Applying decentralized trust management to DNS dynamic updates

Pasi Eronen [1, 2] pasi.eronen@hut.fi
Jonna Särs [2] jonna.sars@nixu.fi

[1] Helsinki University of Technology
[2] Nixu Ltd.

Abstract

DNS dynamic updates can be used to modify the data of a DNS zone. This can be used to update DNS records of hosts with dynamic IP addresses, for example. DNS dynamic updates can be authenticated using the DNSSEC transaction signatures or the TSIG mechanism.

While there are existing mechanisms for authenticating the source of update requests, mechanisms for authorization, i.e. specifying who is allowed to change what, are inadequate in many cases.

In this paper, we propose a solution for authorizing DNS dynamic updates, based on the decentralized trust management approach, and more specifically, the KeyNote 2 system. We have also modified the BIND 9 name server to use this approach. Our solution supports the separation of DNS server administration and update authorization, and also allows the specification of more flexible access restrictions than the use of access control lists.

1 Introduction

DNS has long been a good example of the lack of security in the basic Internet infrastructure. It is a critical service, but was originally not designed to resist active attacks. Well known attacks against DNS include spoofing and cache poisoning [2, 22]. Some of the vulnerabilities are caused by implementation bugs, but a key issue in others is the lack of authentication of DNS data and to some extent also the lack of authentication of the communicating parties.

The effort to provide authentication for DNS data and the protocol parties is called DNSSEC. Simultaneously, some new functionality and features have also been added to DNS.

The new features in the DNS protocol, especially DNS dynamic updates [24], have also created new security problems. Dynamic updates allow modification of DNS data, so in addition to authentication of data, the question of authorization (i.e., who is allowed to change what) also important.

In this paper, we identify a number of problems with existing mechanisms for securing DNS dynamic updates, and propose an alternative mechanism, based on the decentralized trust management approach.

The rest of this paper is organized as follows. The next section presents the DNS dynamic updates and the current mechanisms for securing them. Section 3 describes the current authorization mechanisms, and some problems and limitations in these. The basic idea behind trust management and the details of our proposed solution are described in Section 4. Section 5 gives an outline of our implementation, which combines BIND 9 and the KeyNote 2 trust management library. The next section evaluates the proposed solution and describes possible enhancements. Finally, Section 7 contains our conclusions from this research.

2 DNS concepts

The Domain Name System (DNS) is a hierarchical, distributed database that provides mappings between domain names and IP addresses, and other information. We assume the reader is familiar with the basic concepts and uses of DNS.

The DNS security extensions [8] are defined to counter the security problems inherent in the earlier DNS specifications. They provide data integrity and authentication using digital signatures, and optional authentication of transactions (requests and replies). In addition, DNSSEC defines how the necessary public keys are authenticated and stored in DNS.

In this paper we concentrate on the DNS dynamic update mechanism and the extensions needed to secure it. While the authentication of DNS zone data is important for the overall security of the Internet, it is outside the scope of this paper.

2.1 DNS dynamic updates

An interesting new feature of DNS is the possibility to dynamically update DNS data [24]. This can be used, for example, to allow the DHCP server to update DNS records of hosts with dynamic IP addresses. Especially mobile hosts

that roam from one network segment to another would benefit from keeping the same domain name even if their IP addresses change as they move.

Another interesting application of dynamic updates is updating certificates stored in the domain name system [11]. A CA could dynamically add new certificates when they are created. If the certificates could be updated easily and automatically, their lifetimes could be shorter, and the risks related to revocation could be reduced.

We also see the dynamic update mechanism as a potential replacement for the traditional (and error-prone) approach of manually editing the zone files. With dynamic updates, the changes could be done remotely and the client could perform sanity checks on the data. This, however, is a usability issue with very little relevance to security.

The operations that can be performed with dynamic updates are adding and deleting resource records (RRs). There are a few restrictions to how the RRs can be manipulated by dynamic updates. For example, the SOA serial number can only be incremented, and the last NS RR cannot be removed. In addition to specifying the desired update, the request can include preconditions to the update. Such a precondition can be the existence or non-existence of some name or resource record. The server checks the preconditions before the update is performed, and if they are not met, it stops processing the request and returns an error message.

