DNSSECチュートリアル

民田雅人
株式会社日本レジストリサービス
Internet Week 2009 - H3

目次

- DNS編
  - BIND 9の設定
  - DNSのIPv6対応
- DNSのリスク編
  - DNSキャッシュポイズニング
  - 従来型の毒入れ攻撃手法
  - 短いTTLのリスク
  - Kaminsky型の毒入れ攻撃手法
- DNSSEC編
  - DNSSECのしくみ
  - DNSSECの現状
- DNSSEC編(続き)
  - 電子署名とDNSSEC
  - DNSSECの鍵と信頼の連鎖
  - DNSSECのリソースレコード
  - BINDキャッシュサーバでのDNSSECの設定
  - BIND権威サーバでのDNSSECの設定
  - 鍵更新と再署名
  - DNSSEC化によるDNSデータの変化
  - DNSSEC関連技術
  - DNSSECのまとめ

BIND 9の設定

DNS

DNSサーバの違い

- 権威DNSサーバ(以下、権威サーバ)
  - ゾーン情報(＝パブリックな情報)を設定し、その情報を答える
  - **インターネット全体からの問合せに答える**
- キャッシュDNSサーバ(以下、キャッシュサーバ)
  - 自身では最小限の情報を持ち、必要な情報は権威サーバから検索して答える
  - 検索した情報をキャッシュし、次回以降の同じ問合せに備える
  - **サービス対象からの問合せのみに答える**
    ⇒ インターネット全体に答える必要は無い

BIND 9のnamed

- namedの1プロセスで権威サーバとキャッシュサーバを兼用できる
  - 多くのBIND(BIND 8以前を含む)が兼用で運用されている
  - キャッシュ・権威を分離できるがBIND運用では少数派
- 可能であれば、分離を検討する
  - サーバが1台でも分離可能

- **DNSSECには、BIND 9.6系以降が必須**(後述)
  - 本章の説明はBIND 9.4系以降を前提としている
  - 9.3までの設定ではサーバとしての動作に問題が発生する
  - 最新版は BIND 9.6.1-P1 だが基本的設定は同じ

キャッシュサーバ専用のBIND 9の設定例
自組織 192.0.2.0/24

```
// 自ネットワークの定義
acl mynet {
  192.0.2.0/24;
  localhost;
} ;

// グローバルオプションの設定
options {
  // キャッシュサーバ
  // BIND 9のデフォルトはyes
  recursion yes;
  // アクセス制限
  allow-query         { mynet; };
  allow-recursion     { mynet; };
  allow-query-cache   { mynet; };
  .....
};
```

```
// ルートサーバへのhint
zone "." {
  type hint;
  file "named.root";
};
```

- allow-queryで問合せを許可するIPアドレスを自ネットワークに限定する
- allow-recursion, allow-query-cacheも設定する
  - BIND 9.4から必須

権威サーバ専用のBIND 9の設定例

```
// グローバルオプションの設定
options {
  // 権威サーバの場合は no
  recursion no ;
  .....
  allow-transfer { none ; } ;
};

// root.hintの設定
zone "." {
  type hint ;
  file “root.hint" ;
} ;
```

```
// example.jp のマスタ設定
zone "example.jp" {
  type master ;
  file "example.jp.zone" ;
  allow-transfer
          { w.x.y.z ; } ;
} ;

// example2.jpのスレーブ設定
zone "example2.jp" {
  type slave ;
  file "bak/example2.jp.zone" ;
  masters { z.y.x.w ; } ;
} ;
```

兼用型BIND 9設定の基本

1. 問合せを許可するIPアドレスを限定する
   = キャッシュサーバにアクセス制限
   - allow-query; 問合せを許可するIPアドレスを指定する (デフォルトは全て)
   - allow-recursion; 再帰問合せ許可するIPアドレスを指定する (デフォルトはlocalhostとlocalnets)
   - allow-query-cache; キャッシュデータへのアクセスを許可するIPアドレスを指定する(デフォルトはlocalhostとlocalnets)
2. ゾーン情報はIPアドレスを限定しない
   = 権威サーバはアクセス制限しない
   - プライマリ・セカンダリはどちらも同じ

兼用型のBIND 9の設定例
自組織 192.0.2.0/24 viewを活用

```
// 自組織のネットワーク
acl mynet {
  192.0.2.0/24;
  localhost;
} ;

view "recursion" {
  match-clients { mynet ; };
  recursion    yes;
  zone "." {
    type hint ;
    file "named.root";
  };
};
```

```
view "external" {
  match-clients { any; };
  recursion no;
  zone "." {
    type hint ;
     file "named.root";
  };

  //  example.jp
  zone "example.jp" {
    type master ;
    file "master/example.jp" ;
  };
};
```

TIPS

- BIND 9ではviewを利用すると、サービス対象によってサーバの挙動を変更できる
  - 外部ネットワークからの問合せには、権威専用サーバとして応答
  - 内部ネットワークからの問合せには、キャッシュ・権威兼用サーバとして応答
  - 参考: Secure BIND Template
    http://www.cymru.com/Documents/secure-bind-template.html
- サーバが1台でも、複数のIPアドレスを設定すれば、キャッシュサーバと権威サーバは分離できる
  - listen-on, listen-on-v6 を設定

DNSのIPv6対応

DNS

DNSのIPv6対応は2つある

- DNS通信のIPv6対応
  - DNSの問合せ、応答のやりとりにIPv6の通信を使う
- DNSコンテンツ(ゾーンデータ)のIPv6対応
  - IPv6アドレス(AAAAリソースレコード(RR)) を登録する
  - IPv6の逆引きを登録する
- 2つは独立の事象
  - IPv4の通信を使ってIPv4アドレス(A RR)を検索
  - IPv4の通信を使ってIPv6アドレス(AAAA RR)を検索
  - IPv6の通信を使ってIPv4アドレス(A RR)を検索
  - IPv6の通信を使ってIPv6アドレス(AAAA RR)を検索

DNS通信のIPv6対応

- DNSサーバの実装が、IPv6での通信に対応しているかどうか
  - 比較的新しい実装は、ほとんどがIPv6での通信に対応
  - もちろん、サーバOSと、ネットワークでの対応も必要
- 権威サーバの実装
  - NSD、PowerDNS、BIND 9、BIND 8(8.4系)、etc…
- キャッシュサーバの実装
  - Unbound、PowerDNS recursor、BIND 9、BIND 8(8.4系)、etc…

BIND 9のIPv6関連の設定項目(抜粋)

- IPv6アドレスを直接記述できるもの
  ⇒ IPv4アドレスと併記できるもの
  - allow-query, allow-recursion, allow-query-cache
  - allow-notify, allow-transfer
  - match-destinations, match-clients
  - zone文内のmasters
- IPv6専用のオプションがあるもの
  ⇒ IPv4とは別に設定するもの
  - listen-on-v6　　listen-onのIPv6版
  - notify-source-v6　　notify-sourceのIPv6版

DNSコンテンツのIPv6対応

- 正引き　ドメイン名 ⇒ IPv6アドレス
  - AAAA RRを使う

```
www.example.jp.  IN AAAA  2001:DB8::1
```

- 逆引き　IPv6アドレス ⇒ ドメイン名
  - PTR RRを使う点はIPv4と同じ
  - 4bit単位で区切り、逆順に並べ最後に“ip6.arpa”
  - 2001:DB8::1
    ⇒ 2001:0DB8:0000:0000:0000:0000:0000:0001

```
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8
  .B.D.0.1.0.0.2.ip6.arpa.  IN PTR www.example.jp.
                                              (1行で記述)
```

  - 桁数が多いので記述ミスに注意

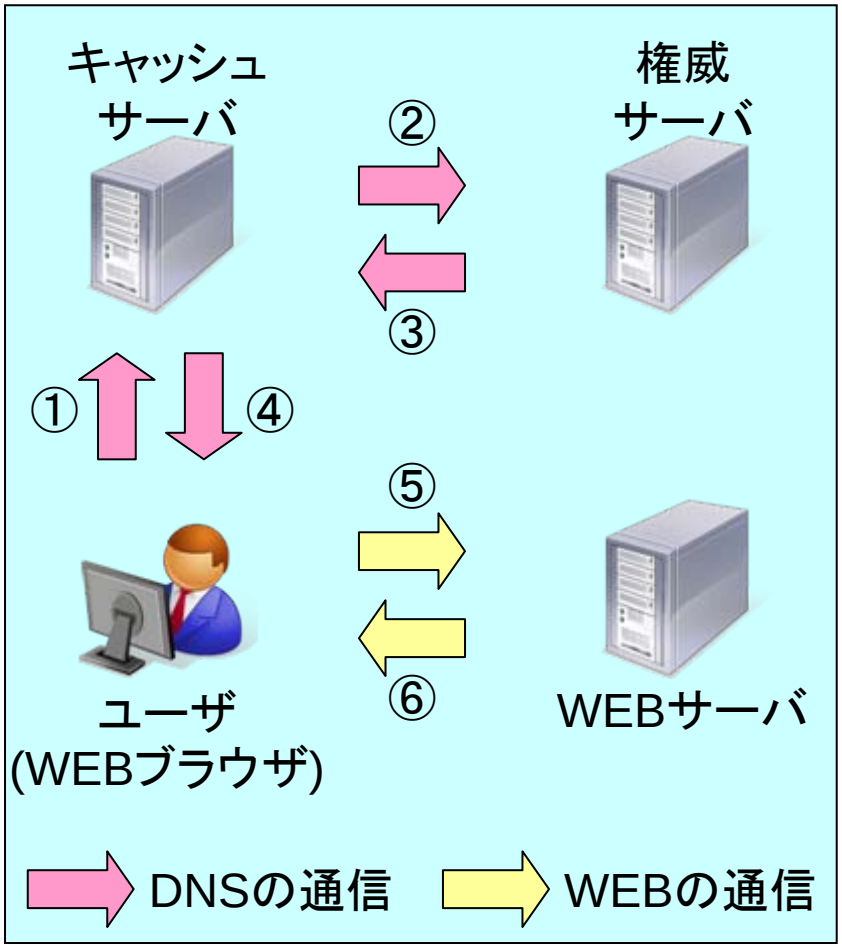
IPv6の逆引き記述のTIPS

- 人手で記述する場合、ミスを減らすため **digコマンドの-xオプション**を活用するのがベター
  - digコマンドは、DNS登録の有る無しに関わらず QUESTION SECTION をコメントで表示

```
$ dig -x 2001:db8::1
.....
;; QUESTION SECTION:
;1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. IN PTR
.....
```

- digの出力をコピー&ペーストして活用
  - hostコマンドでも可

IPv4/IPv6の混沌とした世界



- DNSはIPv4のみでも、WEBのAAAA RRがあれば、ユーザはWEBにIPv6でアクセス可(Google, 2ch等のIPv6)
  - 関係する全通信がIPv4/IPv6両対応とは限らない
- ①~⑥のIPv6/IPv4環境に、なんらかの問題がある
  - ユーザがWEBアクセスに時間がかかる、あるいはアクセス不能になることもある

トラブルシューティングを難しくする可能性がある

DNSのIPv6対応 まとめ

- DNSでは、通信とゾーンデータのIPv6対応がある
  - 最近の実装であれば、いずれも対応済
- 正引きの登録はAAAA RRを使う
- 逆引きは4bitづつで区切り最後に「ip6.arpa.」
  - 桁数が多いので記述ミスに注意
- WEBサーバーとPCはIPv6で通信していても、DNSはIPv4で通信していることもある
  - PC(WEBブラウザ) ⇔ キャッシュサーバ
  - キャッシュサーバ ⇔ 権威サーバ

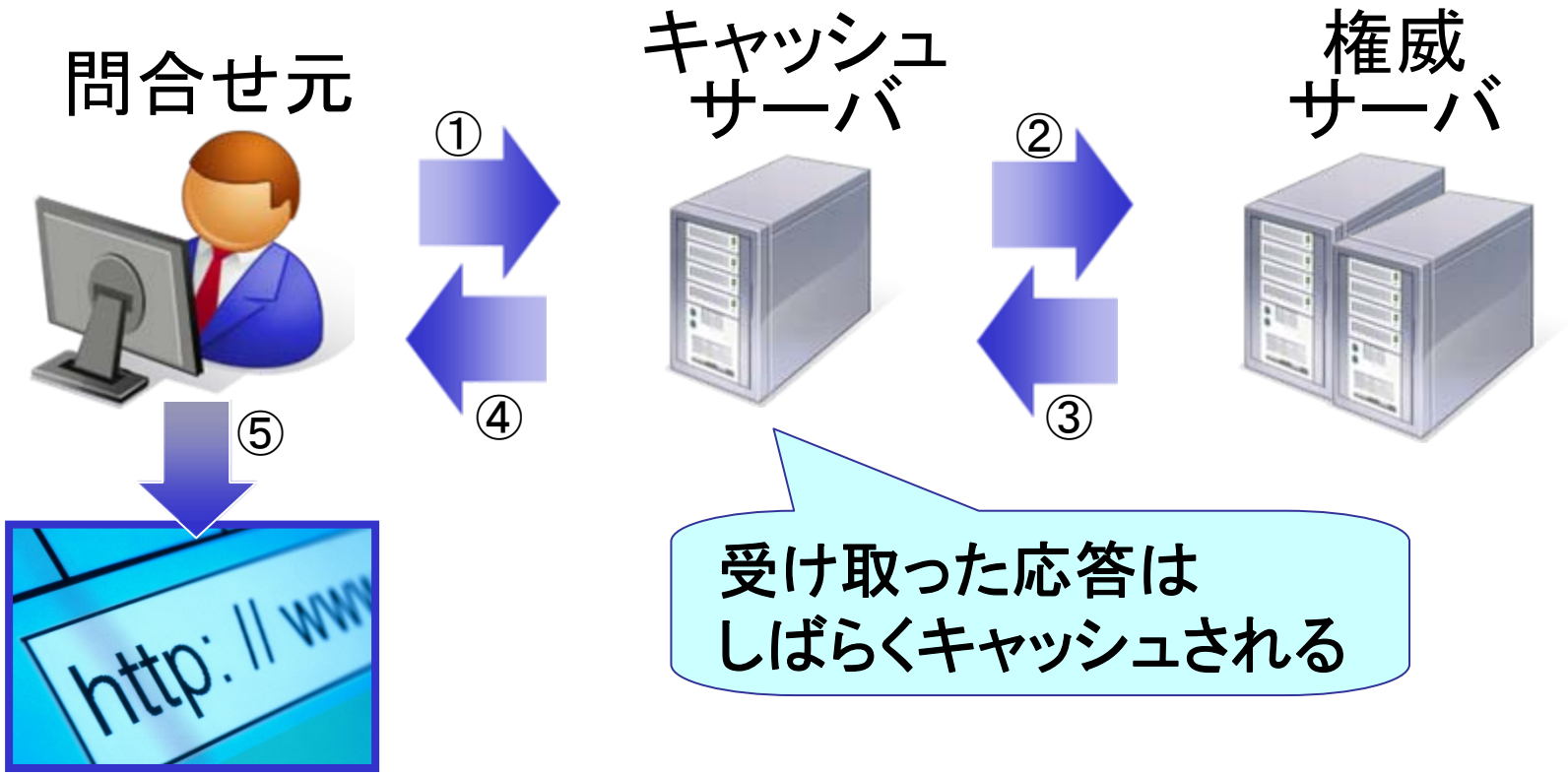
DNSキャッシュポイズニング (毒入れ)

