社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＤＰＳ－117　(17)
2004－CSEC－　24　(17)
2004／3／4

アクセス制御機能付 DNS の実装と評価

馬場 達也　　日下 貴義　　山岡 正輝　　松田 栄之

株式会社 NTT データ 技術開発本部

〒104-0033 東京都中央区新川 1-21-2 茅場町タワー

E-mail:　{babatt, kusakat, yamaokam, matsudasg}@nttdata.co.jp

あらまし　DNS（Domain Name System）は、インターネットにおいて、ホスト名から IP アドレスへの変換を行うという重要な役割を担っているが、そのセキュリティ対策は十分であるとはいえない。これまでに、DNS にセキュリティ機能を加えるための拡張が議論されてきたが、DNS のデータに対するアクセス制御の仕組みはまだ十分に実現されていない。そこで、著者らは、DNS の問い合わせ元をネームサーバ側で認証し、その認証結果に基づいてリソースレコードの内容を返却するか否かを決定するアクセス制御機能付 DNS を提案してきた。本稿では、アクセス制御機能付 DNS を実装し、主に処理性能について評価した結果を報告する。

キーワード　DNS，アクセス制御，ネームサーバ，セキュリティ

Implementation and Estimation of a Domain Name System with Access Control

Tatsuya BABA, Takayoshi KUSAKA, Masaki YAMAOKA, and Shigeyuki MATSUDA

Research and Development Headquarters, NTT Data Corporation

Kayabacho Tower, 1-21-2, Shinkawa, Chuo-ku, Tokyo, 104-0033 Japan

E-mail:　{babatt, kusakat, yamaokam, matsudasg}@nttdata.co.jp

Abstract　DNS (Domain Name System) plays an important role in the Internet. It provides the mechanism for translating internet domain names for network hosts into IP addresses. As the Internet has grown to become a business infrastructure, security extensions to the DNS have been discussed and developed. However, any sort of access control lists or other means to differentiate inquirers are not provided in these extensions. Therefore, the authors have been proposed the access control mechanism for DNS, which authenticates inquirers. In this paper, we show the development of the DNS with access control and the estimation results.

Keyword　DNS, Access Control, Name Server, Security

1. はじめに

DNS（Domain Name System）[1, 2]は、インターネットにおいて、ホスト名を IP アドレスに変換する名前解決と呼ばれる重要な機能を提供しているだけでなく、メール配送やサービス検索などにも利用されており、インターネットのインフラともいうべき重要なシステムとなっている。しかし、インターネットは、元来、研究者間の通信基盤として発展してきたという経緯もあり、DNS におけるセキュリティ対策は十分であるとはいえない。

DNS におけるセキュリティ対策としては、これまでに、DNSSEC（DNS Security Extensions）[3]や TSIG（Transaction Signature）[4]、SIG(0)[5]などが提案されてきた。DNSSEC では、DNS のデータであるリソースレコードに対して、公開鍵暗号技術を使用して署名を施すことにより、データの改ざんを防ぐことを可能としている。また、TSIG および SIG(0)では、ネットワーク上を流れる DNS メッセージに MAC（Message Authentication Code）やデジタル署名を付加することによって、ネットワーク上での DNS メッセージの改ざんや、送信者のなりすましを防ぐことを可能としている。

しかし、DNS への問い合わせに対するアクセス制御の機能は、これらの技術では提供されていない。これは、DNS のデータは公開されるべきものという考えの下で設計されてきたため、アクセス制御に対するユーザの要求はあるものの、議論の対象外とされてきたためである。しかし、ホストにアクセスする前の DNS の名前解決の段階でアクセス制御を行うことができれば、従来のホストやファイアウォールでのアクセス制御に加えて、さらにセキュリティを高めることができ

ると考えられる。また、年々増加しているサービス妨害攻撃（DoS 攻撃）では、送信元を偽造した大量のパケットを送りつけることにより、ターゲットのネットワークやホストに負荷をかけることを目的としており、このような攻撃はホストやファイアウォールでのアクセス制御では防ぐことができない。しかし、DNS の名前解決の段階で、アクセスを許可するユーザのみに IP アドレスを開示するようなアクセス制御を行うことができれば、これらの攻撃をある程度防ぐことができると考えられる。