If the prerequisites are satisfied, the server checks if the authenticated requester is authorized to perform the particular update, conducts some additional sanity checks, and finally updates the data. The update is performed as an atomic operation. That is, either all the update commands in a message are performed or none of them are.

We expect the use of dynamic updates to grow in the future. New operating systems are already starting to depend on dynamic updates. For example, Windows 2000 uses them a lot.

As the dynamic updates become more widely used, the need to secure them increases. It is important to notice that there are really two separate DNS use cases with different security requirements. Querying for data requires data authentication but not necessarily authentication of messages. Dynamic updates require transaction authentication and also authorization, i.e., a way to specify who is allowed to change what. The next section describes existing mechanisms for authenticating dynamic updates, and Section 3 discusses authorization.

2.2 DNS transaction authentication

There are several ways that can be used to authenticate the parties involved in a DNS transaction. The most simple means is using the IP addresses of the communicating parties. This method is weak, as it cannot protect message integrity and spoofing the addresses is relatively easy. It could be sufficient in a closed network with relatively low security requirements. In networks with higher security requirements, cryptographic authentication methods should be used.

There are currently three cryptographic mechanisms for authenticating dynamic updates: DNSSEC public key signatures or SIG(0)s, TSIG shared secret signatures with the related TKEY key exchange method, and the GSS-API TSIG/TKEY mechanism. None of these methods encrypt the messages, since DNS data is usually considered public.

DNSSEC request and transaction signatures, or SIG(0)s for short, specify a strong authentication method using public key cryptography [8, 9]. They can be used to protect and authenticate DNS requests and responses. The solution is scalable: the public key of a server or resolver is stored in DNS as a KEY RR and can be found easily by anyone who wants to authenticate it. Unfortunately, public key cryptographic operations can be computationally expensive, so using them for all transactions would have a serious impact on performance.

A more lightweight solution for message authentication and integrity protection are Secret Key Transaction Signatures (TSIG) [23]. The TSIG mechanism uses shared-secret message authentication codes (MACs) instead of public key cryptography. They are computationally inexpensive, but require that the communicating parties share a secret key. The secret keys can be configured manually, or they can be established using the TKEY mechanism.

The TKEY mechanism can support many different ways of establishing secret keys. One of the possible modes is Diffie-Hellman [10]. Diffie-Hellman key exchange is a way for two parties to create a shared secret without requiring any confidentiality of the messages they send each other, so encryption is not needed. However, the messages must be authenticated to prevent spoofing. The authentication can use a TSIG signature if a previous secret key has been established between the parties. Otherwise, it must be done with the SIG(0) public key signatures.

Microsoft's GSS-API TSIG/TKEY mechanism provides a way of using existing Kerberos authentication infrastructure for authenticating update requests [17]. It operates in two phases. First, a TKEY exchange is used to negotiate a security context between the client and the server. That security context is then used to create and verify transaction signatures.

3 DNS update authorization

Simply authenticating the client making an update request is not enough; we also need to know whether the client is allowed to perform that kind of modification. In this section, we describe the currently used approaches for specifying the security policy in this context, and discuss some problems with them.

3.1 Existing solutions

An early proposal for securing dynamic updates suggested that the KEY RR would include a few bits (the "signatory" field) that indicate if the key may be used for dynamic updates [7]. The KEY record's name, class, and the few bits would be used to encode the update policy. However, since there are only four bits available for the purpose, this approach to authorization is hardly flexible enough for most situations. Attempts to use more bits to gain more flexibility have not been very successful either [21]. Indeed, the use of KEY signatory bits for encoding policy has been considered obsolete for a long time.

The current proposal for securing dynamic updates abandons the concept of storing the policy inside DNS, and simply states that the policy is fully implemented in the primary server's configuration [26]. The proposal suggests a couple of things which should be supported by policy implementations, but leaves other aspects to the implementer.

In practice, the policy is currently implemented using local configuration files, which are essentially a form of access control lists.

3.2 Access control lists in BIND

The first version of BIND that supports dynamic updates was BIND 8. In BIND 8 the dynamic updates are protected by very simple ACL facilities. The ACLs can grant a client a permission to update any record in a zone based on an IP address or a TSIG key. Restrictions on the record name or type are not supported.

BIND 9 introduced a more flexible "update-policy" mechanism. The access control decision can be based on the name being updated, the name of the KEY record (or TSIG key), and the record type.