DNSのリスク

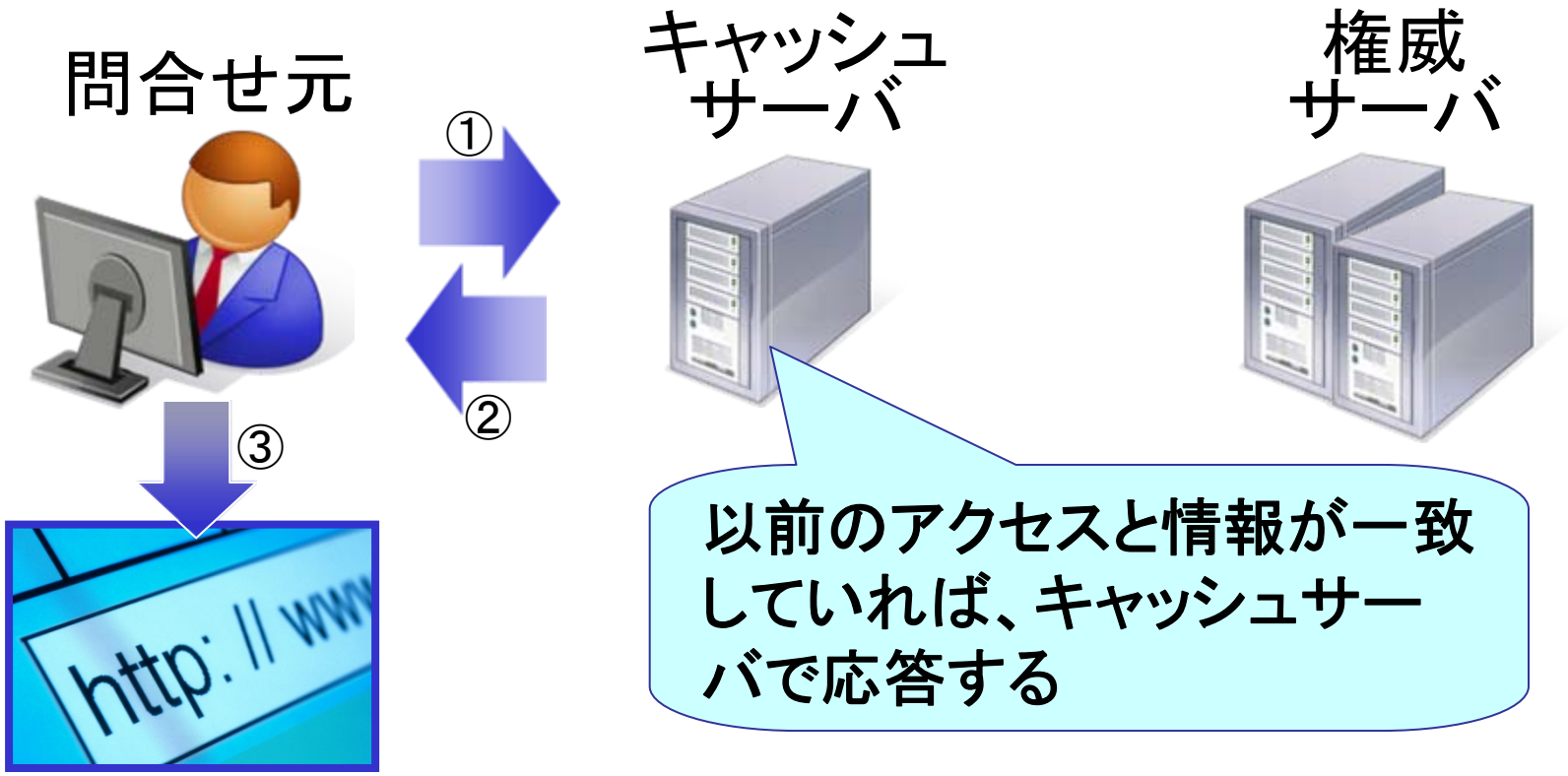
DNSキャッシュポイズニング (毒入れ)

- 予めキャッシュサーバに偽の情報を覚えこませ、ユーザが正しいアクセスを行ったつもりでも、偽装サイトへ誘導する手法
- フィッシングやファーミングの為の攻撃手法

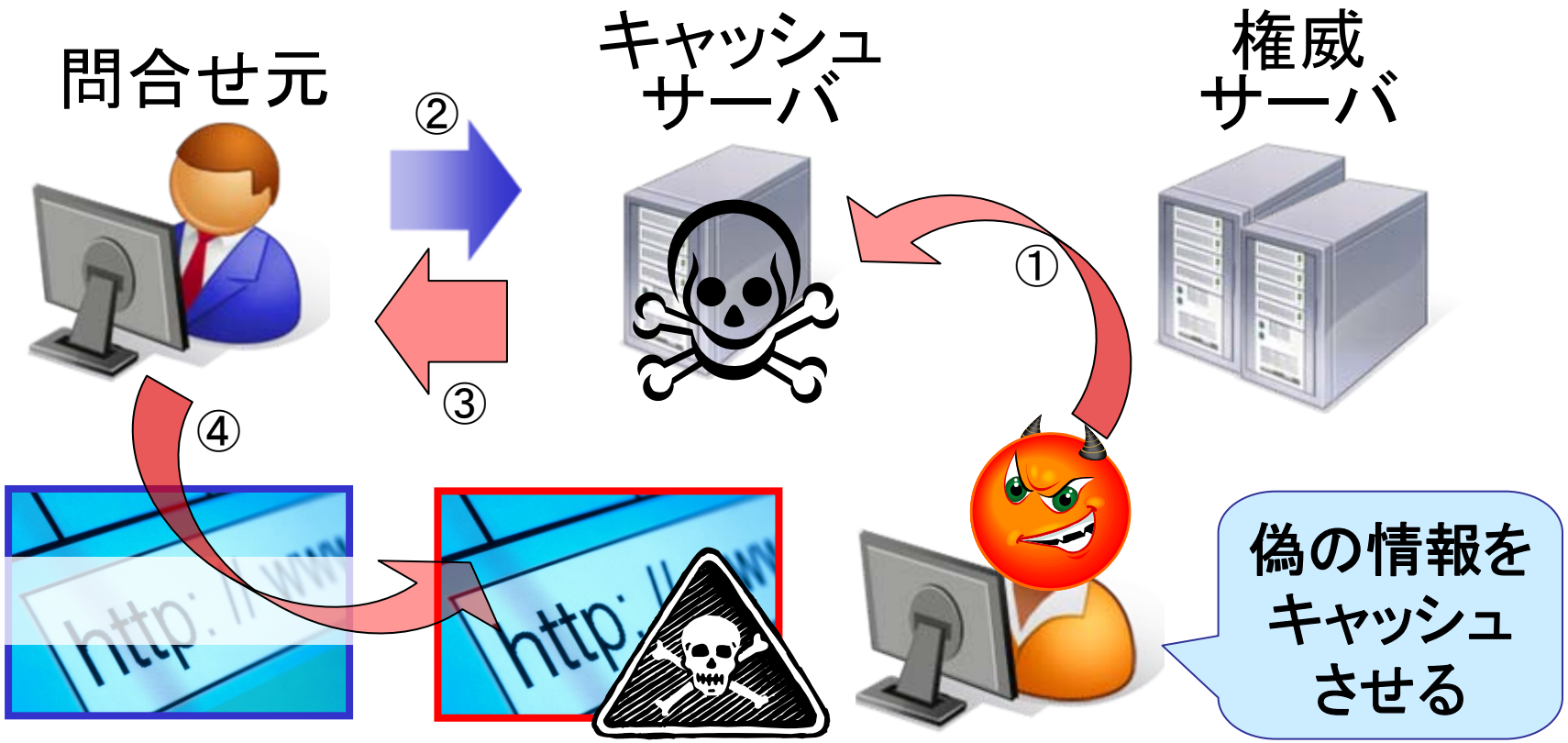
DNSの正常な流れ(1回目のアクセス)



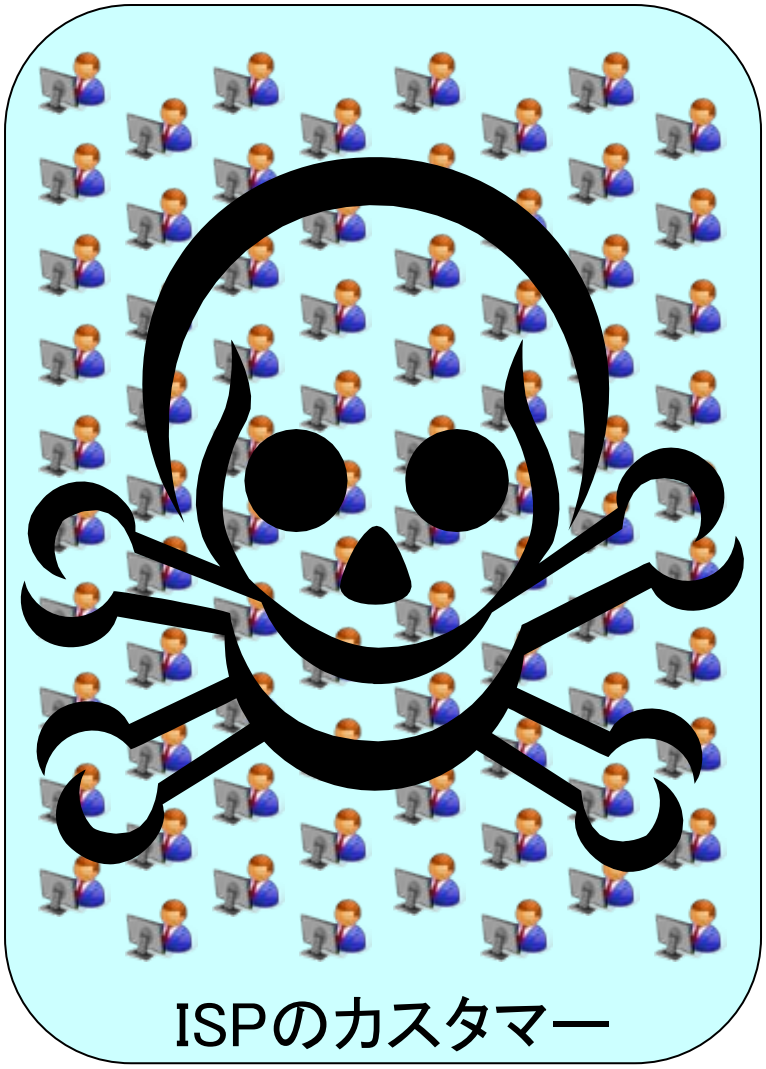
DNSの正常な流れ(2回目以降)



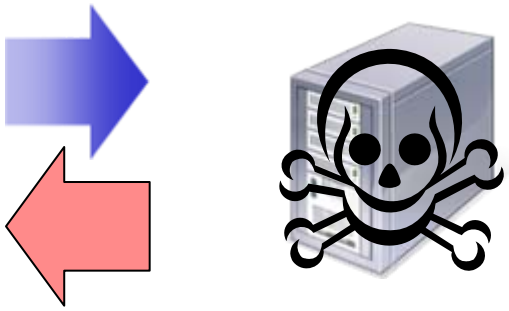
DNSへの毒入れ攻撃



ISPのキャッシュサーバが狙われたら



ISPのキャッシュサーバ



カスタマー全員が
被害に会う

DNS毒入れ攻撃の特徴

- ユーザが正常なアクセスを行っても、フィッシングサイトに誘導される
  - 攻撃されたことに気づきにくい
- 同じキャッシュサーバのユーザ全員が影響を受ける
  - 大手ISPのキャッシュサーバが攻撃されると被害は甚大
- 攻撃そのものの検出が容易ではない
  - キャッシュへの毒入れは、見た目は通常のDNSパケットであるため、正常な応答と攻撃の区別が簡単ではない

DNSへの毒入れの問題

- 1990年ごろには、DNSへの毒入れの問題が知られていた
  - 当時は設定が正しく行われていないためと考えられていた節もある
- 攻撃手法として知られるようになったのは1990年代後半のAlterNICの事件がきっかけ

DNSへの毒入れ攻撃手法の分類

- 従来型
  - Kashpureff型
  - 偽装応答型
- Kaminsky型
  - 2008年8月に公開された手法

注意:
「従来型」は、最近発見されたKaminsky型に対する
本資料での言葉であり、一般的なDNSの用語では無い

従来型の毒入れ攻撃手法

DNSのリスク

Kashpureff型による毒入れ

- 攻撃者が管理する権威サーバへ問合せさせ、正規の応答パケットに問合せ内容と関係ないドメインの情報を附加してキャッシュサーバへ送り込む手法
- 1997年7月の大規模DNS乗っ取り事件
  - http://www.internic.net/ へアクセスすると、http://www.alternic.net/ の内容が表示された
  - AlterNICのEugene Kashpureff氏によるもの

Kashpureff型の攻撃 example.jpでwww.jprs.co.jpの毒入れ

- example.jpゾーンの設定(一般の実装では不可能)

```
@                       IN NS    www.jprs.co.jp.
www.jprs.co.jp.            A     192.0.2.10
www                        A     192.0.2.1
```

- キャッシュサーバがwww.example.jpを検索

```
;; 回答セクション
www.example.jp.            A     192.0.2.1
;; 権威セクション
example.jp.                NS    www.jprs.co.jp.
;; 追加セクション
www.jprs.co.jp.            A     192.0.2.10
```

www.jprs.co.jpの嘘の値をキャッシュする

Kashpureff型攻撃の対策

- キャッシュサーバは、問合せたドメインのゾーン外のレコードがあったら、信用してはいけない
  - example.jpドメインの応答に、jprs.co.jpの情報があるのはそもそも怪しい

```
;; 回答セクション
www.example.jp.      A    192.0.2.1
;; 権威セクション
example.jp.          NS   www.jprs.co.jp.
;; 追加セクション
www.jprs.co.jp.      A    192.0.2.10 ← 信用してはいけない
```

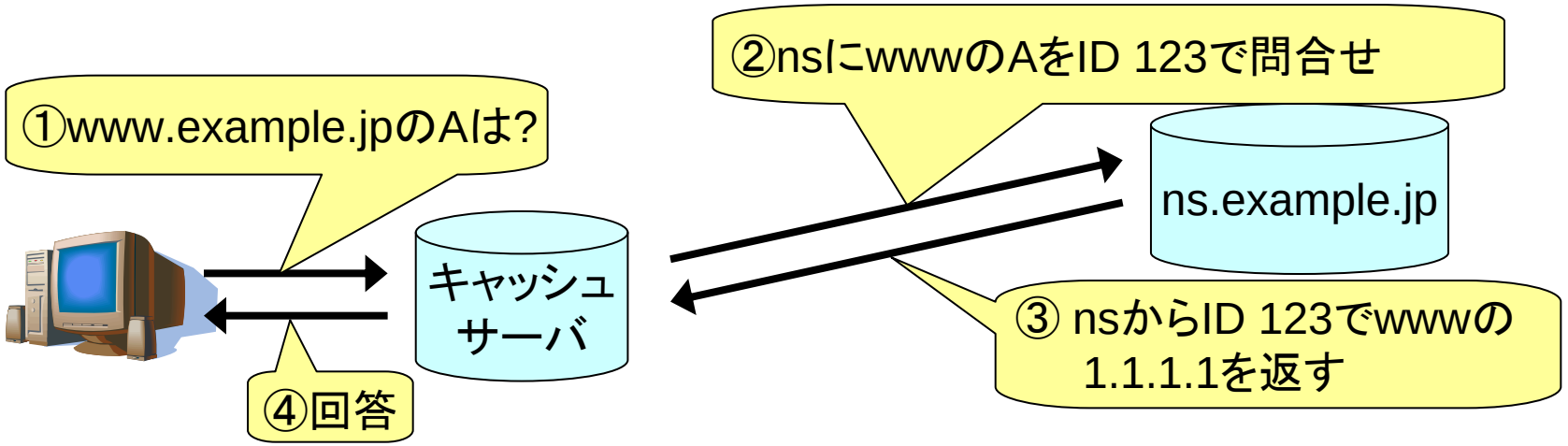
- BINDの場合4.9.6、8.1.1で対策が行われた
  - 権威サーバ側も対策が行われて設定できなくなった

DNSプロトコルのおさらい

- DNSの通信は問合せと応答の単純な往復
  - この名前のIPアドレスは?
    ⇒ IPアドレスはXXXXだよ
- トランスポートは主にUDP
  - 条件によってTCPになることもある
- 問合せパケット　　クエリ名+ID+etc...
- 応答パケット　　クエリ名+ID+回答+etc...