そこで、著者らは、DNS におけるアクセス制御を実現する仕組みとして、アクセス制御対応ネームサーバ（AC ネームサーバ）がアクセス制御対応クライアント（AC クライアント）上のユーザを SIG(0)の仕組みを利用して認証し、その認証結果に従ってアクセス制御を行う方式を提案している[6]。本稿では、提案した方式をプロトタイプとして実装し、主に処理性能について評価した結果を報告する。

2. DNS アクセス制御方式

最初に、これまでに提案した DNS におけるアクセス制御方式について説明する。

2.1. DNS アクセス制御の考え方

（1）アクセス元の識別および認証

本方式では、AC ネームサーバが、AC クライアントまたは AC クライアント上のユーザを識別および認証するために、SIG(0)を使用する。DNS メッセージに付加される SIG(0)リソースレコード（SIG(0) RR）には、問い合わせ元ユーザが生成したデジタル署名とドメイン名形式の署名者 ID が含まれているため、ネームサーバは、署名者 ID を基に問い合わせ元ユーザを識別し、署名を検証することでアクセス元を認証することができる。SIG(0)では、通常は、署名者 ID として FQDN（Fully Qualified Domain Name）で記述されたホスト名を使用するが、ユーザ ID を「baba._user.example.com」のように表記することで、リソースレコードへのアクセスをユーザ単位で制御できるようにする。

（2）データの機密性確保

DNS メッセージを暗号化するための仕組みとして、著者らが新たに提案した TENC リソースレコード（TENC RR：Transaction Encryption Resource Record）を使用する。TENC では、オリジナルの DNS メッセージ全体を、共通鍵暗号を使用して暗号化し、暗号化したメッセージを新たに作成した DNS メッセージ中の TENC RR のデータ部に挿入して送信する（図 1）。これにより、DNS メッセージに含まれるリソースレコードだけでなく、DNS ヘッダも暗号化されるため、返却したエラーの種類やリソースレコードの数なども第三者に知られないようにすることが可能となる。

また、TENC で使用する暗号化用の秘密鍵を AC クライアントと AC ネームサーバとの間で共有するために、もともとは、TSIG で使用する認証鍵を共有するために標準化された鍵交換の仕組みである TKEY（Transaction Key）[7]を拡張したものを使用する。TKEY では、TSIG で使用する MAC アルゴリズムの ID を指定することで、生成する認証鍵の種類を相手に通知する仕様となっている。そこで、このアルゴリズム ID に、TENC で使用する共通鍵暗号アルゴリズムの ID を追加することによって、AC クライアントと AC ネームサーバとの間で暗号化用の秘密鍵を共有することができるようにする。

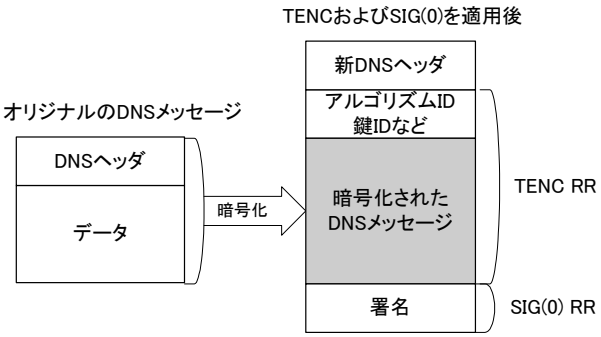

図 1 TENC による DNS メッセージの暗号化

（3）アクセス制御リストの記述

アクセス制御リスト（ACL）は、同じゾーンを管理する複数のネームサーバで共有されるように、ゾーンデータ内に TXT レコードを使用して記述する。これにより、ゾーン転送時に自動的に ACL も転送されるようになる。この TXT レコードでは、以下のように、「ACL」という文字に続いて、対象となるリソースレコードのタイプ、識別子（SIG(0) RR の署名者 ID で使用する FQDN）を記述することで、アクセスを許可するホストまたはユーザをリソースレコード毎に記述する。

```
www    IN  A     192.168.0.30
       IN  TXT   “ACL A baba._user.example.com.”
       IN  TXT   “ACL A kusaka._user.example.com.”
```

