RFC 4641 プロトコル理解 SWG

Appendix A. Terminology

- Anchored key
  - 地表のレゾルバたちに設定されているDNSKEY。
  - この鍵は更新するのがhardである。ゆえにanchor(ed)なる語で表現される。
- Bogus
  - RFC 4033の5章も参照のこと。
  - あるRRSetに施された署名がDNSKEYで検証できなかったとき、そのRRSetはBogusと見なされる。

Appendix A. Terminology

- Key Signing Key あるいは KSK
  - Key Signing Key(KSK)とはzone apexに対応するkey setのみに署名するのに用いられる鍵である。
  - ある鍵がKSKであるという事実は、署名ツールだけが関心を示す事柄である。
- Key size
  - このドキュメントを通じて’key size’という言葉は’modulus size’と読み替えて差し支えない。
  - 数学的にはmodulus sizeという表現が、より正確だが、このドキュメントは運用者を対象としているので、key sizeという表現が平易である。
  - **modulus** —— n. (pl. moduli [-lai])【物】率, 係数; ＝ absolute value. (Infoseek マルチ辞書)

Appendix A. Terminology

- Private and public keys
  - DNSSECは公開鍵暗号方式を用いてDNSをセキュアにする。
  - 公開鍵暗号方式は、public key(公開鍵)とprivate key(秘密鍵)という2つの(数学的な関係のある)鍵の存在性を根拠としている。
  - public keyはDNSKEY RRによりDNSで公開される。
  - private keyはヒミツであるべきである。

Appendix A. Terminology

- Key rollover
  - key rollover(key supercessionと呼ぶ流派もある)とは、鍵の有効期限の終わりに、ある鍵ペアを別の物に交換する行為である。
- Secure Entry Point (SEP) key
  - 親ゾーンにその鍵を指すDSが存在するか、trust anchorとして設定されているKSK。
  - プロトコル上の要請ではないが、それらの鍵のSEP flag(RFC 4035参照)を立てておくことを推奨する。

Appendix A. Terminology

- Self-signature
  - DNSKEYに施された署名だけに関係する。
  - DNSKEY xに対してDNSKEY xで施した署名をself-signature(自己署名)と呼ぶ。
  - 注意
    - さらなる根拠が伴わなければ、self-signatureは信頼のよりどころとはならない。
    - 例えばハッシュを使ってDNSKEYの正しさを確認するのに役立つ。

Appendix A. Terminology

- Signing the zone file
  - 管理者がキモチいい音のパターンを奏でながら、楽しげに自分が管理しているゾーンファイルに署名するイベント。
- Signer
  - 秘密鍵にアクセスし、ゾーンデータ中のリソースレコード群に署名するシステム。
  - signerは、例えば署名の失効が近づいたRRSetだけ、など、ゾーンの一部だけに署名するよう設定してもよい。

Appendix A. Terminology

- Zone Signing Key(ZSK)
  - ゾーン中のすべて(たぶんDNSKEY RRSetを除いて)に署名するのに使われる鍵。
  - ある鍵がZSKであるという事実は、署名ツールだけが関心を示す事柄である。
- Zone administrator
  - ゾーンデータに署名し、それをprimary authoritativeサーバで公開する’role’(役職)。

RFC 4641

&

Internet Draft RFC4641-bis-02

Abstract

- このRFC4641では、DNSSECの鍵と署名について運用上の問題を議論している。
  - 議論の内容としては、以下のとおり。
    - 鍵生成
    - 鍵の収納
    - 署名の生成
    - 鍵のロールオーバー
    - 関連するポリシー
- DNSSECの運用に関する多くの内容を含んでおり、鍵のサイズとDNSSECの特性に合わせた最新の要件を述べられている。

- RFC4641はRFC2541(DNSのセキュリティについて鍵交換をベースに議論しているRFC)の内容をカバーしているため、RFC4641が標準化されるとRFC2541の内容は破棄される。

1. Introduction

- この文書読んでる人はDNSのRFC1034と1035の知識があることとDNSSECの開発がしたい人が前提だよ。
- で、RFC4033と4034、4035についても読んどいてくれ。

