records. This fact makes the zone data file more manageable and smaller.

Replay Attacks. SK-DNSSEC provides protection against replay attacks since signatures cannot be reused. Signatures in PK-DNSSEC may be exposed to replay attacks unless the inception and expiration dates are very close. The expiration date is usually set to be 4 or 16 times the TTL [20]. Finally, as mentioned earlier, there is no need for NXT RRs which were introduced to avoid to release an authenticated general statement of nonexistence which, clearly, can be replayed.

Mutual Authentication. When necessary, mutual authentication can be achieved very efficiently in SK-DNSSEC as it requires only an additional MAC computation. Indeed, once a DNS server receives a request, it must compute a MAC function in order to determine whether the request comes from an authorized name server.

Access control lists implemented in BIND version 8 and 9 [5] demand mutual authentication to prevent IP spoofing attacks. In particular, in BIND 8 and 9, it is possible to create IP address-based access control lists to queries via the named address match list *acl* and the *allow-query* substatement or the more versatile BIND 9 *view* mechanism. Access control lists specify the IP addresses of the resolvers that are allowed to query the server. Similarly, *allow-recursion* can be used to specify which resolver is allowed to send recursive queries.

Without any mutual authentication mechanism in place, access control lists are practically useless [6].

Confidentiality. SK-DNSEC can provide confidentiality for queries or answers, if needed, by including the fields DNS_Req and DNS_Ans directly into the symmetric encryption. DNS is a public service, therefore confidentiality is not required. However, DNS can be used for purposes other than DNS. For instance, DNS can be used to exchange private host keys among authorized users or a DNS-based system can be used to manage large private domain spaces of corporations which may have the legitimate need to hide certain parts of their name space.

Finally, a note about the threat model. Observe that, SK-DNSSEC requires DNS servers to store locally secret keys which may be esposed in case the servers are fully compromised, whereas PK-DNSSEC is secure as long as the off-line signing authority is honest.

The threat scenario outlined in [5] requires that the attacker has complete control of a DNS machine and that the changes to the corresponding portions of the name tree, resulting from the attack, are accepted as correct by other machines. Once the attacker is able to impersonate a DNS server (e.g., secret keys may have been exposed), nothing can be done to preclude malicious changes to the portion of the DNS system where the compromised server is authoritative.

5. SK-DNSSEC AS PKI

The infrastructure provided by secure DNS is very well suited to be used as a public-key distribution system. The properties of DNSSEC make this obvious, as observed in [7]:

- Global real time availability: Any machine connected to the Internet has easy access to DNS.
- Scalability: The hierarchical organization of DNS allows easy scaling.
- Globally unique names: The hierarchical naming system of DNS builds a logical structure for names which are globally unique.
- Cryptographic binding of name and key: KEY resource records facilitate the binding of DNS names to keys.

In PK-DNSSEC, the association of a domain name with a public key is done via KEY RRs. The KEY record can store different kinds of cryptographic keys which can be used

[5]BIND 4.9 has similar access control mechanisms provided by *secure_zone* and *xfrnets*.

[6]To prevent unauthorized dynamic updates or zone transfers, a TSIG-based mechanism is, in most cases, well adequate.

for several applications other than DNS. In addition, certificates and related certificate revocation lists can be stored in CERT RRs [23]. Analogously, SK-DNSSEC could use KEY RRs or CERT RRs to store public keys and certificates. Authoritative servers will certify the authenticity of these public keys by including them into an appropriate field in DNS_Ans.

6. EFFICIENCY

In the following subsections we compare more in detail a DNS system based on SK-DNSSEC with one based on PK-DNSSEC in the following scenarios:

1. *PK-DNSSEC with SIG RR.* For each RRset in the answer, a pre-calculated SIG RR is included. This method provides authentication and integrity only for each RRset since DNS messages are not signed as a whole.

2. *PK-DNSSEC with SIG(0).* DNS messages do not contain SIG RRs, but are rather signed as a whole by SIG(0)-type signatures. The entire message is therefore authenticated and its integrity is guaranteed by public-key cryptographic signatures.

3. *SK-DNSSEC.* DNS messages are secured by symmetric signatures and encryption which create an authenticated (and, optionally, private) channel between resolver and name servers.

It is important to point out that different zones have very different requirements. The root zone, for example, has a relatively small number of entries but the root server receives a huge number of requests per day. Zones such as `.com` have many entries and serve a reasonably big number of requests. Some entries in a zone might be queried very often and some almost never.