ただし、DNS のツリー構造をたどるために必要な SOA、NS、SIG、NXT、KEY、DS の各リソースレコードに対してはアクセス制御を行わず、すべての問い合わせに対して回答することとする。

2.2. プロトコルシーケンス

（1）通常名前解決

AC クライアントは、最初にローカルネームサーバ経由で通常のクライアントと同様の名前解決処理を行う。アクセス先のリソースレコードがアクセス制御されていない場合は、ここで回答が返却される。

（2）アクセス先 AC ネームサーバの探索

通常名前解決で回答が得られなかった場合は、アクセス制御対応名前解決を行う。アクセス制御対応名前解決では、AC クライアントは、AC ネームサーバに直接問い合わせを発行するため、最初にアクセス先 AC ネームサーバの探索を行う。

アクセス先 AC ネームサーバの探索処理は図 2 のようになる。AC クライアントは、アクセス先ゾーンの NS レコードをローカルネームサーバ経由で問い合わせ、ゾーンを管理する AC ネームサーバのホスト名を取得する。そして、その AC ネームサーバのホスト名に対する A レコードをさらに問い合わせることにより、アクセス先ゾーンを管理している AC ネームサーバの IP アドレスを取得する。

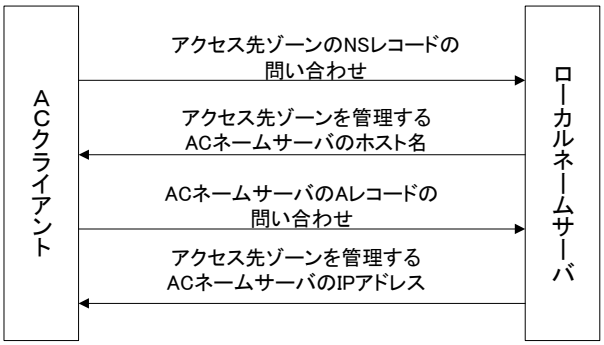

図 2 AC ネームサーバの探索シーケンス

（3）アクセス制御対応名前解決

アクセス先ゾーンを管理する AC ネームサーバの IP アドレスを取得したら、AC クライアントは、AC ネームサーバに対して、ローカルネームサーバを介さずに直接問い合わせを行う。

DNS メッセージを暗号化する場合は、図 3 のように、最初に AC クライアントと AC ネームサーバとの間で TKEY による暗号化用の秘密鍵のセットアップを行う。そして、TKEY によって生成された秘密鍵を使用して、DNS メッセージを暗号化し、その内容を TENC RR に挿入して送信する。ここで交換されるメッセージには、すべて、SIG(0) RR を付加して認証を行う。AC ネームサーバでは、SIG(0) RR の署名者 ID に対応した公開鍵を使用して、受信した DNS メッセージに付加されている署名を検証する。署名の検証に成功した場合は、該当するリソースレコードの ACL を参照し、その署名者に対してリソースレコードの内容を返却して良いかどうかを判断する。アクセスが許可されている場合には、回答に AC ネームサーバの秘密鍵で署名を施した SIG(0) RR を付加した DNS メッセージを AC クライアントに送信する。AC クライアントでは、受信した DNS メッセージに付加されている署名を、AC ネームサーバの公開鍵を使用して検証する。

DNS メッセージを暗号化しない場合は、鍵交換フェーズは存在せず、問い合わせメッセージに SIG(0) RR を付加したものを、AC ネームサーバに直接送信する。