- このRFCの内容は、事前のworkshopや、わずかな実運用経験を通じて得たDNSSECのノウハウを、ゾーン管理者たちのためにまとめたもの。
- このRFCの公開後からの、DNSSEC実運用の中で、最適な手法が蓄積されているはずなので、この文書は、公開された時点でのベストプラクティスなのであって、今君らが読んでいる現在のベストではないよ。

- という前置きもあり、この文書は2006年公開で、実際に2010年現在、bits-2がInternet draftで提案されている。
- このプレゼン資料の中では旧来のものを黒字、Internet Draftで追加されたものを赤字、Internet Draftにて削除されたものを緑字で記載することで差分を比較しながら内容を紹介していく。

1. Introduction

- ここで紹介する手法は署名付きゾーンの保守方法に焦点をあわせている。
(つまり、権威サーバ上の署名(signing)や鍵発行(publishing))
- インターネット上の検証をおこなっているクライアントから再署名やキーのロールオーバーのような、DNSゾーンの運用が分かりやすいように意図している。

- それぞれ議論した内容と章立ては以下のとおり。
  - 2章は、"信頼の鎖"を維持する重要性について、
  - 3章は、主に秘密鍵について詳細な内容、
  - 4章は、公開鍵についての問題点、
  - 4.1については、公開鍵が発行された以後のあるゆるタイミング問題について
  - 4.2と4.3では、ロールオーバーへの対処と鍵の破棄について。
  - 4.4では信頼の鎖を維持するための子サーバの公開鍵に対する親サーバでの対処法について

- 文書中で用いられている紙面上の取り決めについては、付録Cにて説明しているのでご参考に。

1. Introduction

- このRFCで記述されている内容は、運用上の提案であってプロトコルには関連しない。
  - このためRFC2119は考慮しない。
    ※RFC2119はRFC内で使われている用語の意味を定義をしている文書。 「MUST」とか「MAY」とかの細かい意味合いを規定している。

- また、DNSSECプロトコルで発展した内容が反映されたため、RFC2541の内容が陳腐化する。
  - RFC2541(DNSのセキュリティについて鍵交換をベースに議論している)

- 暗号化アルゴリズム、DNSレコード、親子間の鍵と署名交換などの手法を選択したことにより、各項目の説明が大きく追加・書き直しされている。

1.1 The Use of the Term Key (「鍵」という用語について)

- このRFCの読者はDNSSECがベースとしている非対称の鍵の概念について精通しているものと仮定している。

- それゆえに、この文書では、「鍵」という用語をいくぶん大雑把に使っている。

  - 例えば 「鍵がデータを署名するのに使われた(a key is used to sign data)」と書かれた場合、「鍵のペアのうちの秘密鍵が署名に使われた」と読者が読み解いてくれるものと想定して書かれている。

  - そしてそれがまた「公開鍵がDNSKEY RRにて公表される」ということと、「鍵交換で使われるのは公開鍵である」ということでもあるとも読者の方で読み解いて欲しい。

1.2 Time Definitions (「時間」の定義)

- 文書内では「時間」に関連する用語を多用している。 それぞれは以下のとおりの意味である。

- ・「署名の有効期間(Signature validity period)」
  - ある署名が有効な期間を指す。有効期間は、RRSIG RR内の「署名開始」の項目で指定された時刻で開始され、RRSIG RR内の「期限満了」項目にて指定された時刻で終了する。

- ・「署名の公開期間(Signature publication period)」
  - 特定の鍵で作られた署名が、同じ鍵で作られた新しい署名で置き換えられた後の時間を指す。この置き換え処理は、マスターゾーンファイル内で適切なRRSIGを公開していることによって行われる。
  - あるゾーンであるRRSIGの公開が終了した後、RRSIGがキャッシュから消え、確実にDNSから除外される前に、少しの間時間がかかるかもしれない。

- ・「鍵の有効期間(Key effectivity period)」
  - 鍵のペアが有効であろうと期待される期間を指す。
  - この期間は、この鍵の使い方が断続的かどうかにかかわらず、その鍵で作成された全ての署名の、最初の開始日時のタイムスタンプと、最後の満了日の間と定義される。
  - 鍵の有効期間は、複数の署名の有効期間にまたがることができる。