3.3 Problems in existing approaches

We see several problems in the current approaches. The access control lists are usually stored on the primary master name server. Since their contents are very security critical, there has to be a mechanism for authenticating and authorizing modifications to these configuration files. Presently this is usually achieved by standard Unix access control facilities (which cannot express access rights to only a part of a configuration file). An additional complexity is the fact that the name server is not necessarily operated by the same party which actually "owns" the zone (and should be responsible for deciding who can change it). Since a name server can host thousands of zones, giving shell accounts for everyone is not an attractive solution.

Another problem is related to the ACL's granularity of expression. While in theory the ACL can express every possible characteristic of a DNS update request, in practice the configuration file parser allowing such flexibility would be fairly complex. Thus, as seen in the case of BIND, implementers are likely to support only the simple cases. This could hamper the adoption of dynamic updates for new, innovative use cases.

4 Proposed solution

The limitations of ACLs have been recognized before. In [4] Blaze et al. argue that "the use of identity-based public-key systems in conjunction with ACLs are inadequate solutions to distributed (and programmable) system-security problems." *Trust management*, introduced by Blaze et al. [5] proposes an alternative solution.

4.1 Decentralized trust management

Trust management systems use a set of unified mechanisms for specifying both security policies and security credentials. The credentials are signed statements (certificates) about what principals (users) are allowed to do. Thus, even though they are commonly called certificates, they are fundamentally different from traditional name certificates. Usually the access rights are granted directly to the public keys of users, and thus trust management systems are sometimes called key-oriented PKIs.

Examples of trust management systems include PolicyMaker, which originally introduced the term trust management [5], its successors KeyNote and KeyNote 2 [3], and in some respects, SPKI [12]. In this paper, we use the KeyNote 2 system. Our main motivation was the availability of a reasonably good implementation written by Angelos Keromytis. This library is already used in several other applications, including OpenBSD's ISAKMP daemon [6, 14].

Other examples where trust management has been successfully used include maintaining distributed firewall rules [15], controlling access rights in Jini and CORBA [13, 18], and managing authorization for Java applets [20].

4.2 Trust management for dynamic updates

KeyNote is a simple and flexible trust management system designed to be suitable for a wide range of applications that need to make access control decisions in a distributed environment.

The KeyNote trust management system has four basic components. One is a language for describing "actions", i.e., the operations that need access control. It also has a mechanism for identifying "principals", the entities that can be authorized to perform actions. Further, it has a language for specifying "assertions" which define what actions the different principals should be allowed to do. Finally, it has a "compliance checker" which determines if an action requested by a principal should be allowed or denied, given a policy and a set of credentials. [3]

Field	Description	Example
What is being modified		
zone	zone name	example.com
name	fully-qualified name	saturn.example.com
type	RR set type as text, or ANY	A
ttl	time-to-live in seconds	3600
rdata	record data, in zone file format (may be empty)	192.168.1.2
operation	add or delete	
Who is modifying		
client_ip	client IP address	192.168.3.4
client_protocol	TCP or UDP	
client_port	client TCP/UDP port number	
tsig_key_name	from named.conf key-section	dhcp1.example.org–ns1.example.org
Context		
app_domain	KeyNote domain	dyndns
time	current date and time	2000-10-18_17:15:33

Table 1: Attributes for dynamic update assertions

When using KeyNote for DNS dynamic updates, the principals are authenticated with standard DNSSEC transaction authentication mechanisms.

The basic structure of a KeyNote assertion consists of a maximum of seven fields: a Version number, an Authorizer, the Licensees of the credential, Local-Constants, Conditions, a Comment and a Signature. The Authorizer field is mandatory, and all other fields are optional. The Authorizer field specifies who is authorizing the principals specified in the Licensees field to perform an action if the action attributes meet the specified Conditions.

The Authorizer can be "POLICY"or a public key of a user. "POLICY" represents the root of the trust hierarchy, and is allowed to do anything. Because only public keys can be used to sign credentials, assertions whose Authorizer is "POLICY" must be stored locally to protect their integrity and authenticity. Together, these assertions are called the policy. The KeyNote specification allows the use of other non-cryptographic identifiers in addition to "POLICY", but we will not consider them here. If a public key is used, it can also be specified in the Local-Constants field and referenced by name in the Authorizer field. Assertions authorized and signed by a public key are called credentials.