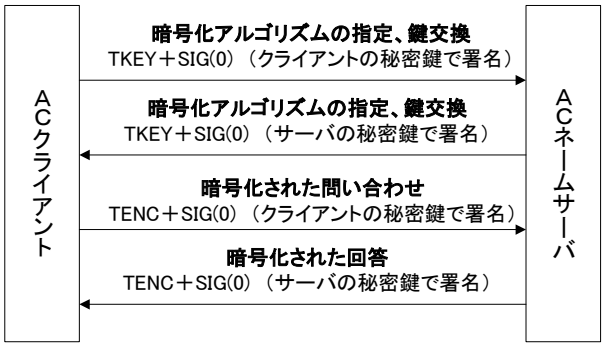

図 3 暗号化を有効にした場合のプロトコルシーケンス

3. アクセス制御機能付 DNS の実装

これまでに述べた AC ネームサーバおよび AC クライアントの機能を、プロトタイプとして実装した。

3.1. AC ネームサーバの実装

AC ネームサーバ機能は、BIND（BIND 9.2.1）を改造することで実現した。BIND には、SIG(0)が付加された問い合わせを受信し、検証する機能が実装されているが、SIG(0)を付加した回答を送出する機能は実装されていない。このため、BIND に、回答への SIG(0)付加処理を追加した。そして、暗号化鍵を生成できるように、BIND の TKEY 処理を改造し、さらに TENC 処理を追加した。TENC で使用する暗号化処理は OpenSSL 0.9.6b を利用した。

TENC で必要となる暗号化アルゴリズムや SIG(0)で必要となる秘密鍵は、BIND の設定ファイルである named.conf で指定できるようにした。

3.2. AC クライアントの実装

AC クライアント機能は、既存のアプリケーションを変更することなく、アクセス制御機能を利用できるようにするために、共通ライブラリである glibc（glibc 2.3.2）上に実装した。また、TENC で使用する暗号化

処理は OpenSSL 0.9.6b を利用した。

SIG(0)署名の生成に使用するユーザの秘密鍵は、別のユーザがアクセスできないように、各ユーザのホームディレクトリの.sdns ディレクトリ配下に置くようにした。

3.3. 公開鍵の管理方式

本方式では、AC クライアントおよび AC ネームサーバ間で交換されるメッセージに付加される SIG(0)署名を検証するために、相手の公開鍵が必要となる。この公開鍵は、事前に取得して保持しておいてもよいが、それでは、スケーラビリティの面で問題となる。そこで、本実装では、クライアントユーザおよび AC ネームサーバの公開鍵をあらかじめ KEY レコードとして DNS に登録しておき、SIG(0)署名の検証時に、SIG(0)に含まれている ID から KEY レコードを DNS に問い合わせることで、公開鍵を動的に取得できるようにした。

3.4. AC ネームサーバの探索方式

アクセス先 AC ネームサーバの探索処理では、AC クライアントから、アクセス先ゾーンに対する NS レコードを DNS に問い合わせる。しかし、実際は、名前解決しようとしているドメイン名から、そのリソースレコードを管理しているゾーンの名称を割り出すことは難しい。例えば、www.foo.example.com の A レコードは、foo.example.com ゾーンで管理されているのか、あるいは、example.com ゾーンで管理されているのかを AC クライアント側で判別することはできない。そこで、本実装では、問い合わせ先ドメイン名のラベルを左からひとつずつ削って、その NS レコードを問い合わせるようにした。つまり、www.foo.example.com の A レコードを問い合わせた場合は、最初に、ラベルをひとつだけ削った foo.example.com に対して NS レコードを問い合わせ、エラーが返ってきた場合は、ラベルをさらにひとつ削った example.com に対して NS レコードを問い合わせるようにした。

4. 評価

実装したプロトタイプを使用して、アクセス制御に関わる処理時間およびメモリ使用量について測定した。

4.1. 評価方法

ACL を除いたリソースレコード数および名前解決を行う対象のリソースレコードに対する ACL 数を変化させて名前解決時間を測定した。測定は、暗号化を有効にした場合と無効にした場合に分けて 10 回ずつ行い、最大値および最小値を除いて平均を算出した。また、暗号化を有効にした場合の AC ネームサーバの各フェーズの処理時間とメモリ使用量もあわせて測定した。