ID:　識別のための16bitの値

DNS応答パケットの識別



- 問合せに対応する応答を、ソースアドレス、クエリ名、IDで識別

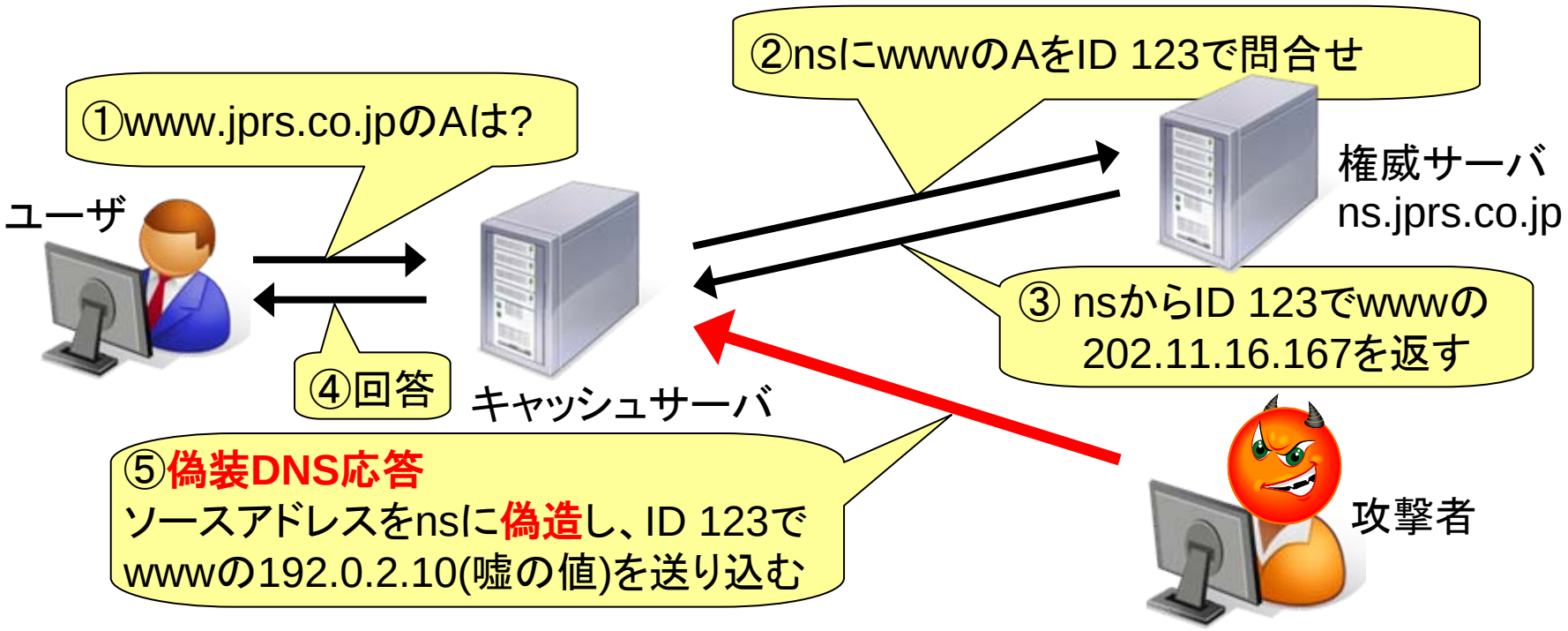
  例) ソースアドレス　　ns.example.jpのIPアドレス
  　　クエリ名　　www.example.jp
  　　ID　　123

- IDが違えば別の応答と判断
  - クライアントとキャッシュサーバ間(①と④)も同様

偽装応答型による毒入れ

- なんらかの手段を使い、本来の応答より先に偽装応答をキャッシュサーバに送り込み、偽情報をキャッシュさせる手法
  - 通常時DNSはUDPで通信するため、偽装応答が容易
- 攻撃手法
  - キャッシュサーバのDNS検索を盗聴し偽装応答を返す
  - キャッシュサーバに問合せを送り、IDを変化させた複数の偽装応答を返す(**オープンリゾルバは非常に危険**)
  - TTLの短いレコードを狙って、キャッシュサーバに偽装応答を送り続ける
  - etc...

偽装応答型の攻撃



- ③より先に⑤の偽DNS応答が送り込まれると、キャッシュサーバは嘘情報をキャッシュする
- ④で嘘情報をクライアントに送り、クライアントは偽のサイトへ誘導される

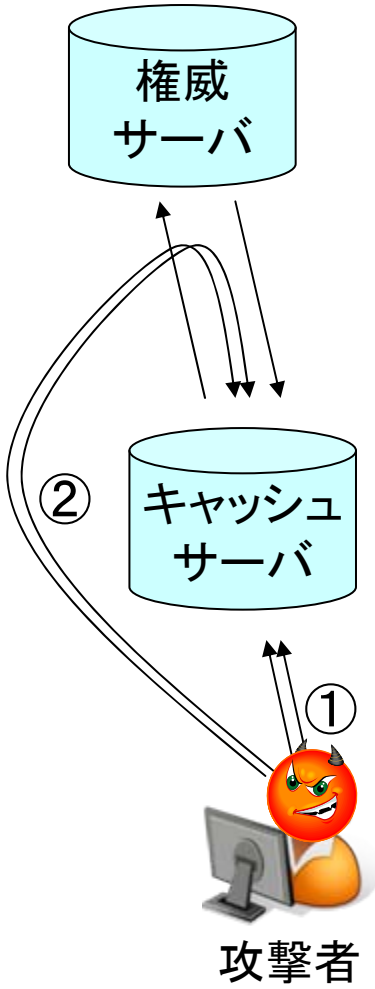
偽装応答型の攻撃手法

① オープンなキャッシュサーバに対して大量の問合せを送る

② 同じサーバに対して、偽装したDNS応答パケットを、IDをランダムに変えながら送る

– クエリ名とソースアドレスは自明

IDが正規応答と一致すれば攻撃が成功



偽装応答型の攻撃が成功する確率

問合せと応答のIDが一致すれば攻撃が成功

攻撃1回あたりの成功確率

$$P_S = \frac{R \times W}{N \times Port \times ID}$$

R: 攻撃対象1台あたりに送るパケット量(pps)
W: 攻撃可能な時間(Query⇒AnswerのRTT)
N: 攻撃対象レコードを保持する権威サーバの数
$Port$: キャッシュサーバのQuery portの数
ID: DNSのID (16bit = 65536)
(R 20000pps, W 10ms, N 2, $Port$ 1で 0.00152)

偽装応答型による攻撃の特徴

- 成功確率は決して低いとは言えない
- しかし、1度攻撃に失敗すると、キャッシュサーバが正規のレコードをキャッシュするため、連続した攻撃はできない
  - 攻撃に失敗した場合、次の攻撃まで、攻撃対象レコードのTTL時間待つ必要がある

短いTTLのリスク
(偽装応答型の応用)

DNSのリスク

短いTTLのリスク
偽装応答型の応用

- 各DNS RR(リソースレコード)のTTLが短いとキャッシュサーバの問合せ頻度は高くなる

TTLが86400秒	→ 1日に**1**回 問合せ
TTLが30秒	→ 1日に**2880**回 問合せ

- 気長に嘘のDNS応答をキャッシュサーバに送り続ければ、**そのうち**当たる

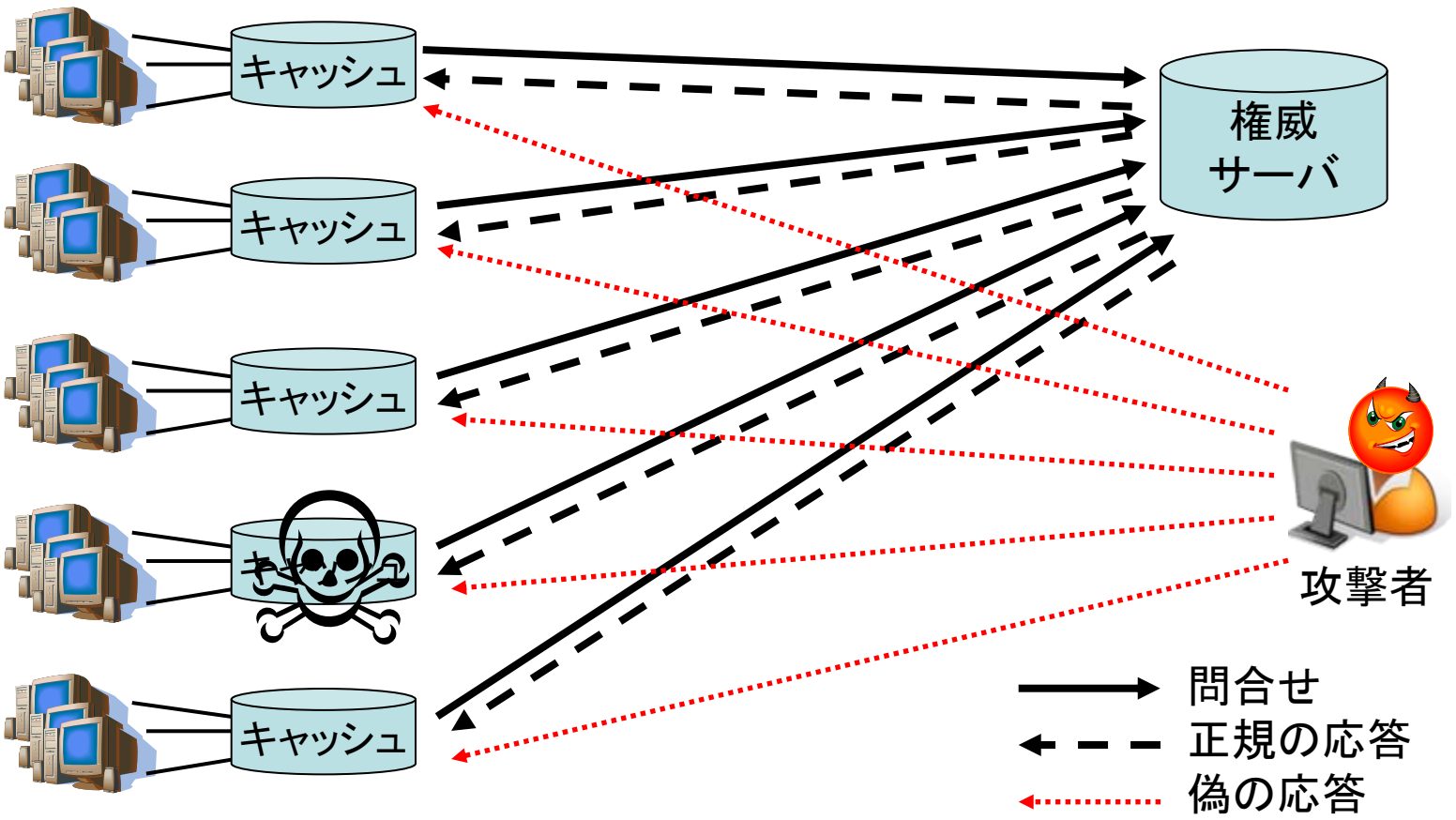
「そのうち」が**意外に短い**

攻撃のストーリ

- アクセスが多くTTLが短いドメイン名を狙う
  - どこのキャッシュサーバでもかまわない
  - 誰が使っていようとかまわない
  - ユーザは多いほどよい
  - 気が付かれないようにこっそりと攻撃したい
- 同時に複数のキャッシュサーバへ
  嘘の応答を定期的に送り続ける

⇒時間の問題で、どこかのキャッシュサーバへの毒入れが成功する

攻撃の様子



あるキャッシュサーバへ毒入れが成功する確率

$$P_S = \frac{R \times W}{N \times Port \times ID}$$

R: 攻撃対象1台あたりに送るパケット量(pps)

W: 攻撃可能な時間(Query⇒AnswerのRTT)

N: 攻撃対象レコードを保持する権威サーバの数

$Port$:問合せに利用するポート数
(**古いBIND**の場合固定なので**1**)

ID: DNSのID (16bit = 65536)

どこかのキャッシュサーバへ毒入れが成功する確率

$$P_V = 1 - (1 - P_S)^V$$

$$= 1 - \left(1 - \frac{R \times W}{N \times Port \times ID}\right)^V$$

V: 攻撃対象のキャッシュサーバの総数

毒入れが1回でも成功する確率

$$P_A = 1 - \left(1 - P_V\right)^A$$

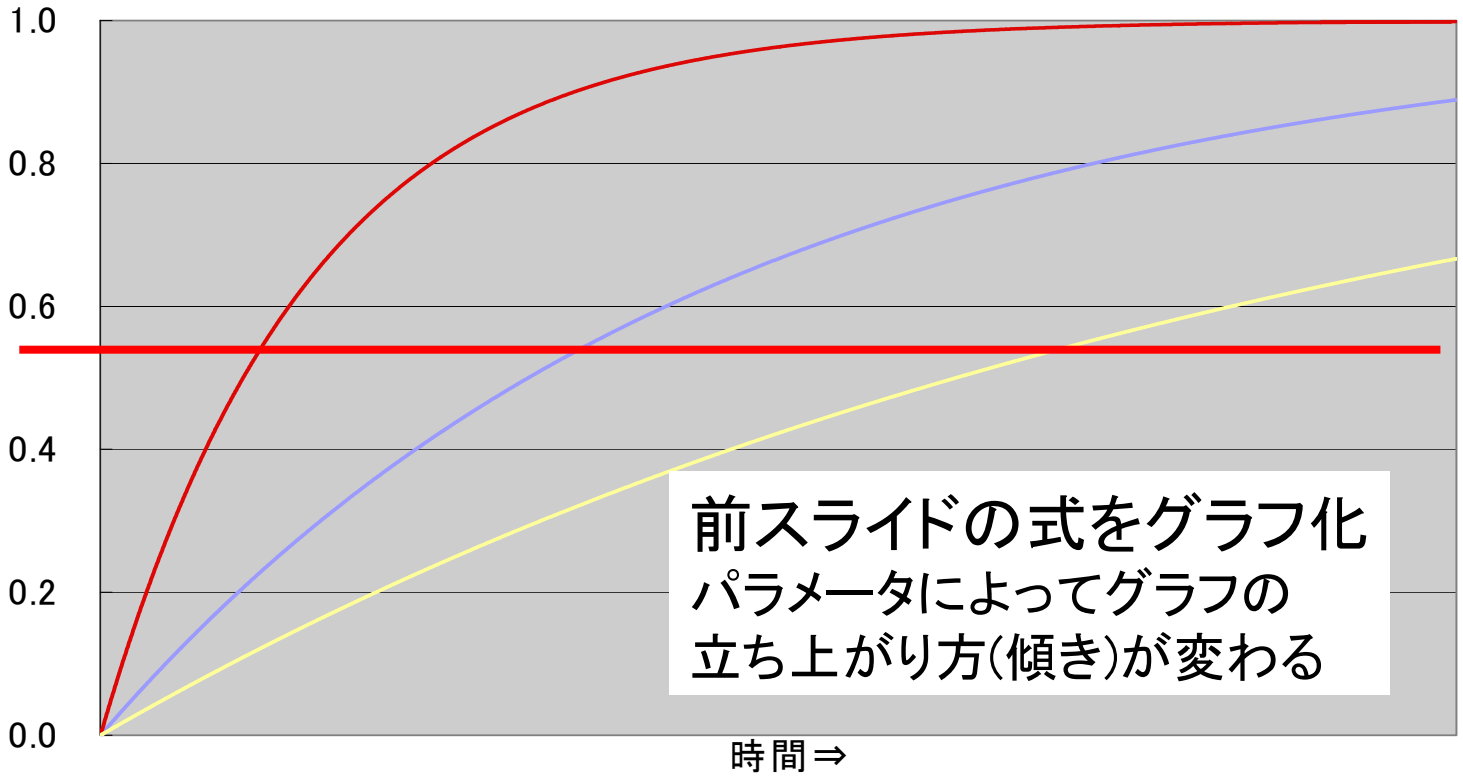
$$= 1 - \left[1 - \left\{1 - \left(1 - \frac{R \times W}{N \times Port \times ID}\right)^V\right\}\right]^{\frac{T}{TTL}}$$

$$= 1 - \left(1 - \frac{R \times W}{N \times Port \times ID}\right)^{\frac{V \times T}{TTL}}$$