The Licensees can be public keys, private keys, or IP addresses. The public keys can also be specified in the Local-Constants field and referenced by name in the Licensee field, just like in the Authorizer field. There can be more than one Licensee to an assertion.

The Conditions specify what requirements the action attributes must meet for the Licensee to be allowed to execute it. If there are no Conditions, all actions allowed to the Authorizer are also allowed to the Licensee. Examples of condition expressions are given in the next section.

The action attributes consist of a name and a value. KeyNote does not specify the semantics of the names and the values: they are particular to the application domain. A reserved attribute called "app_domain" should contain the name of the application domain of the particular assertion. Table 1 presents our proposal for the possible attributes of a KeyNote assertion for DNS dynamic updates.

When a dynamic DNS update request is made, the client should provide its credentials in the additional section of the request. The compliance checker gets the local policy assertions from a configuration file, and the requested action attributes and the client's credentials from the request. The credentials could also come from a zone, if the server is configured to fetch them from it. The compliance checker processes the request and the assertions, and returns either "reject" or "approve" based on them.

4.3 Examples

Here are a few sample scenarios which demonstrate the use of KeyNote assertions for DNS dynamic updates.

Example 1: A very simple example is a situation where a DHCP server residing in the same, protected and trusted network as the DNS server is allowed to dynamically update everything in the domain. IP addresses are used to authenticate the party requesting the update. The assertion is stored locally in the server, so no cryptography is needed. The following assertion allows a DHCP server at 192.168.10.1 to modify everything in the zone helsinki.example.com:

```
Authorizer: "POLICY"
Licensees: "IP:192.168.10.1"
Conditions: ((app_domain=="dyndns") &&
    (zone=="helsinki.example.com"));
```

As we stated earlier, IP addresses are not a strong authentication method. In environments where the network's

friendliness cannot be guaranteed, the transactions need to be authenticated with one of the other DNSSEC transaction authentication methods, such as TSIG or SIG(0).

Example 2: Let us consider an example where the local networks are not fully trusted. We also do not fully trust the parties who need to make updates to the zone, so we want to restrict the types of updates they are allowed to perform.

A company wants to allow a few mobile users to update their address records as they roam from one company subnet to another. They want to use the TSIG authentication method to spare the mobile terminals from performing heavy computations, so the users' secret keys need to be defined in the DNS configuration file named.conf. Here is the definition for a key called johng-key.

```
key johng-key {
  algorithm "hmac-md5";
  secret "VGhpc0lzQVZlcnlCYWRLZXk=";
};
```

The following assertion allows anyone knowing the secret key johng-key to update the address record of "johng.helsinki.example.com", as long as the address is in the subnet 192.168.150/24. The assertion is also stored locally, so it does not need to be signed.

```
Authorizer: "POLICY"
Licensees: "TSIG:johng-key"
Conditions: ((app_domain=="dyndns") &&
    (zone=="helsinki.example.com") &&
    (name=="johng.helsinki.example.com") &&
    (type=="A") &&
    ((operation =="delete") ||
     ((operation=="add") &&
      (rdata~="^192\\.168\\.150\\.[0-9]+$")));
```

Example 3: Now Alice is the owner of the DNS server at a big ISP hosting hundreds of zones. Some of Alice's clients want to be able to easily update data in their own zones. Alice is obviously not going to take the risk of letting its clients modify the configuration files of her name servers. In fact, she would rather herself edit the configuration files as little as possible. To allow for remote configuration of the update policy, Alice delegates all update rights to her public key.

```
Authorizer: "POLICY"
Local-Constants: Alice="dsa-hex:712d8c12fbae..."
Licensees: Alice
Conditions: (app_domain=="dyndns");
```

Example, Inc. is one of Alice's clients who wants to be able to administer data in its own zone. Alice authorizes Bob, the DNS admin of Example, Inc. to update their zone.

```
Local-Constants: Alice="dsa-hex:712d8c12fbae..."
    Bob="rsa-hex:ef4289028181..."
Authorizer: Alice
Licensees: Bob
Conditions: ((app_domain=="dyndns") &&
    (zone=="example.com") &&
    (time<="2001-05-31"));
Signature: "sig-dsa-sha1-hex:23c49ac9a0be..."
```

This credential is signed with Alice's key, and is valid until end of May 2001. Because the credential is protected with Alice's signature, it does not have to be stored locally in the DNS server. Alice can send it to Bob, and Bob presents it every time he requests an update.