4.2. 評価環境

評価環境を図 4 に示す。クライアントと sdns.com ネームサーバには、アクセス制御対応のものと非対応のものを用意し、それぞれのクライアントから sdns.com ネームサーバの管理する”www.sdns.dom”の A レコードに対して名前解決を行った。この際に、クライアントユーザおよびサーバの公開鍵は、それぞれ、home.dom ネームサーバおよび sdns.dom ネームサーバに KEY レコードとして登録しておいた。

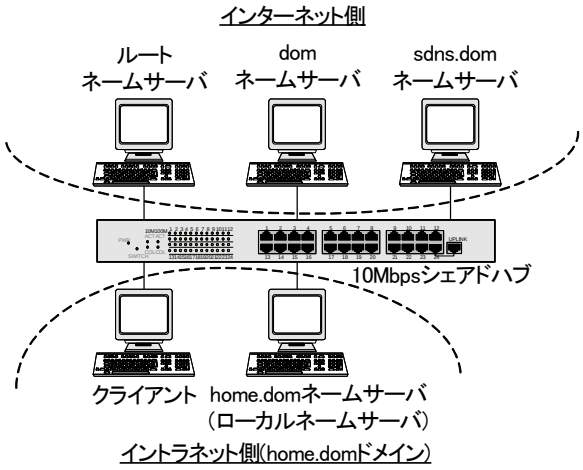

図 4 評価環境

評価に使用したマシンのスペックは表 1 のとおりである。なお、ルートネームサーバ、dom ネームサーバ、home.dom ネームサーバ(ローカルネームサーバ)には、改造を施していない、通常の BIND 9.2.1 を使用した。

表 1 評価用マシンのスペック

マシン	CPU	メモリ	OS
DELL OptiPlex GX260	Intel Pentium 4 2.80GHz	1024MB	Red Hat Linux 9

また、署名アルゴリズムには 768 ビットの DSA、データの暗号化アルゴリズムには 3DES-CBC、DH（Diffie-Hellman）鍵には 768 ビットのものを使用した。本来は、署名鍵および DH 鍵の長さは 1024 ビット以上のものを使用するべきであるが、DNS メッセージを UDP で送信する際の 512 バイトの限界を超えてしまうため、これらの鍵の長さを 768 ビットとした。この 512 バイトの限界は、クライアントとサーバの両方で EDNS0 をサポートすることで解決することが可能であるが、今回のプロトタイプでは対応していない。

4.3. 評価結果

ACL 数を 5 で固定し、ACL を除いたリソースレコード数を変化させた場合の名前解決時間は、図 5 のグラフのようになった。リソースレコード数を変化させても名前解決時間にはほとんど影響がなく、暗号化を無効にしている場合は約 0.09 秒、暗号化を有効にしている場合は約 0.13 秒であった。

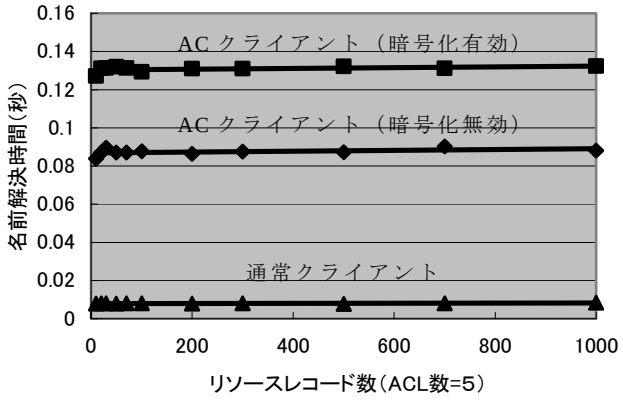

図 5 リソースレコード数と名前解決時間の関係

また、ACL を除いたリソースレコード数を 50 で固定し、ACL 数を変化させた場合の名前解決時間は、図 6 のグラフのようになった。ACL 数が増えるとともに名前解決時間もやや増加したが、同一のアクセス制御対象リソースレコードに ACL を 100 付与した場合でも、暗号化を無効にしている場合で 0.10 秒未満、暗号化を有効にしている場合で 0.14 秒未満であった。