A:　攻撃数(=T/TTL)

T:攻撃時間　TTL : DNSレコードのTTL

毒入れが1回でも成功する確率の時系列変化



確率が0.5を超えると毒入れが成功しても不思議ではない

internetweek.jpの例

• TTL	**600秒**
• Authoritative Server	2台(IPv4のみ)
• 1台あたりの攻撃レート	10pps
• 攻撃対象のキャッシュサーバ	500台
• RTT(とりあえず10msと仮定)	10ms
• 確率0.5を超えるまでの時間	**約13日**

internetweek.jpで 仮にTTLを30秒にすると…

- TTL **30秒**
- Authoritative Server 2台(IPv4のみ)
- 1台あたりの攻撃レート 10pps
- 攻撃対象のキャッシュサーバ 500台
- RTT(とりあえず10msと仮定) 10ms
- 確率0.5を超えるまでの時間 **約16時間**
  - 攻撃に必要な総帯域 約4Mbit/sec

Kaminsky型の毒入れ攻撃手法

DNSのリスク

Kaminsky型の毒入れ攻撃

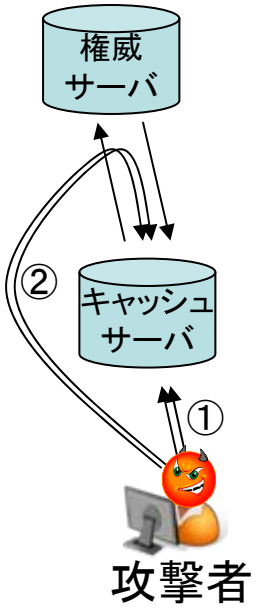
- 攻撃者がキャッシュサーバに、攻撃対象レコードと同じドメインの**存在しない名前**を検索させ、**追加セクションに攻撃対象レコードを設定**した偽装応答をIDを変化させながら大量に送る(偽装応答型の一種)
- www.example.jpの偽IPアドレスをキャッシュさせる

① 問合せ

```
no0000.example.jp.        A
```

② 偽装応答

```
;; 回答セクション
no0000.example.jp.        A        192.0.2.1
;; 権威セクション
example.jp.               NS       www.example.jp.
;; 追加セクション
www.example.jp.           A        192.0.2.10
```



Kaminsky型と他の方式の比較

- Kashpureff型との比較
  - 追加セクションを利用する点は同じ
  - Kashpureff型は現在の実装では外部名のため無視されるが、Kaminsky型は内部名となるため、**キャッシュ対象**となる
- 従来の偽装応答型との比較
  - Kaminsky型は**存在しない名前**を使用するため、攻撃に失敗してもクエリ名を変えることで、TTLに関係なく**連続した攻撃が可能**
    nx0000.example.jp, nx0001.example.jp, nx0002.…

Kaminsky型の攻撃はほぼ100%成功する

Kaminsky型攻撃の対策

- 問合せポートのランダム化
  - キャッシュサーバの問合せポートが固定だったものを、問合せ毎にランダムに変化させ、攻撃成功確率を約1/65000に低減

攻撃1回あたりの成功確率

$$P_S = \frac{R \times W}{N \times 1 \times 65536} \quad \Rightarrow \quad P_S = \frac{R \times W}{N \times 65000 \times 65536}$$

- 対症療法だが、実用上十分な効果を得られる
  - ただし**執拗な攻撃**には耐えられない

各実装での対策状況

- BINDは9.3.5-P1 9.4.2-P1 9.5.0-P1 で対策
  - これ以前のものはKaminsky型攻撃手法には無力
  - パフォーマンス面を 9.3.6、9.4.3、9.5.1で改善
  - 現在は9.4.3-P4、9.5.2-P1、9.6.1-P2以降を強く推奨
- Unboundは当初から対策済み
- dnscache(djbdnsのキャッシュサーバの実装)は、Kaminsky型攻撃手法の攻撃は簡単ではないが、別の攻撃手法が存在するため不適(尚、修正パッチも存在する)

参考:キャッシュ済みのRRは Kaminsky型の攻撃で上書きできるのか

- 攻撃対象はWEBサーバなどのIPアドレス
  - キャッシュサーバがこのようなRRをキャッシュしている
    ⇒権威サーバからの正式な回答でRRを得ている
  - Kaminsky型では、追加セクションにRRを設定する
- 権威サーバからの正式な回答の方が高ランク
  - RFC2181「5.4.1 Ranking data」より
- RFCに忠実に実装してあれば、キャッシュデータの上書きは行わない(BIND 9等)
  - RFC通りに実装していない場合、上書きの可能性がある

まとめ:毒入れ

- キャッシュへの毒入れはDNSプロトコルそのものが持つDNS最大の脆弱性
  - UDPを使う、IDが16bitしかない、etc…
  - 特に、Kaminsky型の攻撃は、成功確率を飛躍的に高めた毒入れ攻撃手法
  - BINDを含め現行の実装は、問合せのポート番号を、都度ランダムに変更するため、攻撃されにくくなっているが、完全ではない
- 完全対処は、DNSプロトコルの拡張であるDNSSECの導入

DNSSECのしくみ

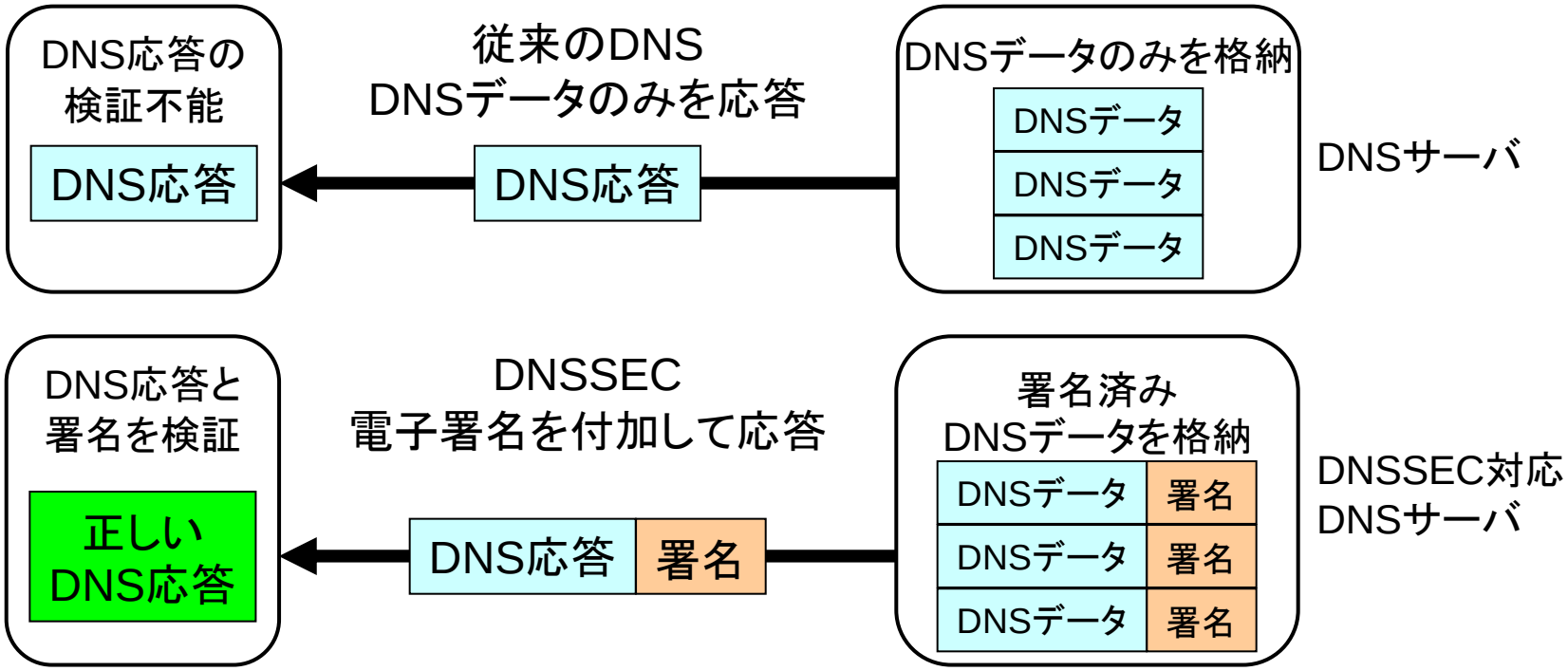
DNSSEC

DNSSECとは

- DNSセキュリティ拡張(DNS SECurity Extensions)
  - 検索側が受け取ったDNSレコードの出自・完全性(改ざんのないこと)を検証できる仕組み
  - 従来のDNSとの**互換性を維持した拡張**
  - Kaminsky型攻撃手法の発覚を1つの契機に、多くのTLDが導入開始あるいは導入予定
- キャッシュへの毒入れを防ぐことができる現状唯一の現実解
  - 他の技術も存在するが標準化が成されていない

従来のDNS vs DNSSEC

- DNSサーバが応答に電子署名を付加し出自を保証
- 問合せ側でDNS応答の改ざんの有無を検出できる



DNSSECのスコープ

- 対象としているもの
  - DNS問合せの回答が、ドメイン名の正当な管理者からのものであることの確認
    ⇒ **出自の保証**
  - DNS問合せの回答における、DNSレコードの改変の検出
    ⇒ **完全性の保証**
- 対象としていないもの
  - 通信路におけるDNS問合せと回答の暗号化
    ※DNSレコードは公開情報という考え方から

DNSSECの現状

DNSSEC

DNSSEC関連RFC

- DNSSECプロトコル関連
  - RFC4033,4034,4035　DNSSEC
  - RFC5155　NSEC3
  - RFC5011　トラストアンカーの自動更新
⇒ BIND 9.7系で実装
- DNSSEC運用関連
  - RFC4641　運用ガイドライン
- その他、IETFで検討継続中のものあり

DNSSEC対応ソフトウェア

- 権威サーバ
  - BIND 9(ISC)、NSD3(NLnetLab)等が対応済み
- キャッシュサーバ
  - BIND 9(ISC)、Unbound(NLnetLab)等が対応済み
- ライブラリとツール
  - OpenDNSSECプロジェクトが進行中
- Microsoft Windows
  - Windows7とWindows Server 2008R2で対応
- インターネットアプリケーション(メーラ、ブラウザ等)
  - 通常はISPのキャッシュサーバで署名検証を行うため、特別な対応は不要

ルートゾーンのDNSSEC対応状況

- 2008年10月
  - ICANNが、DoC(NTIA)、NIST、VeriSignと協調し、2009年中のルート署名を目指すとの声名を発表
- 2009年6月 ICANN35での会合
  - ICANNがKSK、VeriSignがZSKを管理するというモデルを当座の方針として合意
- 2009年7月 IETF75での併設会議
  - ルートゾーンのDNSSEC化の技術的懸念点が指摘
- 2009年10月 RIPE 59
  - 2010年7月から運用開始と発表
    ⇒ 2009年12月にルートゾーンに実験的な署名を実施

TLDのDNSSEC対応状況(1/2)
導入済

種別	TLD名	特記事項
ccTLD	SE(スウェーデン)	・2005年9月に導入開始、世界で最初にDNSSEC対応したTLD ・2009年1月から料金を無料化 ・これまでに多くのノウハウを外部に発信
	PR(プエルトリコ)	・2006年8月に導入開始
	BG(ブルガリア)	・2007年1月に導入開始
	BR(ブラジル)	・2007年6月に導入開始、2009年1月に全属性で対応 ・最新方式(NSEC3)を採用した最初のTLD
	CZ(チェコ)	・2008年9月に導入開始
	TH(タイ)	・2009年3月に導入開始、アジアで最初にDNSSEC対応したccTLD
	TM(トルクメニスタン)	・2009年10月に導入開始
gTLD	MUSEUM	・2008年9月に導入開始
	GOV(米国政府)	・2009年2月に導入開始、2009年末に全組織が対応予定
	ORG	・2009年6月に導入開始、2010年に本サービス化予定

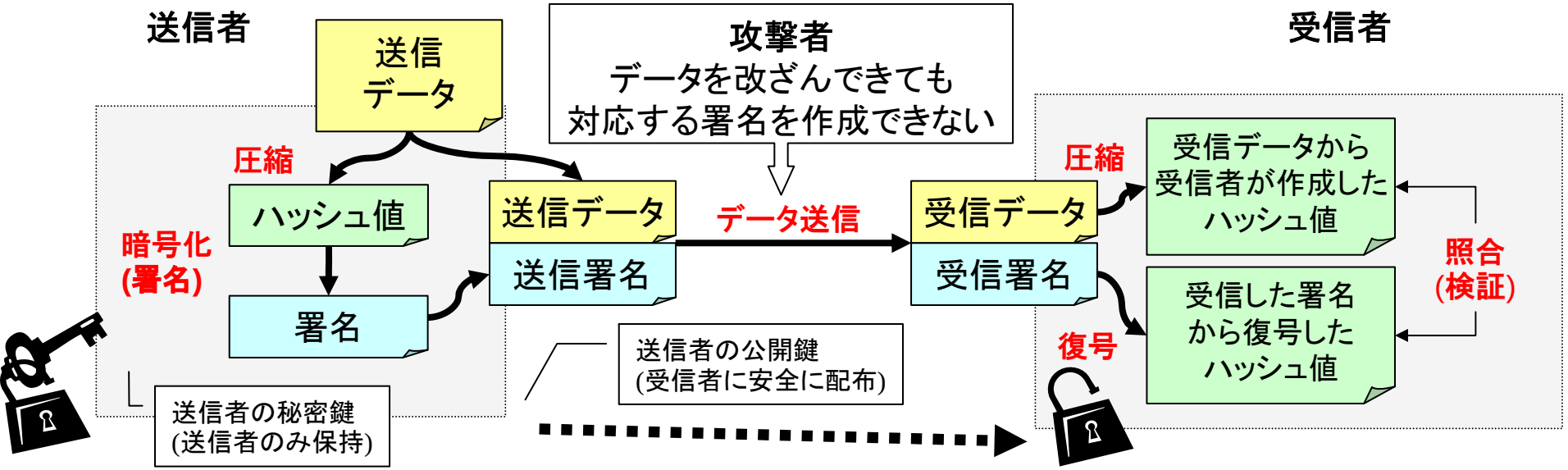
TLDのDNSSEC対応状況(2/2)
導入を表明(非公式を含む)

種別	TLD名	特記事項
ccTLD	CA(カナダ)	・2009年10月にテストベッドを開始
	CH(スイス)	・2009年9月に実地検証開始、2010年2月サービスイン予定
	CN(中国)	・2010年末までに導入予定
	DE(ドイツ)	・2009年5月にテストベッドを開始
	GR(ギリシャ)	
	JP(日本)	・2010年を目処に導入予定
	KR(韓国)	・2010年6月に導入し、2011年1月に全空間で対応予定
	LI(リヒテンシュタイン)	・2009年9月に実地検証開始、2010年2月サービスイン予定
	MY(マレーシア)	・2010年第四四半期に導入予定
	RU(ロシア)	
	UK(イギリス)	・プロトコル策定・IANAとの共同実験など積極的に活動
gTLD	BIZ	
	CAT	・2009年中に導入予定
	COM	・2011年の早い時期に導入予定
	EDU	・2010年3月末に署名予定
	INFO	
	NET	・2010年末までに導入予定