Now Bob can write another credential to Charlie, the administrator at Example Inc's Helsinki office. This credential gives Charlie the right to update everything under helsinki.example.com, as long as the request comes from the subnet 192.168.150/24. Note that Bob does not need to bother Alice with this.

```
Local-Constants: Bob="rsa-hex:ef4289028181..."
       Charlie="rsa-hex:ddd16090b43c..."
Authorizer: Bob
Licensees: Charlie
Conditions: ((app_domain=="dyndns") &&
    (zone=="helsinki.example.com") &&
    (client_ip~="^192\\.168\\.150\\.[0-9]+$") &&
    (time<="2001-03-31"));
Signature: "sig-rsa-md5-hex:05c76f03451f..."
```

When Charlie requests an update, he attaches the credentials delegating the access right from Alice to Bob and from Bob to himself in the message's additional section, and signs the message using his private key. The name server verifies the credential chain delegating the authority to perform the update from POLICY to Alice to Bob to Charlie, and makes the access control decisions based on that chain.

The details of how the KeyNote library actually verifies the certificates and performs the access control decision are described in [3].

5 Implementation

We have modified the BIND 9 name server to use the approach described in the previous section. For trust management, we used the KeyNote C library written by Angelos Keromytis. The library is also used in e.g. OpenBSD's ISAKMP implementation [6, 14].

The modifications consist of three major parts. First, we added support for "anonymous" SIG(0) signatures. Second, we added a way of specifying and loading the local security policy, and third, added calls to the KeyNote library to check authorization before performing an update. The details of these modifications are described in the next sections. Together, a little over 1000 lines of C were added.

5.1 "Anonymous" authentication

The SIG(0) record contains a DNS name pointing to the KEY record where the corresponding public key can be found. Currently BIND requires that this KEY record is stored in a local zone, i.e. a zone which the server is authorative for. This makes sense, because only the name of the KEY record can be used in an access control list, so the

binding between the key and the name obviously needs to be secure.

However, when using decentralized trust management for making an access control decision, the server really does not need to know who owns a particular key. We modified BIND so that the client can include the relevant KEY record in the message's additional section. This could be called "anonymous authentication" since we gain no information about who the key belongs to.

5.2 Policy specification

Due to the syntax of the BIND configuration file and KeyNote assertions, specifying KeyNote assertions inside the main named.conf file seemed difficult. Thus, we added a configuration statement "keynote-policy-file", which specifies the path of the local policy file. The file is a plain text file, containing KeyNote assertions separated by blank lines. The file is read whenever the server configuration is loaded (at startup, or after "rndc reload"). The assertions in the local configuration do not have to be signed.

5.3 Authorization checking

When a dynamic update is performed, the attributes described in Table 1 are given to the KeyNote engine, in addition to any authentication information. Also, the contents of any CERT records [11] in the additional section are used as assertions. The assertions must naturally be signed to be valid.

5.4 Test client

To test our modification to BIND, we also wrote a client program which sends dynamic update requests. The client was written in Java, and uses Brian Wellington's dnsjava library [25]. Writing the client was quite simple since the library already contained most of the required functionality. The only thing missing was support for SIG(0) signatures. The amount of Java code written or modified was less than 200 lines.

The signed assertions themselves can be generated using the "keynote" tool included with the KeyNote library. In real world, of course, better user interfaces would be needed.

6 Evaluation and future work

We think the main benefit of our proposal is the possibility to separate the server administration from update authorization. Trust management policies are easy to distribute across networks without the need to change local configuration files in the servers. Each server can have its own policy, but the management can still be done in a centralized manner. If public keys are used for authentication, the policy management tasks can even be easily and securely delegated to those parts of the organization where they naturally belong. This approach to policy management scales well even for large environments.

There are also clear benefits in using generic security components instead of application-specific ones. There is no need for every developer to (badly) re-invent the wheel. Consequently, when there are a lot of applications sharing the development cost, the security components can be created, evaluated and tested more carefully. The need to accommodate more than one application automatically results in a syntax which is more flexible than application-specific ACLs. However, this may also create risks: if the developer is not careful in using the component, the system may become too flexible, allowing users to create configurations that make no sense. In our implementation, we have tried to avoid this risk.