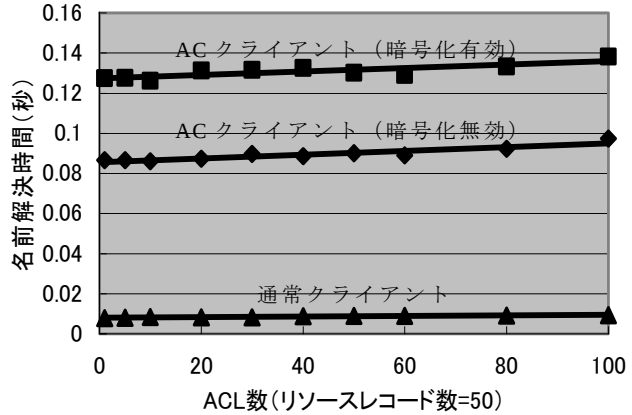

図 6 ACL 数と名前解決時間の関係

また、暗号化を有効にした場合の AC ネームサーバの各フェーズの処理時間の平均は表 3 のようになった。ACL 数を変化させた場合は、通常名前解決フェーズの A レコード回答処理とアクセス制御対応名前解決フェーズの A レコード回答処理で処理時間に変化があったが、それ以外の処理には変化が見られなかった。

表 3 AC ネームサーバでの処理時間

処理内容	処理時間（秒）
通常名前解決フェーズ	
Aレコード回答処理 （アクセス制御判定処理）	0.001～0.003 （ACL=1～100）
ACネームサーバ探索フェーズ	
NSレコード回答処理	0.001
Aレコード回答処理	0.001
鍵交換フェーズ	
ユーザ公開鍵取得処理	0.027
リクエストSIG(0)検証処理	0.002
DH処理および鍵生成処理	0.007
レスポンスSIG(0)署名処理	0.002
アクセス制御対応名前解決フェーズ	
クエリSIG(0)検証処理	0.002
クエリ復号化処理	0.001
Aレコード回答処理 （アクセス制御判定処理）	0.001～0.003 （ACL=1～100）
レスポンス暗号化処理	0.001
レスポンスSIG(0)署名処理	0.002
計	0.048～0.052

AC ネームサーバのメモリ使用量については、ACL の数ではなく、ACL を含んだ総リソースレコード数が影響した。ACL を含んだ総リソースレコード数を変化させた場合のメモリ使用量の変化は図 7 のようになった。

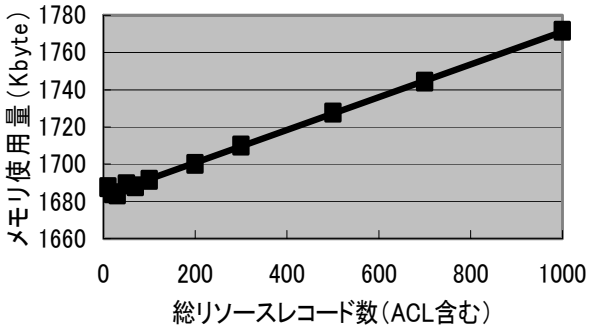

図 7 総リソースレコード数とメモリ使用量の関係

5. 考察

5.1. 名前解決時間

評価結果から、名前解決時間は、AC ネームサーバに登録されているリソースレコード数を増加させてもほとんど影響がないことがわかった。しかし、ACL の数を増加させた場合には、名前解決時間がわずかであるが増加した。これは、アクセス元のユーザ名と ACL 中のユーザ名を比較する処理に時間がかかっていることが主な原因である。ただし、暗号化を有効にし、あるリソースレコードに対して 100 の ACL を付与した場合でも、名前解決時間は 0.14 秒未満であり、この程度