電子署名とDNSSEC

DNSSEC

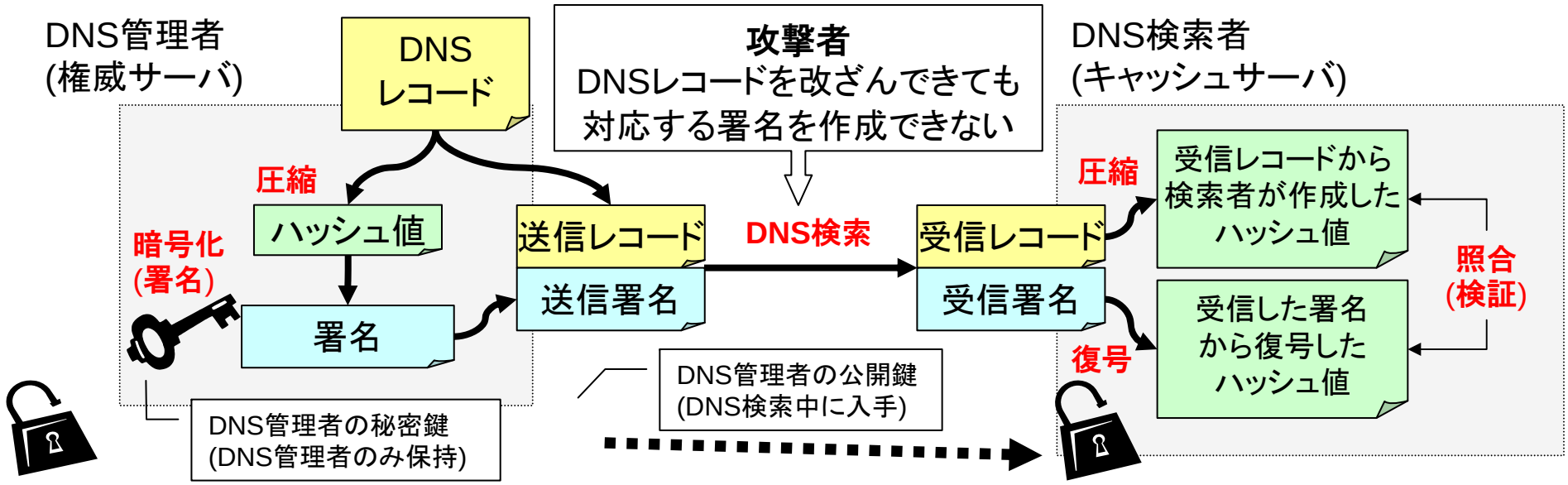
電子署名の概念



電子署名の概念

- 送信者の秘密鍵で送信データのハッシュ値を**暗号化**したものが**署名**
- 公開鍵で署名を復号すれば送信者作成のハッシュ値が得られる
- 受信データから受信者が作成したハッシュ値と、公開鍵で復号したハッシュ値が同じであるか照合(**検証**)する
- 同じであれば送信者からの完全なデータであると判断できる
  - 署名を作成できるのは送信者しかいない(出自の保証)
  - データが改ざんされていれば比較が一致しない(完全性の保証)

電子署名のDNSへの応用 (DNSSEC)



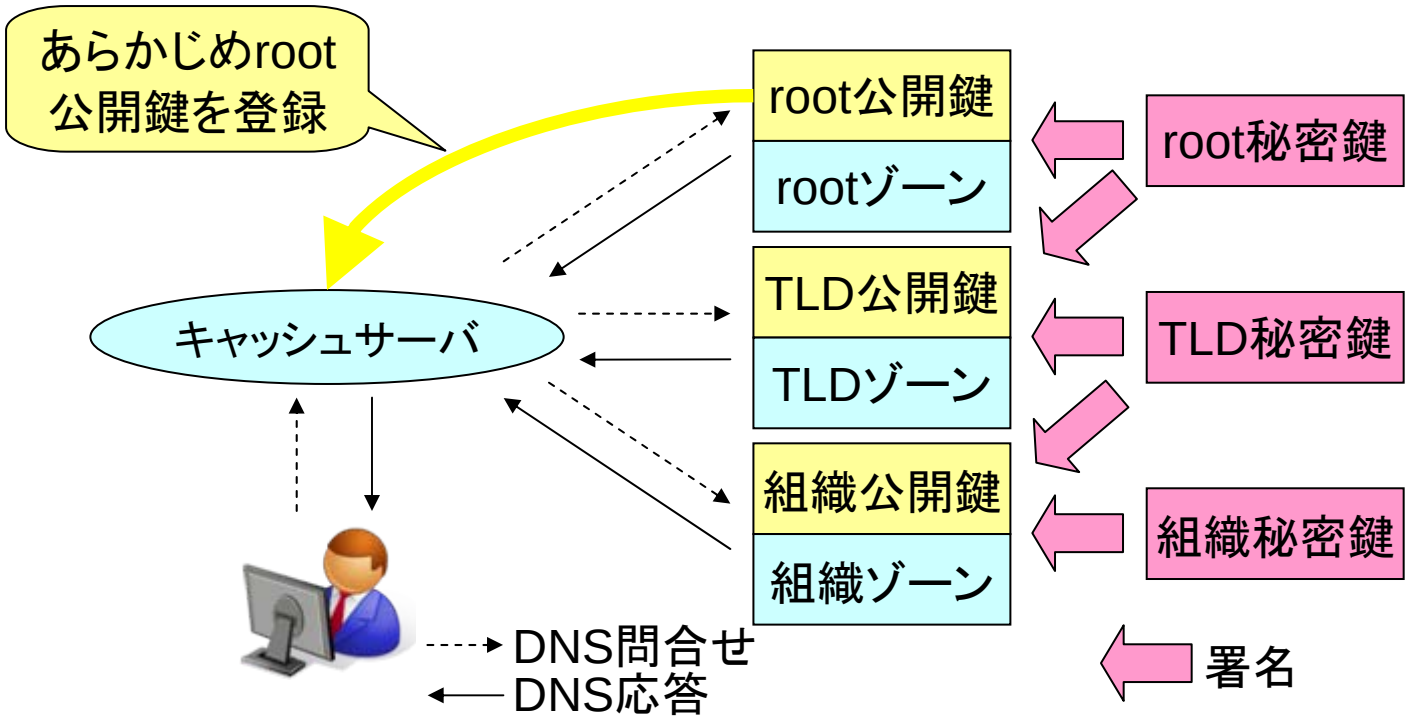
電子署名のDNSへの応用 (DNSSEC)

- DNS管理者は、**署名鍵**(秘密鍵＋公開鍵)を作成
- DNS管理者は、**DNSレコード(ハッシュ値)を秘密鍵で署名**
- 検索を受けたDNSサーバは、**DNSレコードに署名を添付**して回答
- DNS検索者は、DNS管理者の公開鍵を用いて署名を復号し、検索で得たDNSレコードと照合することで、出自・完全性を検証
- この仕組みを基本単位とし、DNS階層で**信頼の連鎖**を作ることで実現

DNSSECの鍵と信頼の連鎖

DNSSEC

DNSSECの信頼の連鎖の概念図



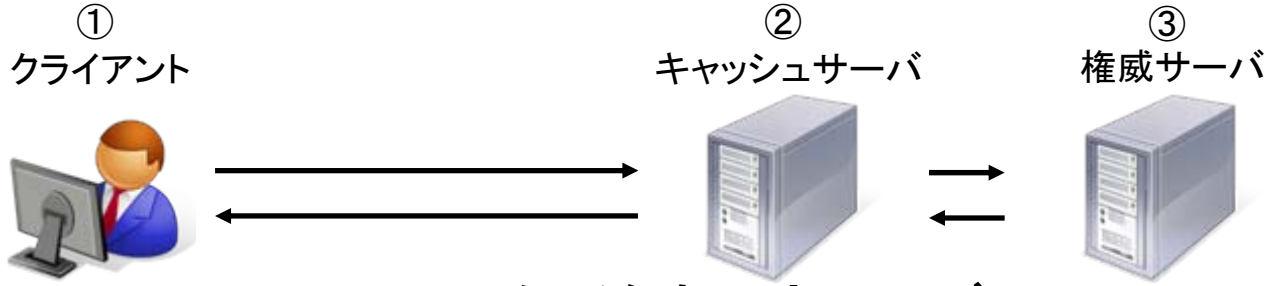
- 秘密鍵で、自ゾーンと下位ゾーンの公開鍵に署名
- root公開鍵をキャッシュサーバに登録することで、rootから組織ゾーンまでの信頼の連鎖を確立

用語:バリデータ(Validator)

- DNSSECにおいて、バリデータは署名の検証を行うもの(プログラム、ライブラリ)を指す
- バリデータの所在
  - キャッシュサーバが署名検証を行う場合、キャッシュサーバがバリデータそのもの
    ⇒ 現状、もっとも一般的なDNSSECのモデル
  - WEBブラウザ等のDNS検索を行うアプリケーションが直接署名検証を行うモデルも考えられる

DNSSEC化による名前解決モデルの変化

- 従来のDNSでの名前解決モデル



- DNSSECでの名前解決モデル

  - 多くの場合バリデータは②に実装
  - バリデータが①に実装されていても問題ない

DNSSSECの2種類の鍵 KSKとZSK、そしてDS (1)

- KSK (Key Signing Key)
  ZSK公開鍵を署名するための鍵
  注) KSK自身の公開鍵にも署名を行う
- 暗号強度の高い鍵
  - RSAで2048bitや4096bitなど
  - 利用期間を長くできるため、鍵更新の頻度を低くできる
  - 署名コストは高いが、少数の鍵情報のみを署名対象とするため問題にはならない
- KSK公開鍵と暗号論的に等価な情報(DSレコード:後述)を作成し、親ゾーンに登録する
  - KSKを変更する場合、DSも更新する

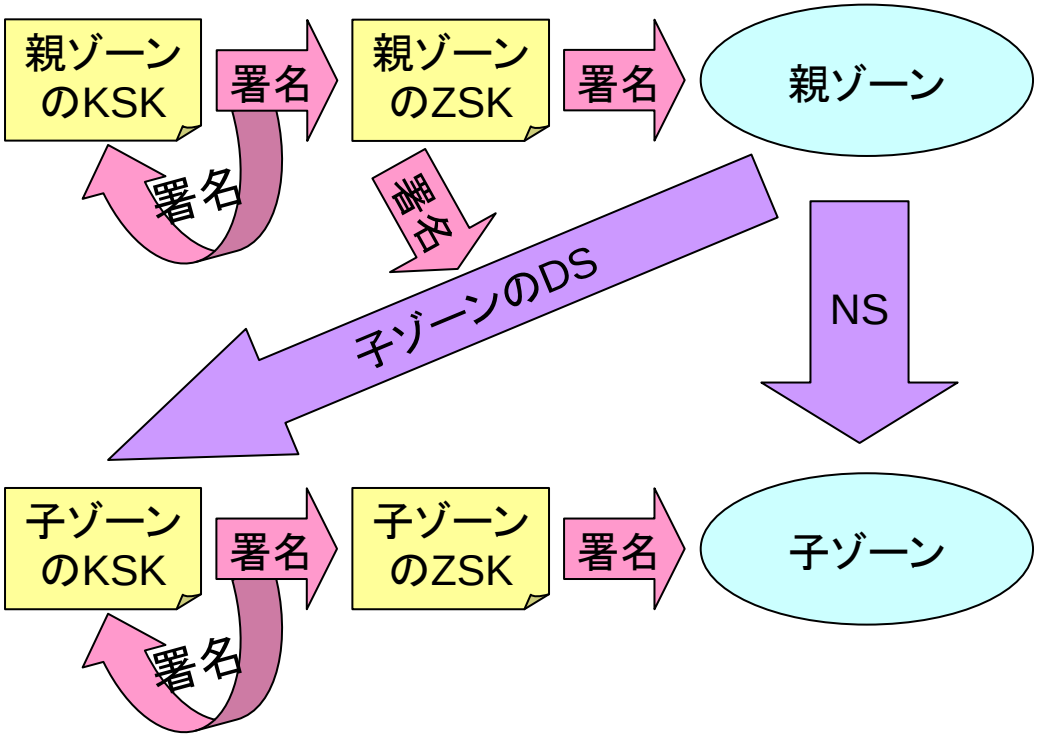
DNSSSECの2種類の鍵 KSKとZSK、そしてDS (2)

- DS - Delegation Signerの略
  - KSK公開鍵を、SHA-1/SHA-256等のハッシュ関数で圧縮したDNSレコード
    ⇒ KSK公開鍵と等価の情報
- 親ゾーンの委任ポイントに、NSと共に子ゾーンのDS情報を登録
  - 親ゾーンの鍵でDSに署名してもらうことで、信頼の連鎖を形成する

DNSSSECの2種類の鍵 KSKとZSK、そしてDS (3)

- ZSK (Zone Signing Key)：
  ゾーンを署名するための鍵
- 暗号強度の低い鍵
  - RSAで768bitや1024bitなど
  - 署名コストが低く大規模ゾーンでも運用できる
  - 安全確保のため、ある程度頻繁に鍵を更新する必要がある
- 鍵更新は親ゾーンとは関係なく独立で行える

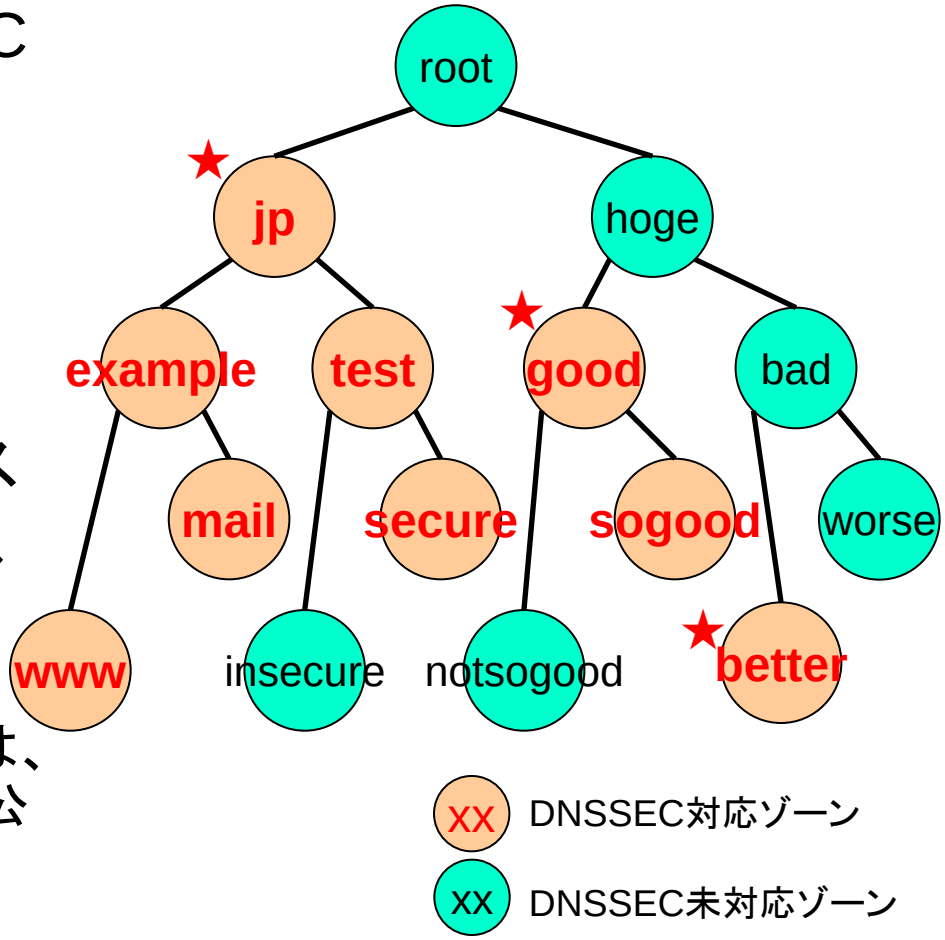
DNSSECの信頼の連鎖



- 公開鍵暗号による信頼の連鎖を形成
- キャッシュサーバが、KSKの公開鍵を使って署名を検証
  ⇒**トラストアンカー**

トラストアンカー

- キャッシュサーバはDNSSEC検証を行う際の頂点となるゾーンのKSK公開鍵を予め登録する
  ⇒ **トラストアンカー**
- この図ではキャッシュサーバが★印のKSK公開鍵をトラストアンカーとして保持すれば、それ以下のドメインを検証できる
  - DNSSEC普及の最終状態では、トラストアンカーはrootのKSK公開鍵のみとなる



DNSSECの
リソースレコード(RR)