6.1 Trust infrastructure

If our solution is to be used in a truly decentralized manner, a mechanism for distributing the certificates to the clients is required. This can, of course, be done partially manually; i.e. when a user requests access rights from an administrator, the administrator can send a complete set of credentials required in response.

This requires the client to store all the required credentials. An alternative solution would be store the credentials on the server. Although they could be stored in a configuration file, a better idea is to store them in CERT records in some particular zone. This zone does not need to be protected as well as an access control list, because all the credentials are signed. Our implementation does not support this yet.

Depending on the situation, an infrastructure for revoking old credentials may be needed. KeyNote currently does not have any support for revocation. Revocation and validation schemes for another trust management system, SPKI, are discussed in e.g. [16].

6.2 Implementation

We found that the KeyNote library is actually quite usable. The only serious deficiency we found is that the library is not thread safe. Therefore, we had to make sure that only one "worker" thread is used. Usually BIND 9 uses one worker thread per CPU, so this is a problem only on multiprocessor machines.

Our implementation added the security functionality directly to the BIND source code. This leads to the coupling of application and security code, which can be undesirable, because the application becomes part of the trusted environment. Also, it has been argued that BIND is too large as it is, and should be split into separate components. One of these components could be a "update gatekeeper", which

would check the signatures and credentials, and forward the messages to the real server if the operation is allowed. This gatekeeper could be independent of the DNS server software used, and could be used even when the server's source code is not available. There are advantages and disadvantages in both approaches, and further work is needed to find out which one is actually better.

Our implementation is still by nature a research prototype. Optimizations and extensive testing would be needed before it could be used in a production environment. Some tests with real users and DNS administrators should also be made to determine the usability of the solution. After all, the concepts of trust management might prove to be hard to understand for people who are used to thinking in terms of ACLs.

6.3 Denial of service attacks

When sending a request to the server, the server has to check the signature of the message and any certificates included. This check usually takes about the same time whether the signature is valid or not. By sending a very large number of messages (possibly with a large number of certificates as well), it is possible to consume lots of CPU time, possibly leading to a denial of service condition.

We have not yet performed any measurements on how many messages per second would be needed for this attack. This concern applies to standard BIND 9 as well, since it also checks the signatures on messages, but much more messages would be required.

This is really a problem in the protocol design; future protocols should implement some protection against denial of service [1, 19].

6.4 Access decisions based on the zone data

Our implementation currently has a limitation in its ability to express one category of real world constraints to access rights. Currently, existing zone data is not used when performing access control decisions. For example, if a user is allowed to delete only address (A) records, and attempts to delete all records for a given name, the request is denied even if only address records exist for that name. To give another example, there is currently no way to specify that a user is allowed to add a record only if no record with that name already exists. The user may specify these conditions in the request as preconditions to the update, but there is no way of enforcing them as a policy by the server.

Implementing this in KeyNote could be difficult, because while the condition expressions are quite flexible, they are not a Turing-complete programming language—unlike in PolicyMaker, for instance [5]. We intend to further investigate this problem.

7 Conclusions

DNS dynamic updates are a useful way to modify DNS zone data, especially in environments where a part of the data changes often. The DNS security extensions define a reasonable variety of mechanisms for authenticating the source of update requests. However, we find that the available methods for authorizing dynamic updates and verifying access rights introduce a number of problems.

We propose a new solution for authorizing DNS dynamic updates. By combining the existing facilities for authenticating update requests with state-of-the-art authorization mechanisms, we have created a solution which is more flexible and scalable than the existing approaches. We hope this allows more widespread use of DNS dynamic updates.

We have implemented the proposed solution using the BIND 9 name server and the KeyNote C library. While the implementation is only a prototype and needs further work, it has shown us that the idea works.

Acknowledgements

We would like to thank Angelos Keromytis, Alexander Krotov, Ilkka Tuohela, and Camillo Särs for their comments on earlier versions of this paper.

References

[1] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant authentication with client puzzles. In Bruce Christianson, Bruno Crispo, and Mike Roe, editors, *Proceedings of the 8th International Workshop on Security Protocols*, to appear in the Lecture Notes in Computer Science series, Cambridge, UK, April 2000. Springer.