6.1 Size of Stored Information

The amount of data stored in a zone file is directly proportional to the number of delegation points and hosts served by the authoritative name server. Due to the variable length of domain names and number of children of each zone, it is not possible to talk about exact numbers of bytes. In Table 3 we show what additional content needs to be stored in a DNSSEC server compared with a traditional DNS server and analyze the impact on the zone file. According to [1], signing a zone file under PK-DNSSEC with SIG RRs increases its size by a factor of seven. In SK-DNSSEC, the zone file remains substantially unchanged and NXT records are not needed. A secret key of approximately 128 bits has to be stored for the parent and for each child node in the zone file.

6.2 Message Size

The size of DNS messages is an important parameter to consider when comparing DNSSEC proposals. As the Table 4 shows, there might be significant differences.

This is mainly due to the fact that a name server based on PK-DNSSEC with SIG RR (as currently implemented in BIND version 9), when queried for a particular resource record, would return the public-key signature computed over an entire RRset along with the RRset itself which is needed to verify the signature. Querying for n different types of RRs would force a name server to return n public-key signatures as well as n corresponding RRsets. The message size may increase notably.

SK-DNSSEC employs very small signatures and only one signature has to be sent for each query. For instance, when using a block cipher with 128-bit keys as encryption algorithm and HMAC-MD5 with 128-bit keys as MAC function, a signature in SK-DNSSEC is only 384 bits long (authoritative answer). On the other hand, each referral includes two signatures (one for the symmetric certificate and one for the actual signature), thus the total space required to authenticate the message is actually 768 bits.

Assuming that a typical DNS message is 100 bytes (800 bits) long [1], it can be deduced that SK-DNSSEC authoritative answers and referrals would perfectly fit into a 512-byte UDP datagram!

6.3 Computational Time

In order to measure the time that is needed to perform cryptographic operations employed by the DNSSEC proposals, we have selected the crypto library *OpenSSL 0.9.6b* [7]. This library is very popular, well written, and free of charge. In addition, it is available for almost any operating system. Most importantly, *OpenSSL* is the standard library used in BIND starting from the version 9.1. We ran our actual time measurement experiments on a *Redhat 7.0 Linux* system using the `gettimeofday()` function from the *GNU C Library 2.2*. The performance tests were run on a Intel Pentium III 800MHz machine with 256MB of RAM.

In Table 5 we show the average time that is needed to perform the cryptographic operations used by DNSSEC. The first column contains the name of the operation, the second the key length in bits and the third the length of the data input in bytes. The forth column contains the average computational time in microseconds. We took the mean value of $10,000$ random inputs for each operation. Finally we normalized each time measurement by the time needed to compute a HMAC-MD5 on a 500-byte message in the last column.

It is necessary to assign specific algorithms and make assumptions concerning the message size if we want to compare the time needed for a specific operation. A typical DNS message is about 100 bytes long (according to [1]) but we used messages of 500 bytes for our measurements to make sure we are considering worst-case scenarios.

SK-DNSSEC uses HMAC-MD5 with a 128-bit key as symmetric signature algorithm and Blowfish in CBC mode with 128-bit keys. (The implementation of Blowfish is almost three times faster than the current implementation of AES.) Therefore, to sign a message or to generate a symmetric certificate, 3×128 bits $= 48$ bytes need to be encrypted. Public-key signatures are computed over the hash of the input.

SK-DNSSEC Name Server.

- *Authoritative Answers.* From the normalized values of Table 5, it can be deduced that generating authoritative answers in SK-DNSSEC would cost the equivalent of 1.88 times a single computation of HMAC-MD5 over a 500-byte message with a key of 128 bits. Indeed, to generate an authoritative answer, the name server

[7]For AES, we actually used the *OpenSSL* developer version.

	PK-DNSSEC with SIG RRs	PK-DNSSEC with SIG(0)	SK-DNSSEC
additional content types	KEY RRs, NXT RRs, SIG RRs	KEY RRs	shared secrets
additional content	at least one SIG RR for every RRset; one NXT RR for every domain; at least one public key for each child domain.	at least one KEY RR for each child.	a shared secret for the parent and each child.