DNSSEC

DNSSEC関係のRR一覧

- DNSKEY　KSK・ZSK公開鍵の情報
- RRSIG　各RRへの署名
- DS　KSK公開鍵のハッシュ値を含む情報(親ゾーンに登録)
- NSEC　次のRRへのポインタと存在するレコード型の情報
- NSEC3　NSECを改良したもの(後述)
- NSEC3PARAM　NSEC3に必要な情報

DNSKEY RR

- KSKとZSKの公開鍵を示すRR
  - オーナー名はゾーン頂点(=ゾーン名)
  - KSKとZSKを必要に応じて複数(後述)設定

```
example.jp. IN DNSKEY
    256 3 5 AwEAAeNO41ymz+Iw(行末まで省略)
     ①  ②③                ④
```

① フラグ(256:ZSK、257:KSK)

② プロトコル番号 (3のみ)

③ 暗号化アルゴリズム

④ 公開鍵(Base64で符号化)

DNSSEC暗号化アルゴリズム(抜粋)

番号	略称	参照
5	RSASHA1	[RFC3755] [RFC3110]
7	NSEC3RSASHA1	[RFC5155]
8	RSASHA256	[RFC5702]
10	RSASHA512	[RFC5702]

注) 5と7に差は無く、NSECとNSEC3 (後述) で使い分ける

DNSSECの暗号化アルゴリズム一覧

http://www.iana.org/assignments/dns-sec-alg-numbers

DS RR

- DS - Delegation Signer
  - 子ゾーンのKSKの正当性を親ゾーンで承認
  - 親ゾーンにのみ記述する唯一のRR

```
example.jp. IN DS 63604 5 1 DF…(16進数40文字)
example.jp. IN DS 63604 5 2 E8…(16進数64文字)
                  ①    ② ③        ④
```

① 鍵のID
② 暗号化アルゴリズム
③ ハッシュのアルゴリズム(1:SHA-1，2:SHA-256)
④ ハッシュ化したKSK公開鍵

RRSIG RR

- 各RRへの署名で、RRset毎に存在する

```
ns.example.jp. IN RRSIG A 5 3 86400
                         ① ② ③   ④
    20091208144031 20091108144031 40002 example.jp.
          ⑤              ⑥          ⑦        ⑧
        NiVihYAIZBEwfUUAbPazDRIbvhNH8S(以下省略)
                              ⑨
```

① 署名対象のRRの種類
　　ここではns.example.jpのA RR

② 暗号化アルゴリズム

③ ラベルの数
”ns.example.jp”だと3、”*.example.jp”だと2

RRSIG RR(続き)

④ 署名対象RRのTTL
⑤ 署名の有効期限
⑥ 署名の開始時刻
⑦ 鍵のID
⑧ ドメイン名
⑨ 署名

- 署名は、元のRRの全て(TTL、クラス等を含む)と、RRSIGの署名そのものを除いた残りを含めて計算し作成する

DNSSECにおける不在証明

- DNSSECではドメイン名が存在しない場合、**存在しないこと**を**証明**する必要がある
  - 万が一存在しないドメイン名を偽装されたときのために、存在するのかどうかを検証できる仕組みが必須
- 存在するRRは署名(RRSIG RR)を付加して検証することで存在を証明できる
  ⇒ 存在しないRRは署名不能

ハンバーガーのパティの有無

- パティ(肉)が有る
  - パティの存在を判断できる
- パティが無く、バンズ(パン)も無い
  - パティが無いかどうか判断不能
    ⇒ 単純に配膳が遅れているだけ?
- クラウン(バンズ上部)とヒール(バンズ下部)があるのにパティが無い
  - クラウンとヒールの存在が判断できる
    ⇒ パティが存在しないことを確実に判断できる

NSEC RR

- NSECは存在しないものを証明(署名検証)するためのRR
  - 存在するレコードすべてを整列し、次のレコードへのリストを生成することで、存在しないものを証明する
  - NSEC RRにRRSIGを付加し署名検証を行う

NSEC RRの例

- sec2.example.jpを問合せた場合の応答

```
sec1.example.jp.  IN  NSEC
   sec3.example.jp. NS DS RRSIG NSEC
```

(権威セクションで応答)

- sec1.example.jp の次(アルファベット順)のドメイン名は sec3.example.jpで、NS, DS, RRSIG, NSECのRRが存在する
  ⇒ sec2.example.jp は存在しないことを示す

NSEC3 RR

- NSECを使った不在証明では、NSEC RRを辿れば、完全なゾーンデータを入手できる
  ⇒NSEC方式はゾーンデータの公開と等価
  - Walker(DNSSEC Walker)というツールで、NSEC方式のDNSSEC化ゾーンのデータを入手可能
- **NSEC3** (RFC 5155)
  - ドメイン名を一方向性ハッシュ関数でハッシュ化したものを整列する

NSEC3 RRの例

```
4HTJTU7UP56274L1C00Q9MLPHG2A2H85.example.jp.
 IN NSEC3
 1 0 3 123ABC ←NSEC3の関連パラメータ
B0B790UE4SAE4QB4RTB3PJSIH6JAOB7R NS DS RRSIG
```

NSEC RRと比べると

- ラベルがハッシュ化されBase32でエンコード
  - 元のドメイン名は推測不能
- NSEC3の関連パラメータを付加

NSEC3の関連パラメータ

- 前スライドのRR例

```
1 0 3 123ABC
① ② ③    ④
```

① ハッシュアルゴリズム(1:SHA-1 RFC5155)

② NSEC3 オプトアウトフラグ
(1ならオプトアウト、0はオプトアウトしていない)

③ 繰り返し

④ ソルト(16進数で表記。例は3バイト分のソルト)

NSEC3のハッシュ値計算方法

- ハッシュの計算アルゴリズム
  ① 値に**ソルト**を結合
  ② 次に**ハッシュアルゴリズム**でハッシュ値を計算
  ③ ①、②を**繰り返し**で指定された回数適用する
- 計算の元になる値は、小文字で正規化したドメイン名(のワイヤーフォーマット)

NSEC3でのオプトアウト

- 一部の委任先がDNSSEC化している場合
  - 主に.JP、.COM等のTLDで該当
- ゾーン内のレコード全てにNSEC3 RRを用意すると、それに付随するRRSIGを含めた計算コストが膨大となる
  - DNSSEC化していない委任情報に、署名を付加する必要性は薄い
- 必要のある委任先にのみNSEC3 RRを用意

NSEC3PARAM RR

```
example.jp. IN NSEC3PARAM 1 0 3 123ABC
```

- ゾーン提供(権威サーバ)側が、NSEC3の計算を行うために必要なレコード
  - NSEC3のパラメータを抜き出したもの
  - オーナー名はゾーン頂点(ゾーン名)

NSEC3での権威サーバの応答

- 存在しない名前の検索を受けた権威サーバ
  - クエリ名のハッシュ値を計算
  - あらかじめ整列してあるNSEC3 RRの中から、前後に該当するものを署名と共に権威セクションで応答
  - 実際は複数のNSEC3を応答
    (説明省略　RFC5155 Section 7.2参照)
- NSEC3のオーナー名の問合せ
  - ドメイン名としては存在しないもの
    ⇒ 名前エラー(NXDOMAIN)を応答する

BINDキャッシュサーバでのDNSSECの設定

DNSSEC

DNSSEC対応の実装

- NSEC対応の実装
  - BIND 9.3.0 ～
    権威サーバとキャッシュサーバ
  - NSD 2.0.0 以降　　権威サーバのみ
  - Unbound　　キャッシュサーバのみ
- NSEC3対応の実装
  - BIND 9.6.0以降
  - NSD 3.0以降　　権威サーバのみ
  - Unbound　　キャッシュサーバのみ

BINDキャッシュサーバでのDNSSECの設定

- 通常のキャッシュサーバの設定に、署名の検証を行う設定を追加する
- named.conf の options 部分以下を追加する

```
dnssec-enable        yes;
dnssec-validation    yes;
```

- 署名の検証に必要な情報を登録する
  - 検証対象の公開鍵情報の登録
    ⇒ トラストアンカーの登録

署名の検証を行うオプション

- dnssec-enable
  - DNSSEC対応にするかどうかのオプション
  - BIND 9.4以降のデフォルト　yes
- dnssec-validation
  - DNSSECの署名検証を行うかどうかのオプション
  - BIND 9.4のデフォルト　no
  - BIND 9.5以降のデフォルト　yes

```
options {
    ....
    dnssec-enable         yes;     // BIND 9.5以降であれば
    dnssec-validatioin    yes;     // 設定しなくてもよい
    ....
};
```

KSK公開鍵の入手

- DNSSEC対応ドメインのKSKの公開鍵を入手する
  - 以下.ORGと.SEを例にする
- .ORG の公開鍵
  http://www.pir.org/dnsseckeys.php?db=content/Website&tbl=ORG_Advantage&id=2.4
  - http://www.pir.org/からDNSSECを辿る
- .SE の公開鍵
  https://www.iis.se/docs/ksk.txt
  - http://www.iis.se/en/domaner/dnssec/ より

.ORGの公開鍵 (2009年11月時点)

.SEの公開鍵(2009年7月時点)
2つある(後述)

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

se.                         IN DNSKEY 257 3 5 (
                                      AwEAAdKc1sGsbv5jjeJ141IxNSTdR+nbtFn+JKQpvFZE
                                      TaY5iMutoyWHa+jCp0TBBAzB2trGHzdi7E55FFzbeG0r
                                      +G6SJbJ4DXYSpiiELPiu0i+jPp3C3kNwiqpPpQHWaYDS
                                      9MTQMu/QZHR/sFPbUnsK30fuQbKKkKgnADms0aXalYUu
                                      CgDyVMjdxRLz5yzLoaSO9m5ii5cI0dQNCjexvj9M4ec6
                                      woi6+N8v1pOmQAQ9at5Fd8A6tAxZI8tdlEUnXYgNwb8e
                                      VZEWsgXtBhoyAru7Tzw+F6ToYq6hmKhfsT+fIhFXsYso
                                      7L4nYUqTnM4VOZgNhcTv+qVQkHfOOeJKUkNB8Qc=
                                      ); key id = 49678
se.                         IN DNSKEY 257 3 5 (
                                      AwEAAeeGE5unuosN3c8tBcj1/q4TQEwzfNY0GK6kxMVZ
                                      1wcTkypSExLCBPMS0wWkrA1n7t5hcM86VD94L8oEd9jn
                                      HdjxreguOZYEBWkckajU0tBWwEPMoEwepknpB14la1wy
                                      3xR95PMt9zWceiqaYOLEujFAqe6F3tQ14lP6FdFL9wyC
                                      flV06K1ww+gQxYRDo6h+Wejguvpeg33KRzFtlwvbF3Aa
                                      pH2GXCi4Ok2+PO2ckzfKoikIe9ZOXfrCbG9ml2iQrRNS
                                      M4q3zGhuly4NrF/t9s9jakbWzd4PM1Q551XIEphRGyqc
                                      bA2JTU3/mcUVKfgrH7nxaPz5DoUB7TKYyQgsTlc=
                                      ); key id = 8779

-----BEGIN PGP SIGNATURE-----
Version: PGP Desktop 9.8.3 (Build 4028)
Charset: utf-8

wj8DBQFJQmz4/OxRKPRA7psRAqKyAKCqzF2oamv1kwY3/5f27ioxicVMZACfX8By
sKp405q8KBbheYVYKb5gE7k=
=T8Is
-----END PGP SIGNATURE-----
```

トラストアンカー(KSK公開鍵)の登録

- named.conf にトラストアンカーを登録
  - それぞれの公開鍵情報から、
    “IN DNSKEY” と “(“ “)” を除いてtrusted-keysに設定

```
trusted-keys {
      "org“    257 3 7
               "AwEAAYpYfj3aaRzzkxWQqMdl7YExY81NdYSv+qayuZDo
               dnZ9IMh0bwMcYaVUdzNAbVeJ8gd6jq1sR3VvP/SR36mm
               <中略>
               DqL+3wzUdF5ACkYwt1XhPVPU+wSIlzbaAQN49PU=" ;

      "se“     257 3 5
               "AwEAAeeGE5unuosN3c8tBcj1/q4TQEwzfNY0GK6kxMVZ
               1wcTkypSExLCBPMS0wWkrA1n7t5hcM86VD94L8oEd9jn
               <中略>
               bA2JTU3/mcUVKfgrH7nxaPz5DoUB7TKYyQgsTlc=“ ;
};
```

trusted-keysの設定

- trusted-keysの書式
  - ドメイン名 数字 数字 数字 公開鍵
  - 公開鍵は“ “ で囲み、空白、TAB、改行等があってもよい
- 本例では.ORGと.SEの公開鍵を設定
  - .SEの場合、現在2種類の鍵が用意されている
    Key ID 49678　2008年から2009-12-31まで有効
    Key ID 8779　 2009年から2010-12-31まで有効
    いずれか一方を登録する(有効期限の長いものを利用)
- 近い将来、ルートゾーンがDNSSECで署名され各TLDがルートゾーンへDNSSEC対応の情報(DS)を登録すれば、**”.”の公開鍵のみを設定**する

キャッシュサーバの動作確認

- named.conf の変更が終わったら、キャッシュサーバ用のnamedを再起動する
- digコマンドでDNSSEC対応ゾーンの確認

```
$ dig @127.0.0.1 +dnssec org soa

$ dig @127.0.0.1 +dnssec www.iis.se a
```

キャッシュサーバへのdigの結果

```
$ dig @127.0.0.1 +dnssec www.iis.se a

; <<>> DiG 9.6.1 <<>> @127.0.0.1 +dnssec www.iis.se a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3247
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.iis.se.                    IN      A

;; ANSWER SECTION:
www.iis.se.             60      IN      A       212.247.7.221
www.iis.se.             60      IN      RRSIG   A 5 3 60 20090702105501 20090622105501 22079 iis.se. nLM6<行末まで省略>

;; AUTHORITY SECTION:
iis.se.                 3600    IN      NS      ns.nic.se.
iis.se.                 3600    IN      NS      ns3.nic.se.
iis.se.                 3600    IN      NS      ns2.nic.se.
iis.se.                 3600    IN      RRSIG   NS 5 2 3600 20090702105501 20090622105501 22079 iis.se. E<行末まで省略>

;; Query time: 1402 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Jun 25 19:29:48 2009
;; MSG SIZE  rcvd: 44402 msec
```

digの結果のflagsフィールド

- flags: qr rd ra **ad**;
  - DNSにおけるさまざまな状態を表すフラグ
- DNSSECに関係するflag
  - ad: Authentic Data
    署名が検証できた正しいデータであることを示す
  - cd: Checking Disabled
    署名のチェックを行っていない状態を示す
- 署名の検証を行わない場合は+cdを指定

```
dig @127.0.0.1 +cd www.iis.se a
  flags: qr rd ra cd;
```

  - BINDはtrusted-keysを設定すると内部では**必ず署名の検証を行う**

DNSSEC検証の失敗

- 誤ったトラストアンカーを設定した場合

```
dig  +dnssec www.iis.se a
status: SERVFAIL
```

⇒ 答えが得られない

- 現行のBINDでは、誤ったトラストアンカーを設定すると、異常な時間がかかる(現行BINDの不具合?)
  - 手元の実験環境で.SEドメインに誤った公開鍵を登録してみたところ、27秒程必要だった
  - digはデフォルトで15秒待つ(5秒待ちリトライを2回)
    ⇒ digのデフォルトのままではタイムアウトとなる

BIND権威サーバでのDNSSECの設定

DNSSEC

DNSSEC鍵の作成: dnssec-keygen

- -a 鍵生成アルゴリズムの指定
  - NSEC3RSASHA1などを指定する
- -b ビット長
  - ZSK:NSEC3RSASHA1の場合 1024ビット以上
  - KSK:NSEC3RSASHA1の場合 2048ビット以上
- -f KSK
  - KSKを作成する場合に指定
- 最後に名前(ゾーン名)を指定

dnssec-keygen の実行

- ZSKを作る

```
dnssec-keygen -a NSEC3RSASHA1 -b 1024
                    example.jp > zsk-example.jp
```

  - 鍵のファイル名を表示するので、その結果を保存する
    K*example.jp*.+007+23522

  3桁の数字はアルゴリズム、5桁は識別子(ID)

- 1組の鍵ファイルができる

  K*example.jp*.+007+23522.key　　⇒公開鍵

  K*example.jp*.+007+23522.private　　⇒秘密鍵

- KSKを作る