[2] Steven M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the 5th USENIX UNIX Security Symposium*, Salt Lake City, Utah, June 1995.

[3] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system version 2. RFC 2704, IETF, September 1999.

[4] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In Jan Bosch, Jan Vitek, and Christian D. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, Lecture Notes in Computer Science volume 1603, pages 185–210. Springer, 1999.

[5] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of*

the 1996 IEEE Symposium on Security and Privacy, pages 164–173, Oakland, California, May 1996.

[6] Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Trust management for IPsec. To appear in *Proceedings of the Network and Distributed System Security Symposium (NDSS 2001)*, San Diego, California, February 2001.

[7] Donald E. Eastlake. Secure domain name system dynamic update. RFC 2137, IETF, April 1997.

[8] Donald E. Eastlake. Domain name system security extensions. RFC 2535, IETF, March 1999.

[9] Donald E. Eastlake. DNS request and transaction signatures (SIG(0)s). RFC 2931, IETF, September 2000.

[10] Donald E. Eastlake. Secret key establishment for DNS (TKEY RR). RFC 2930, IETF, September 2000.

[11] Donald E. Eastlake and Olafur Gudmundsson. Storing certificates in the domain name system (DNS). RFC 2538, IETF, March 1999.

[12] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylönen. SPKI certificate theory. RFC 2693, IETF, September 1999.

[13] Pasi Eronen and Pekka Nikander. Decentralized Jini security. To appear in *Proceedings of the Network and Distributed System Security Symposium (NDSS 2001)*, San Diego, California, February 2001.

[14] Niklas Hallqvist and Angelos D. Keromytis. Implementing internet key exchange (IKE). In *USENIX Annual 2000 Technical Conference — Freenix Refereed Track*, pages 201–214, San Diego, California, June 2000.

[15] Sotiris Ioannidis, Angelos D. Keromytis, Steve M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS 2000)*, pages 190–199, Athens, Greece, November 2000.

[16] Yki Kortesniemi, Tero Hasu, and Jonna Särs. A revocation, validation and authentication protocol for SPKI based delegation systems. In *Proceedings of the 2000 Network and Distributed System Security Symposium (NDSS 2000)*, pages 85–101, San Diego, California, February 2000.

[17] Stuart Kwan, Praerit Garg, James Gilroy, and Levon Esibov. GSS algorithm for TSIG (GSS-TSIG). Work in progress, Internet draft ietf-dnsext-gss-tsig-01, November 2000.

[18] Tuomo Lampinen. Using SPKI certificates for authorization in CORBA based distributed object-oriented systems. In *Proceedings of the 4th Nordic Workshop on Secure IT systems (NordSec '99)*, pages 61–81, Kista, Sweden, November 1999.

[19] Jussipekka Leiwo, Pekka Nikander, and Tuomas Aura. Towards network denial of service resistant protocols. In *Proceedings of the 15th International Information Security Conference (IFIP/SEC 2000)*, Beijing, China, August 2000. Kluwer.

[20] Pekka Nikander and Jonna Partanen. Distributed policy management for JDK 1.2. In *Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS '99)*, pages 91–101, San Diego, California, February 1999.

[21] Young-Chul Shim, Hee-Won Shim, Man-Hee Lee, and Ok-Hwan Byeon. Extension and design of secure dynamic updates in domain name system. In *Proceedings of 5th Asia-Pacific Conference on Communications and 4th Optoelectronics and Communications Conference (APCC/OECC '99)*, pages 1147–1150, Beijing, China, October 1999.

[22] Paul Vixie. DNS and BIND security issues. In *Proceedings of the 5th USENIX UNIX Security Symposium*, pages 209–216, Salt Lake City, Utah, June 1995.

[23] Paul Vixie, Olafur Gudmundsson, Donald E. Eastlake, and Brian Wellington. Secret key transaction authentication for DNS (TSIG). RFC 2845, IETF, May 2000.

[24] Paul Vixie, Susan Thompson, Yakov Rekhter, and Jim Bound. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, IETF, April 1997.

[25] Brian Wellington. Dnsjava home page. http://www.xbill.org/dnsjava/, August 2000.

[26] Brian Wellington. Secure domain name system (DNS) dynamic update. RFC 3007, IETF, November 2000.