Table 3: Comparison of stored information

	DNSSEC with SIG RRs	DNSSEC with SIG(0)	SK-DNSSEC
DNS Request	It is not possible to secure a request in this scenario.	The request as a whole is signed via a public-key based signature.	The request as a whole is signed via a MAC. A symmetric certificate is attached.
DNS Authoritative Answer	For each RRset there is a SIG RR in the answer. Since a typical answer consists of more than one RRset, several SIG RRs need to be included.	The answer as a whole is signed via a public-key based signature.	The answer as a whole is signed via a MAC.
DNS Referral Answer	In addition to the delegation DNS RRsets (typically more than one), the KEY RRset of the child and a SIG RR for each RRset need to be included.	The KEY RRset of the child is added to the delegation DNS RRsets and the message as a whole is signed via a public-key based signature.	The server returns the delegation DNS RRs signed via a MAC. The MAC and the key of the child are then encrypted and sent along with a symmetric certificate.

Table 4: Message size

operation	key	input	time	norm
Blowfish enc./dec.	128	48	1.7	0.17
Blowfish enc./dec.	128	52	2.0	0.20
AES enc./dec.	128	48	3.8	0.37
AES enc./dec.	128	52	5.0	0.49
HMAC-MD5	128	500	10.2	1.00
HMAC-MD5	128	1000	14.0	1.37
HMAC-SHA1	128	500	16.0	1.57
HMAC-SHA1	128	1000	24.0	2.35
RIPE-MD160	160	500	19.5	1.91
RIPE-MD160	160	1000	30.0	2.94
Create SK authoritat.	128	500	19.2	1.88
Verify SK authoritat.	128	500	10.2	1.0
Create SK referral	128	500	29.9	2.93
Verify SK referral	128	500	11.9	1.17
Create DSA sig.	768	500	2861.6	280.55
Verify DSA sig.	768	500	3492.9	342.44
Create RSA sig.	768	500	5029.7	493.11
Verify RSA sig.	768	500	502.5	49.26
Create DSA sig.	1024	500	4263.6	418.00
Verify DSA sig.	1024	500	5188.6	508.69
Create RSA sig.	1024	500	8741.5	857.01
Verify RSA sig.	1024	500	737.7	72.32

Table 5: Performance comparison

has to open and verify a symmetric certificate which authenticates 100 bytes of data (0.88) and compute HMAC-MD5 over the answer (1).

- *Referrals.* A referral (or delegation answer), instead, would cost as 2.93 times a single computation of HMAC-MD5. Indeed, the name server has to open and verify a symmetric certificate (0.88), compute HMAC-MD5 over the answer (1), encrypt the session keys and the output of the MAC (0.17), and, finally, create a symmetric certificate (0.88).

SK-DNSSEC Resolver.

- *Authoritative Answers.* Verifying an authoritative answer requires a single HMAC-MD5 computation (1).
- *Referrals.* Verifying a referral requires decrypting the keys (0.17) and computing HMAC-MD5 (1) for a total of 1.17.

6.4 Scalability and Interoperability

According to RFC2931 [27], PK-DNSSEC with SIG(0) provides stronger security protection than PK-DNSSEC with SIG RR which in general does not provide protection of the overall integrity of a response. In particular, PK-DNSSEC with SIG(0) provides protection for glue records, DNS requests (mutual authentication), and message headers on requests or responses. SK-DNSSEC provides a similar level of security but at a lower cost. Message authentication codes are very efficient to compute and verify.

Managing DNS symmetric certificates is easier than it might be expected. In fact, symmetric cryptographic techniques may not scale well compared with those based on public-key cryptography. However, this is not the case in the domain name system given its particular tree structure. Initially, each DNS server stores only the public key of the root and successively the root symmetric certificate. Other symmetric certificates will be cached for efficiency improvements. Furthermore, DNS servers do not store anything about resolvers as the information they need is given to them by the parent servers in the DNS tree. If a DNS symmetric certificate of a server is ever lost or compromised, it is always possible to request a new one starting from the root server or from the DNS server upstream in the network.

Symmetric certificates are as manageable as public-key certificates with the exception that DNS symmetric-key certificate cannot be shared.

However, resolvers are usually configured to share the information retrieved from the DNS database only with stub resolvers or name servers acting as such (*"DNS forwarding"*), for which a TSIG-based mechanism would suffice. In particular, resolvers usually do not share DNS responses (for which they are not authoritative or can't provide delegation nodes) with other name servers nor name servers are configured to query other servers that cannot refer to delegated subdomains. Hence, in a strictly hierarchical tree structure (such as in DNS), techniques based on symmetric cryptography scale as satisfactorily as those based on public-key cryptography.