```
dnssec-keygen -a NSEC3RSASHA1 -b 2048 –f KSK
                    example.jp > ksk-example.jp
```

鍵ファイルの中身の例(公開鍵)

```
; This is a zone-signing key, keyid 23522, for example.jp.
; Created: Tue Nov 10 14:56:10 2009
; Publish: Tue Nov 10 14:56:10 2009
; Activate: Tue Nov 10 14:56:10 2009
example.jp. IN DNSKEY 256 3 7
   AwEAAfzJXPiYtSD8DJs+J36dZd+cNrXHxLpuY2xNTF2e0KolkMiVJnse
   zzLcuzrgGP1IeVBCiI+LFQFDcXV69gJZKUpefeOrZ1IJLaVwbkW3pxDo
   2u3qhxY6lr0hgRsmwZ5XVIEnMdOOzdGzZl0VvPOGMNC94WFM+RciLySk
   2QSoJzmz
```

- BIND 9.7.0b2のdnssec-keygenで作成
  - 9.6までのものとはコメント部分に差異がある

鍵ファイルの中身の例(秘密鍵)

```
Private-key-format: v1.3
Algorithm: 7 (NSEC3RSASHA1)
Modulus:
    /Mlc+Ji1IPwMmz4nfp1l35w2tcfEum5jbE1MXZ7QqiWQyJUmex7PMty7OuAY/Uh5UEKIj4sVAUNxdXr2AlkpSl5946tnUgktpXBuR
    benEOja7eqHFjqWvSGBGybBnldUgScx047N0bNmXRW884Yw0L3hYUz5FyIvJKTZBKgnObM=
PublicExponent: AQAB
PrivateExponent:
    M88xduIVfYUrMEY04gZwcrwZmngvIeauCext0mJScgzw96taD7Ho1YvX8+EqPf80nfaE9qaSz4d7IZDqCuErTOJ5stR6uFR69g3av
    8S+j1sw8hD2J3jo7r6m5nfcfTJ//WFaVyojQigu0vMn27gD7tcVLhztyAqJ5muklT8yngE=
Prime1: /9DNJ5ujOsJOyH157EF+hqvsk32XittuPSc9RzHPwUmGdLY1FYq0Eqpr7pRPSFfm7ATRBW9/WNoG26Al+XMOUw==
Prime2: /PgAvwBYrNAS8WqqkLodjowQtApmxCe43iUDjrIERoGaxFPQZigy6IeVodhPeEAolKTP+PC4ttiuYEOtqr37IQ==
Exponent1: WsXltlNExXnjaMMVe172HaVt6hwbpPseD/cXiGbFeKm1Wz64cW9pXGI6sErSIzKFz2QaI1qgDpA29MHMF8ra3w==
Exponent2: LLemYh0sj7fkcVqatiTATs+BsGHaUrh23IYMf/AGA3SrqCLsxvI6NZKqJ8b2HVqyEbykquvaqy/Ye1nbXEBjIQ==
Coefficient: lR/OGOLG5qMAR6LS+cBTchenJ3b17zUnenOdNhLGlserypcpvPWMAIg3VKfIJD9gjiYjWVkaT0dotZ4trUTiPQ==
Created: 20091110055610
Publish: 20091110055610
Activate: 20091110055610
```

- BIND 9.7.0b2のdnssec-keygenで作成
  - 9.6系までは、formatのバージョンがv1.2
  - v1.2はBIND 9.7系のツールでも扱えるが、v1.3は9.7系のツールのみ

dnssec-keygenの注意点

- KSKとZSKの区別に注意する
  - 2組の鍵ファイル(計4個)ができ、
    見た目での識別は困難
- 実行時に鍵ファイル名を保存すると良い

```
dnssec-keygen –f KSK .. > ksk-...
dnssec-keygen ......... > zsk-...
```

ゾーンへの署名：dnssec-signzone

- 署名対象ゾーンファイル、ZSK、KSKを準備
  - 同じディレクトリに用意し、ゾーンファイルは
    ゾーン名とファイル名を一致させると便利

example.jp

Kexample.jp.+007+21891.key	KSK公開鍵
Kexample.jp.+007+21891.private	KSK秘密鍵
Kexample.jp.+007+23522.key	ZSK公開鍵
Kexample.jp.+007+23522.private	ZSK秘密鍵

ゾーンへの署名(続き)

- ゾーンファイルにKSK、ZSKの公開鍵を登録
  - 公開鍵をまとめたファイルを用意し、$INCLUDE文を利用してゾーンファイルから参照する

```
cat `cat ksk-example.jp`.key
      `cat zsk-example.jp`.key > example.jp.keys
```

```
;ゾーンファイル中でkeyファイルを参照
$INCLUDE example.jp.keys
```

- SOAシリアル値の管理は、dnssec-signzoneの -n オプションにまかせるのがベター

署名前のゾーンファイル

```
$TTL    1D
$INCLUDE example.jp.keys
@       IN      SOA     ns root (
                        1       ; Serial
                        10800   ; Refresh
                        3600    ; Retry
                        3600000 ; Expire
                        1800 )  ; Minimum TTL
                NS      ns
                MX      10 mail
;
ns              A       192.0.2.17
www             A       192.0.2.18
mail            A       192.0.2.19

sub1            NS      ns.sub1
ns.sub1         A       192.0.2.49

sec3            NS      ns.sec3
ns.sec3         A       192.0.2.65
$INCLUDE ../sec3.example.jp/dsset-sec3.example.jp.

sub3            NS      ns.sub3
ns.sub3         A       192.0.2.81
```

署名の実行

- dnssec-signzone -H <繰り返し回数> -3 <salt>
  -n <SOAのシリアル値> -k <KSK>
  <ゾーンファイル> <ZSK>

```
dnssec-signzone –H 3 -3 123ABC –n unixtime
   –k `cat ksk-example.jp`.private
   example.jp `cat zsk-example.jp`.private
```

  - -3はNSEC3方式を選びソルトを指定するオプション
- 出力ファイル
  - example.jp.signed　署名済みのゾーン
  - dsset-example.jp.　ゾーンへのDS RR

署名済みのゾーンファイル(抜粋)

```
; File written on Tue Nov 10 16:48:50 2009
; dnssec_signzone version 9.7.0b2
example.jp.             86400   IN SOA  ns.example.jp. root.example.jp. (
                                        1257839330 ; serial
                                        <中略>
                                        )
                        86400   RRSIG   SOA 7 2 86400 20091210064850 (
                                        20091110064850 23522 example.jp.
                                        CDq8qzNsLVa6pRD9VUE71IYzIaO7u5NtYwwM
                                        <中略>
                                        UMhqKQinfJHi/8hv4ff5FK198Dc= )
                        86400   NS      ns.example.jp.
                        86400   RRSIG   NS 7 2 86400 20091210064850 (
                                        20091110064850 23522 example.jp.
                                        UICLoNT5Zszv8LzF0mrkslDMwf9KBmiRSbhN
                                        <中略>
                                        oY1VNG0n6B+Q2ksY12ZXLK4G0yw= )
                        86400   MX      10 mail.example.jp.
                        86400   RRSIG   MX 7 2 86400 20091210064850 (
                                        20091110064850 23522 example.jp.
                                        TQz52cCZQvpgcMFyRPtM2BWKxE8Vfvj/RmSv
                                        <中略>
                                        7GKlXyx3aHYyX3w9O03iXFQz7PA= )
                        86400   DNSKEY  256 3 7 (
                                        AwEAAfzJXPiYtSD8DJs+J36dZd+cNrXHxLpu
                                        <中略>
                                        Zl0VvPOGMNC94WFM+RciLySk2QSoJzmz
                                        ) ; key id = 23522
                        86400   DNSKEY  257 3 7 (
                                        AwEAAe1MfTlcaIiidHDoCMmhuizPPoO5Tzzh
                                        <中略>
                                        wZmOr6UvsYzCJLLJsYb9HH8=
                                        ) ; key id = 21891
```

DSの登録

- dsset-example.jpの内容を親ドメインに登録
  - 子ゾーンの署名時に生成されたもの
  - 内容の例

```
example.jp. IN DS 21891 7 1 6786BB<中略>DBC9A159E
example.jp. IN DS 21891 7 2 12EFC0<中略>0ED6ED6 AFE9130B
```

- DSレコードはKSKの公開鍵からも生成可能
  - dnssec-dsfromkeyコマンドを使用する

BINDの設定：権威サーバ(1/2)

- DNSSECを有効にする
  - named.conf の options 部分に dnssec-enable yes; を追加

```
options {
        ＜省略＞
        dnssec-enable yes; // BIND 9.4以降は
                           // デフォルトが yes;
        ＜省略＞
};
```

BINDの設定：権威サーバ(2/2)

- ゾーンファイルを署名済みのものに変更

```
zone "example.jp" {
        type master ;
        // file "example.jp.zone" ;
        file "example.jp.signed" ;
} ;
```

- namedを再起動

```
rndc reload
```

権威サーバの動作確認

- digコマンドでDNSSEC対応ゾーンの確認

```
dig +dnssec +norec @127.0.0.1 www.example.jp a
```

+dnssec DNSSECを有効にする問合せ

このオプションなしでは通常の(DNSSECでない)ものと同じ結果が返る

+norec　非再帰的問合せ

キャッシュサーバから権威サーバへの問合せと同じ形式の問合せ

権威サーバへのdigの結果(1/2)

- +dnssecなしのdigの応答

```
; <<>> DiG 9.6.1-P1 <<>> +norec @127.0.0.1 www.example.jp a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5506
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;www.example.jp.                        IN      A

;; ANSWER SECTION:
www.example.jp.         86400   IN      A       192.0.2.18

;; AUTHORITY SECTION:
example.jp.             86400   IN      NS      ns.example.jp.

;; ADDITIONAL SECTION:
ns.example.jp.          86400   IN      A       192.0.2.17

;; Query time: 8 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Nov 17 10:36:53 2009
;; MSG SIZE  rcvd: 81
```

権威サーバへのdigの結果(2/2)

- +dnssecありでは、RRSIG RRを加えたものが返る

```
; <<>> DiG 9.6.1-P1 <<>> +dnssec +norec @127.0.0.1 www.example.jp a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60940
;; flags: qr aa; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.example.jp.                        IN      A

;; ANSWER SECTION:
www.example.jp.         86400   IN      A       192.0.2.18
www.example.jp.         86400   IN      RRSIG   A 7 3 86400 20091210064850 20091110064850 23522 example.jp.
      8QfGxUywqIJM1w5adioi8vN2SgfItsKYXPG9Y9qEOwk7I6eMUaeD49dw nepp+JqVr+zmjkL8hFpZYqu8wmZzF016gdRNSQuKV7WkzK3YXP13ft5a
      DOdUCjHarZVzyh62aV1canDOIPBYto0GLFMGnDgjvyLNw8jktdFth803 L3k=

;; AUTHORITY SECTION:
example.jp.             86400   IN      NS      ns.example.jp.
example.jp.             86400   IN      RRSIG   NS 7 2 86400 20091210064850 20091110064850 23522 example.jp.
      UICLoNT5Zszv8LzF0mrkslDMwf9KBmiRSbhN9NBAcdr/WpBRtUeg60/k gD/JM/15gvGEEHqPWpj66BYfC91HdrrRBTma8VSc1x7xz8nhu4WUFnTs
      hQIupzW9mFtoO88D2NaOB6bYLOd8WdXDoY1VNG0n6B+Q2ksY12ZXLK4G 0yw=

;; ADDITIONAL SECTION:
ns.example.jp.          86400   IN      A       192.0.2.17
ns.example.jp.          86400   IN      RRSIG   A 7 3 86400 20091210064850 20091110064850 23522 example.jp.
      urhhO8ocpy3dD5FRLBuUPFWqZga2vXILEA8UdjxQU+nAjzh5xFtUP26L /lxGgDxi3JjMv+hkfoIIPKrjlAQvIp2yh5Jv05kRHTlBXKIJX4ze4g5X
      BukWAXSseSQDCqrVUbLzhxTofIVeTgXXMuDlYAB/ZmkGlB7X+6IUp6vS vgU=