であれば、名前解決を行う実際のアプリケーションの利用への影響はないと考えられる。

5.2. AC ネームサーバの性能要件

暗号化を有効にした場合の AC ネームサーバでの処理時間の合計は、ACL 数を 1 から 100 まで変化させた場合で、0.048～0.052 秒となった。ただし、AC ネームサーバのユーザ公開鍵取得処理の中には、外部ネームサーバからの返答を待っている時間も含まれているため、実際に CPU が動作している時間は、1 クエリあたり 0.024～0.028 秒程度であると考えられる。つまり、今回の評価で使用したスペック（Pentium 4 2.80GHz）と同等の PC を AC ネームサーバに使用した場合は、アクセス制御されているリソースレコードへの問い合わせを、1 秒間に 40 クエリ程度処理することが可能となる。また、AC ネームサーバは、通常のゾーン転送で ACL も同時に転送される仕組みになっているため、アクセス制御機能に対応したセカンダリネームサーバを増やすことで容易に負荷分散を行うことが可能である。適用する環境のアクセス数に合わせて、AC ネームサーバの CPU 性能および台数を調整することで、本方式を問題なく適用することができると考えられる。

AC ネームサーバが使用するメモリ量は、通常の BIND と比較して大きな違いは見られなかった。ACL を含めた総リソースレコード数を変化させた場合は、1 リソースレコードあたり 90 バイト程度増加したが、1000 レコード登録した場合でも、メモリ使用量は 2M バイトにも満たない。このため、メモリ量に関しては特に問題ないと考えられる。

5.3. 既存ネームサーバへの影響

本方式では、ルートネームサーバや、jp ネームサーバなどの TLD（Top Level Domain）ネームサーバへの問い合わせは、多くの場合、最初の通常問い合わせ時と、サーバ側でのユーザ公開鍵取得時に発生するのみで、AC ネームサーバ探索処理などでは、キャッシュが使用される。このため、ルートネームサーバや TLD ネームサーバへの影響は少ないと考える。

また、AC クライアントから、既存のネームサーバが管理するリソースレコードに対して名前解決を行った場合には、通常名前解決フェーズで回答が返されるため、本方式による影響は発生しない。しかし、AC クライアントから、既存のネームサーバが管理するゾーンの存在しないリソースレコードに対して名前解決しようとした場合は、暗号化有効の場合は鍵交換フェーズ、暗号化無効の場合は、アクセス制御対応問い合わせフェーズまで行われてしまい、既存のネームサーバでは、3 クエリ分の処理が余分に生じてしまう。これは、既存のローカルネームサーバが対応できるように、アクセス制御対象のリソースレコードに通常問い合わせを行った場合の回答として、アクセス制御用の専用のエラーコードではなく、既存の「Non-Existent Domain」エラーを返すようにしているためである。これにより、AC クライアントは、存在しないリソースレコードの場合でも、アクセス制御されているリソースレコードの場合と同様の処理をしてしまう。この問題による影響は小さいものであるが、実用化の際には、新たに専用のエラーコードを定義し、ローカルネームサーバでアクセス制御用のエラーコードをクライアントに転送するようにする必要があると考える。

6. まとめ

DNS の問い合わせ元をホストまたはユーザ単位で認証し、認証結果に基づいてリソースレコードへのアクセス制御を行う方式をプロトタイプとして実装した。また、プロトタイプを使用して、主に性能面について評価した結果を報告した。評価の結果、既存の環境や性能に大きな影響を与えることなく、本方式が導入可能であることを示すことができた。

謝辞

本研究は、通信・放送機構（TAO）の委託研究テーマ「次世代 DNS に関する研究開発」の一環として行われているものである。

参　考　文　献

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

[2] P. Mockapetris, "Domain Names – Implementation and Specification", RFC 1035, November 1987.

[3] D. Eastlake 3rd, "Domain Name System Security Extensions", RFC 2535, March 1999.

[4] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[5] D. Eastlake 3rd, “DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.

[6] 馬場，日下，山岡，松田，“DNS におけるアクセス制御プロトコルの検討”，情報処理学会研究報告，2003-CSEC-20，Vol.2003，No.18，pp.173-178，February 2003.

[7] D. Eastlake 3rd, "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.