Nevertheless, we would like to design and implement SK-DNSSEC so that it will interoperate completely with PK-DNSSEC. In Remark 1 of Section 3.3, we justify such a strategy. The code is being written having this interoperability issue in mind ([2]). In particular, we plan to investigate the benefits of a hybrid system where PK-DNSSEC may be used to protect root and top-level domains whereas SK-DNSEC could be used to protect the rest of the DNS tree. Employing SK-DNSSEC in a larger scale, though, would be very convenient and would give us the opportunity to argue better about the load distribution expected on the root and top-level name servers.

7. CONCLUSION

Secure DNS is a big change but inevitable. The PK-DNSSEC proposal is an example of dedication and remarkable work currently coordinated by the IETF. BIND version 9 provides already a working and stable framework for PK-DNSSEC. Despite these efforts, DNSSEC is not widely deployed yet even though DNS names are routinely used for authentication. In this paper, we presented a proposal for DNSSEC that, when properly implemented, offers the highest level of security while reducing network traffic. In addition, it reduces storage requirements and enables efficient mutual authentication. Hopefully, the results contained in this paper will stimulate the deployment of DNSSEC and induce beneficial discussions.

Acknowledgments. The first author would like to thank Aniello Del Sorbo, Steve Kent, and Hilarie Orman for helpful discussions on DNS and DNSSEC. Many thanks to the anonymous referees for their insightful comments.

The SK-DNSSEC system is being developed at the Johns Hopkins University Information Security Institute.

8. REFERENCES

[1] Paul Albitz and Cricket Liu, DNS and BIND, 4th Edition O'Reilly, 2001.

[2] G. Ateniese and A. Del Sorbo, "Design and Implementation Issues in SK-DNSSEC", Manuscript in preparation 2001. Available on www.cs.jhu.edu/∼ateniese/skdnssec.html.

[3] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication". In Advances in Cryptology - Crypto 1996 Proceedings, LNCS Vol. 1109, N. Koblitz ed, Springer-Verlag, 1996.

[4] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm", In Advances in Cryptology - Asiacrypt 2000 Proceedings, LNCS Vol. 1976, T. Okamoto ed, Springer-Verlag, 2000.

[5] Steven M. Bellovin, "Using the Domain Name System for System Break-Ins", Proceedings of the Fifth Usenix Unix Security Symposium, pp. 199–208, June 1995.

[6] D. Davis and R. Swick, "Network Security via Private-Key Certificates", USENIX 3rd Security Symposium Proceedings, (Baltimore; Sept. '92). Also in ACM Operating Systems Review, v. 24, n. 4 (Oct. 1990).

[7] James M. Galvin, "Public Key Distribution with Secure DNS", in 6th USENIX UNIX Security Symposium, July 1996.

[8] Information and statistics about F.root-servers.net, www.isc.org/services/public/F-root-server.html

[9] Hugo Krawczyk, "The order of encryption and authentication for protecting communications (Or: how secure is SSL?)". To appear in the proceedings of CRYPTO 2001.

[10] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[11] *RSA Security site defaced.* ZDNet 2000. www.zdnet.com/zdnn/stories/news/0,4586,2437384,00.html

[12] Secure Network Time Protocol (stime), www.ietf.org/html.charters/stime-charter.html

[13] Eastlake, D., "Bigger Domain Name System UDP Replies", Internet Draft, www.ietf.org/proceedings/98aug/I-D/draft-ietf-dnsind-udp-size-02.txt

[14] Lottor, M., "Domain Administrators Operations Guide", RFC 1033, November 1987.

[15] Mockapetris, P., "Domain Names - Concepts and Facilities", RFC 1034, November 1987.

[16] Mockapetris, P., "Domain Names - Implementation and Specifications", RFC 1035, November 1987.

[17] J. Kohl, C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[18] Eastlake, D. and C. Kaufman, "Domain Name System Security Extensions", RFC 2065, January 1997.

[19] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[20] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[21] EastLake, D., "DSA KEYs and SIGs in the Domain Name System (DNS)", RFC 2536, March 1999.

[22] Eastlake, D., "RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)", RFC 2537, March 1999.

[23] Eastlake, D., Gudmundsson, O., "Storing Certificates in the Domain Name System (DNS)", RFC 2538, March 1999.

[24] Eastlake, D., "Storage of Diffie-Hellman Keys in the Domain Name System (DNS)", RFC 2539, March 1999.

[25] Vixie, P., Gudmundsson, O., Eastlake, D. and B. Wellington, "Secret Key Transaction Signatures for DNS (TSIG)", RFC 2845, May 2000.

[26] Eastlake, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.

[27] Eastlake, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.