;; Query time: 2 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Nov 17 10:36:44 2009
;; MSG SIZE  rcvd: 602
```

DO(DNSSEC OK)ビット

- dig の +dnssec オプション
  - 問合せでEDNS0を使い、DOビットをONにすると共に512バイトを超えるサイズのDNSパケットを受けられることを宣言する
  - DNSSECでは**EDNS0のサポートは必須**
- DOビット
  - DNSSEC OK ⇒ DNSSECの応答を受ける
    ⇒ DNSSECを要求する
  - 権威サーバは、問合せのDOビットがONであれば、DNSSECの情報を含んだ応答を返す

時刻の同期

- DNSSEC運用を行う場合、サーバの時刻を正しく合わせる必要がある
  - NTPなどを利用するのが確実
- 署名には有効期限があり、期限を過ぎると無効
  - 署名が正しくてもサーバの時刻が極端に違うと、署名検証に失敗する
  - DNSSEC対応のゾーンは、定期的に再署名を行うため、有効期限は随時変更となる
- 実用上は、分程度まで合っていれば問題ない

鍵更新と再署名

DNSSEC

鍵更新

- 同じ鍵を長期間使い続けると、様々なリスクが生じる
  - 不注意、偶発的事故、鍵の盗難、暗号解読等
- リスクを最小に抑えるため、DNSSEC対応ゾーンの運用では定期的な鍵更新(鍵の交換)を行う
  - 例えばSE(スウェーデン)の場合、年に1回新しいKSKを生成し、2年間利用する運用を行っている

鍵更新時に留意すべきこと

- 鍵更新は、DNSSECの信頼の連鎖が途切れないよう、注意深く作業する必要がある
  - 鍵情報(DSやDNSKEY)と署名(RRSIG)はDNSのレコードである
    ⇒ キャッシュサーバはこれらを**キャッシュする**
- キャッシュしている情報と、あらたにキャッシュサーバが受け取る情報の整合性を確保する

www.example.jpのAの署名検証 (1)

① 上位からDSを受け取る
- JPの権威サーバからexample.jpのDS (DSの署名検証の解説は省略)

② 当該ゾーンのDNSKEYを受け取る
- example.jpの権威サーバから、example.jpのDNSKEY(複数)とRRSIG(複数)を受け取る

③ DNSKEYからKSKを識別する
- DNSKEYは複数(2個以上)存在するので、フラグが257のDNSKEY(1個以上)を識別する

④ KSKを特定する
- KSKとDSの鍵ID、暗号化アルゴリズムを比べ、KSKを特定

www.example.jpのAの署名検証 (2)

⑤ KSKを認証する

- DSのハッシュアルゴリズムに従ってKSKのハッシュ値を計算し、DSにあるハッシュ値と比較してKSKを認証する

⑥ DNSKEYを認証する

- ③で受け取ったDNSKEYに付随したRRSIG(複数)の鍵IDからKSKの鍵IDと一致するものを識別し、署名検証を行いDNSKEYを認証する

⑦ DNSKEYからZSKを識別する

- DNSKEYのフラグが256のものを識別する
  ここでZSKは複数存在する可能性がある

www.example.jpのAの署名検証 (3)

⑧ www.example.jpのAを受け取る
  – example.jpの権威サーバから、AとRRSIG(1個以上)を受け取る
⑨ www.example.jpのAを認証する
  – RRSIGの鍵IDと一致するZSKで署名を検証する

- 署名検証の際、署名の有効期間、ドメイン名など他のRRSIGのパラメータもチェックされる
- DSやDNSKEY、RRSIG等は、署名検証後もTTLの有効時間キャッシュする

ZSKの更新
事前に鍵を公開する手法 (1/2)

① DNSKEYに新旧のZSKを登録する
- 新ZSKを作成し、旧ZSKと共にDNSKEYに登録し(この状態でKSKを含めてDNSKEYは最低3個)、旧ZSKでゾーンを署名
- DNSKEYのTTL時間(+セカンダリの転送時間)待つ
- 全てのキャッシュサーバが新旧のZSKを含んだDNSKEYをキャッシュするようになり、旧RRSIGでも新RRSIGでも署名を検証できるようになる

② ゾーンの署名鍵を新ZSKに切り替える
- ゾーン内の最長のTTL時間(+セカンダリの転送時間)待つ
- 全てのキャッシュサーバから旧ZSKで署名したRRSIGが無くなる

ZSKの更新
事前に鍵を公開する手法 (2/2)

③ 旧ZSKをDNSKEYから削除する

– DNSKEYは新ZSKとKSKの状態になる

	初期状態	①	②	③
DNSKEY	KSK 旧ZSK	KSK 旧ZSK **新ZSK**	KSK 旧ZSK **新ZSK**	KSK **新ZSK**
RRSIG	旧ZSKでの署名	旧ZSKでの署名	**新ZSKでの署名**	**新ZSKでの署名**

ZSKの更新
2つの署名を使用する手法

① 新ZSKを作成し、新旧両方のZSKでゾーンを署名
  – ゾーン内の最大TTL時間(+セカンダリの転送時間)待つ
② DNSKEYのZSKの新旧を入れ替え、新ZSKでゾーンを署名

	初期状態	①	②
DNSKEY	KSK 旧ZSK	KSK 旧ZSK	KSK **新ZSK**
RRSIG	旧ZSKでの署名	旧ZSKでの署名 **新ZSKでの署名**	**新ZSKでの署名**

ZSKの更新
メリット・デメリット

- 事前に鍵を公開する手法
  - ○ ゾーンへの署名を2回行う必要が無い
  - × ZSKの公開時間が長くなるため、暗号解読攻撃のリスクが高まる(ZSKは鍵長が短い)
  - △ 初期状態から数えて4ステップ必要
- 2つの署名を使用する手法
  - ○ 初期状態から数えて3ステップで終了する
  - × ゾーンへの署名を2回行う必要がある
  - × 鍵変更期間中(①の状態)はDNSデータが大きくなる

KSKの更新
2つの署名を使用する手法(1/2)

① 新KSKを作成し、DNSKEYに登録して、DNSKEYを新KSKと旧KSKで署名する

② 親ゾーンのDS登録を旧から新に切り替える

- 親側のDSの切り替え作業を待ち、その後親側の旧DSのTTL時間分待つ

③ 旧KSKを削除する

KSKの更新
2つの署名を使用する手法(2/2)

	初期状態	①	②	③
親ゾーンのDS	旧DS	旧DS	**新DS**	**新DS**
子ゾーン DNSKEY	旧KSK ZSK	旧KSK **新KSK** ZSK	旧KSK **新KSK** ZSK	**新KSK** ZSK
子ゾーン DNSKEYの RRSIG	旧KSKでの署名 ZSKでの署名	旧KSKでの署名 **新KSKでの署名** ZSKでの署名	旧KSKでの署名 **新KSKでの署名** ZSKでの署名	**新KSKでの署名** ZSKでの署名

時間の流れ

KSKの更新
事前に鍵を公開する手法

① 新KSK(と新DS)を作成し親ゾーンに新旧2つのDSを登録する
  - 親ゾーンのDS登録を待つ、さらに旧DSのTTL時間待つ
② 旧KSKを破棄し、新KSKでDNSKEYに署名する
③ 親ゾーンのDS登録を新DSのみにする

KSKの更新
事前に鍵を公開する手法

	初期状態	①	②	③
親ゾーンのDS	旧DS	旧DS **新DS**	旧DS **新DS**	**新DS**
子ゾーン DNSKEY	旧KSK ZSK	旧KSK ZSK	**新KSK** ZSK	**新KSK** ZSK
子ゾーン DNSKEYの RRSIG	旧KSKでの 署名 ZSKでの 署名	旧KSKでの 署名 ZSKでの 署名	**新KSKでの 署名** ZSKでの 署名	**新KSKでの 署名** ZSKでの 署名

KSKの更新
メリット・デメリット

- 2つの署名を使用する手法
  - ZSKと違い、署名がDNSKEYにのみ作用するので、ゾーンデータの肥大化は問題にならない
  - 親ゾーンとのDSのやり取りが1回で済む
- 事前に鍵を公開する手法
  - 親ゾーンとDSのやり取りが2回必要となる

ゾーンの再署名

- 署名の有効期限が長すぎるのは望ましくない
  - 万が一の事態(鍵の盗難等)において速やかに対応するためには、署名期間は短いほうがよい
- 有効期限が数分の鍵も技術的には可能
  - しかし、休日の対応を考慮すると現実性に欠ける
- 署名の有効期限に達する前に署名の有効期限を更新するために、**ゾーン全体の再署名**が必要となる
  - DNSSECでは、再署名を行ってゾーン情報を定期的に更新する必要がある

NSEC3固有の問題

- NSEC3では、同じハッシュ値を使い続けると辞書攻撃により秘匿している情報が解析されるリスクがある
  - ゾーンの再署名時にソルトを変更し、ハッシュ値を変えるのが望ましい

鍵の有効期限

- 運用面での鍵の有効期限の実用的な値
  - KSK　13ヵ月　　12ヶ月で鍵更新
  - ZSK　～3ヵ月
- KSKの更新はDSの登録変更作業を伴うため、ドメイン名登録の更新にあわせるのが現実的
- ZSKはKSKのような制約は無く、ゾーン内で処理が完結するため、運用面での負荷を考慮しながら期間を短めに設定する

TTLと署名の期間

- RRのTTLが署名期間より長かったら
  - キャッシュしたRRの署名が無効になる事態が発生する
    ⇒ 署名の有効期間はTTLより長い必要がある
- SOAのExpireが署名期間より長かったら
  - セカンダリサーバでゾーンが有効にも関わらず署名が無効になる事態が発生する可能性がある
    ⇒SOAのExpireは署名期間より短い必要がある

鍵管理

- KSKが漏洩すると被害が大きい
  - ゾーンの重要度との兼ね合いで、rootやTLDの重要度はエンドユーザのゾーンに比べ高い
  - DS登録が必要なため、自ゾーンだけでは管理できない
  - HSM(Hardware Security Module)の利用を推奨
- ZSKはKSKに比べリスクは小さい
  - 万が一漏洩した場合でも、KSKに比べ簡単に更新できる
- いずれにしても鍵管理は十分厳重に行う

DNSSEC化による DNSデータの変化

DNSSEC

DNSSEC有無による www.nic.seの検索

- DNSSEC無し

```
$ dig +norec @a.ns.se www.nic.se a | grep SIZE
;; MSG SIZE  rcvd: 157      (親のNSに問合せ)
$ dig +norec @ns.nic.se www.nic.se a | grep SIZE
;; MSG SIZE  rcvd: 173      (自分自身のNSに問合せ)
```

- DNSSEC有り

```
$ dig +norec +dnssec @a.ns.se www.nic.se a | grep
  SIZE
;; MSG SIZE  rcvd: 414      (親のNSに問合せ)
$ dig +norec +dnssec @ns.nic.se www.nic.se a |
  grep SIZE
;; MSG SIZE  rcvd: 1180     (自分自身のNSに問合せ)
```

権威サーバへのdigの結果(1/2)

- +dnssec無しのdigの応答

```
; <<>> DiG 9.6.1 <<>> +norec @ns.nic.se www.nic.se a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14346
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 4

;; QUESTION SECTION:
;www.nic.se.                    IN              A

;; ANSWER SECTION:
www.nic.se.     60              IN              A               212.247.7.218

;; AUTHORITY SECTION:
nic.se.         3600            IN              NS              ns3.nic.se.
nic.se.         3600            IN              NS              ns2.nic.se.
nic.se.         3600            IN              NS              ns.nic.se.

;; ADDITIONAL SECTION:
ns.nic.se.      3600            IN              A               212.247.7.228
ns.nic.se.      3600            IN              AAAA            2a00:801:f0:53::53
ns2.nic.se.     3600            IN              A               194.17.45.54
ns3.nic.se.     60              IN              A               212.247.3.83

;; Query time: 328 msec
;; SERVER: 212.247.7.228#53(212.247.7.228)
;; WHEN: Tue Jul  7 23:39:18 2009
;; MSG SIZE  rcvd: 173
```

権威サーバへのdigの結果(2/2)

- +dnssec有り 各RRにRRSIG RRを加えたものが返る

```
; <<>> DiG 9.6.1 <<>> +norec +dnssec @ns.nic.se www.nic.se a
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5979
;; flags: qr aa; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 9

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.nic.se.                            IN          A

;; ANSWER SECTION:
www.nic.se.     60          IN          A           212.247.7.218
www.nic.se.     60          IN          RRSIG       A 5 3 60 20090714132001 20090704132001 58670 nic.se. izdsOhTB1XThccw2Wv4TZjl

;; AUTHORITY SECTION:
nic.se.         3600        IN          NS          ns2.nic.se.
nic.se.         3600        IN          NS          ns.nic.se.
nic.se.         3600        IN          NS          ns3.nic.se.
nic.se.         3600        IN          RRSIG       NS 5 2 3600 20090714132001 20090704132001 58670 nic.se. pKDbUYXLQpPnhlU9NAZh

;; ADDITIONAL SECTION:
ns.nic.se.      3600        IN          A           212.247.7.228
ns.nic.se.      3600        IN          AAAA        2a00:801:f0:53::53
ns2.nic.se.     3600        IN          A           194.17.45.54
ns3.nic.se.     60          IN          A           212.247.3.83
ns.nic.se.      3600        IN          RRSIG       A 5 3 3600 20090714132001 20090704132001 58670 nic.se. GzLodvUOd0oB4qfhhbp8H
ns.nic.se.      3600        IN          RRSIG       AAAA 5 3 3600 20090714132001 20090704132001 58670 nic.se. 0tvno8Vz7Ihm27AZ+H
ns2.nic.se.     3600        IN          RRSIG       A 5 3 3600 20090714132001 20090704132001 58670 nic.se. UcEcYGX59H8bAVGwhfwko
ns3.nic.se.     60          IN          RRSIG       A 5 3 60 20090714132001 20090704132001 58670 nic.se. NRoFeFzAm0hoyKa2ObxjCfB

;; Query time: 382 msec
;; SERVER: 212.247.7.228#53(212.247.7.228)
;; WHEN: Tue Jul  7 23:39:24 2009
;; MSG SIZE  rcvd: 1180
```

注意：RRSIGは行の途中まで、残り省略

DNSSEC有無による存在しないドメイン名の検索

- example.org DNSSEC無し

```
$ dig +norec @a0 example.org a | grep SIZE
;; MSG SIZE  rcvd: 77
```

- example.org DNSSEC有り

```
$ dig +norec +dnssec @a0 example.org a | grep SIZE
;; MSG SIZE  rcvd: 581
```

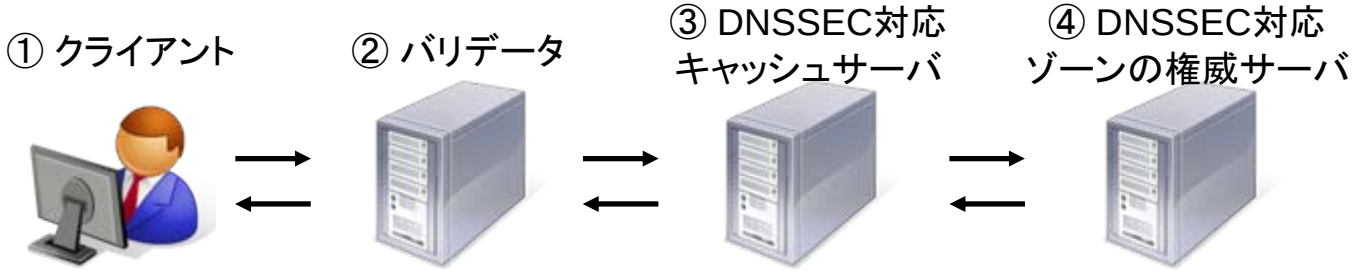
  - example.orgは存在しないドメイン名
  - 「a0」は.ORGの権威サーバの「a0.org.afilias-nst.info」を省略して表記

DNSSEC対応になると

- 署名の付加によりゾーンデータが大きくなる
  - 5～10倍程度(鍵のbit長に依存)
  - プライマリが署名すると、セカンダリにもインパクトがある
- DNS応答パケットのサイズが大きくなる
  - DNSトラフィックが増える
  - キャッシュサーバのキャッシュ効率が落ちる
  - 特に存在しない名前の検索では顕著
- DNSSEC対応のキャッシュサーバの実装では、**DNSSEC設定を行わなくてもDOビットはON**となる

何故DOビットは常時ONなのか

- キャッシュサーバ自身では署名検証を行わなくても、他の署名検証を行うもの(**バリデータ**)のために、署名があれば対象レコードと同時にキャッシュ



DNSSECにおいて署名検証が独立したモデル

②が署名検証を行うキャッシュサーバで③をフォワード先として指定している場合や、②は存在せず①のクライアントが直接署名検証を行う場合等

DOビットが常時ONのインパクト

- 問合せ1回あたりの応答パケットが大きくなる
  - DNSのトラフィックが増加する
- キャッシュサーバではキャッシュに必要なメモリ量が増える
  - 場合によっては、キャッシュできるレコード数が減り、キャッシュ効率に影響する
- 手元のDNSSEC設定とは関係なく、周囲のDNSSEC化の普及度によって影響する

DNSSEC関連技術

DNSSEC

DLV (DNSSEC Lookaside Validation)

- rootに署名が行われていない場合、DNSツリーに部分署名が複数あると、バリデータにトラストアンカーを複数登録する必要がある
  ⇒ バリデータの運用コストが大幅に増加
- DLVを利用すると、部分的な署名ツリーであってもトラストアンカーを複数設定する必要が無くなる
  - 参考資料
    http://dnsops.jp/bof/20090904/dnsops-20090904.pdf

RFC 5011
トラストアンカーの自動更新

- ゾーン管理者がKSKの鍵更新を行う
  - バリデータではトラストアンカーの更新作業が発生する
  - この作業を怠ると、署名の検証ができなくなる
- RFC5011 – トラストアンカーの自動更新
  - 新しい公開鍵をDNSKEYに登録し、既存のDNSKEYの公開鍵を無効にする
    ⇒ DNSKEYのフラグにREVOKEビットが追加
  - BIND 9.7系から実装
  - Unboundも一部実装

OpenDNSSEC

- DNSSECの運用に必要な作業の多くを自動化するツール集

  http://www.opendnssec.org/

  – 本原稿作成時点でβ7
- 今後DNSSEC運用ツールの主流になる可能性あり

DNSSECのまとめ

DNSSEC

従来(DNSSEC無し)との比較(1)

- ゾーン管理(権威サーバ)側
  - ZSKとKSKを作成し管理する必要がある
  - ゾーンに署名を行う必要がある
  - 定期的に鍵の更新を行う必要がある
  - 子ゾーンでは親ゾーンにDSの登録作業を行う必要がある(KSKを変更する度に必要)
- 鍵管理の手間と、ゾーン署名のコストが増大する

従来(DNSSEC無し)との比較(2)

- キャッシュサーバ側
  - DNSSEC機能を有効にし、トラストアンカーを設定する
  - 必要に応じてトラストアンカーを更新する
  - 署名検証による負荷の増大の懸念がある
- 双方でサーバの時刻を**正しく**設定する
  - 署名には有効期間があり、期間を過ぎると**無効**
    ⇒ **署名検証に失敗**する
  - NTP等を利用しサーバの時刻を自動的に同期

DNSSECまとめ

- DNSSECは、公開鍵暗号技術を利用した署名によるDNSデータ保護のしくみ
  - KSKとZSKの2つの鍵を使う
  - 親ゾーンにはNSに加えてDSを登録する
  - 理想的にはルートゾーンのKSKの公開鍵を使って署名を検証
  - 定期的な鍵の更新と再署名とが必要
- 現状、キャッシュへの毒入れ攻撃からキャッシュデータの汚染を防ぐ唯一の現実解

Q and A

JAPAN REGISTRY SERVICES