- ・「最大/最小のゾーンTTL」(Maximum/Minimum Zone Time to Live (TTL))
  - あるゾーンにおけるRRの(完全なセット?)から定義されたTTL項目の最大か最小の値を指す。
  - ここでいう「最小TTL」がSOA RR内で記述された「MINIMUM」の項目とは同じではないことに注意すること。
  - 詳細は「11 」(RFC2308 DNS クエリのネガティブキャッシュ)の項目を参照のこと。

2. Keeping the Chain of Trust Intact (信頼の鎖の維持)

- 有効な信頼の鎖を維持することは重要。
  - なぜなら壊れた信頼の鎖は、bogus(ニセモノ)として認識されるデータをもたらすことになるだろうから。
  - (BogusについてはRFC 4033 、5章で定義している)
  - ※Bogusになると、それ以下の全体のドメインが、問い合わせをしてきているクライアントから見えなくなるかもしれない。

- 問い合わせをしてきているクライアントに対して、セキュアゾーンの管理者はその“セキュアゾーン”というのが、信頼の鎖の一部だと認識せねばならない。

- 序論でも述べたとおり、 再署名や鍵のロールオーバーのようなゾーンのメンテナンスの手法が、インターネット上の問い合わせをしているクライアントに対して透過的であることを意図している。

2. Keeping the Chain of Trust Intact (信頼の鎖の維持)

- セキュアゾーンの管理者は、 ある権威プライマリサーバ上で公開されたデータが、直後に問い合わせをしてきたクライアントには見えないことを留意しておく必要がある。

- それはデータが他のセカンダリのネームサーバへ転送されるまで多少時間を見ておく必要があるかもしれない。またクライアントはキャッシュ(non-authoritive)サーバからデータを取得しているかもしれないからだ。

- マスターからスレーブへのゾーン転送のための時間はnotifyとIXFRを利用したときにはわずかな時間だが、AXFR(フルサイズのゾーン転送)とnotifyを併用した際にはそのタイムラグは増加する。かつ、更新間隔をSOAのタイミングパラメーターだけで設定し、かつAXFRベースでの転送に頼っていた場合そのタイムラグはさらに増加する。

2. Keeping the Chain of Trust Intact (信頼の鎖の維持)

- 問い合わせをしているクライアントにとっては、データが直接に権威サーバやキャッシュサーバ、あるいはなんらかの中継箱(some middle box)から来ているかどうかに関わらず、セキュアなゾーンからの信頼の鎖を構築するために使うことのできるデータだということが重要である。

- クエリが利用可能になったタイミングのパラメーターを注意深く取り扱うことによってのみ、検証するために必要なデータを得られることを確実にする。

- 信頼の鎖を維持することの責任は、信頼の鎖に登場するセキュアゾーン内の管理者によって分け合われる。これ(責任を分かち合うこと)は信頼の鎖を維持することと、漏洩した鍵をできるだけ速やかに置き換えることとのトレードオフをしなくてはならない、いわゆる「鍵の漏洩」のケースの場合に最も明らかになるだろうね。そのとき、管理者は信頼の鎖の健全性を保つこと--それによって危殆化した鍵を利用した攻撃を許容するか、敢えて信頼の鎖を崩し、"Security-aware"なリゾルバーサーバから見えないセキュアなサブドメインを構築するのかのトレードオフを実行しなくてはならない。

- この信頼の鎖に関する詳細は4.3の節を参照のこと。

3. Keys Generation and Storage (鍵生成と保管について)

- このセクションでは鍵の安全性について尊重をもって記述する。
- 世代、有効期間、大きさ、そして秘密鍵の保管について扱っていく。

- ※ この3章はInternetDraft bits-02で章立てを大きく構築し直しているので、ここから現在のRFC4641との差分が大きくなります。

3.1 Zone and Key Signing Keys (3.1 ゾーンと鍵署名鍵[KSK])

- DNSSECの検証プロトコルは異なる種類のDNSKEYを区別しない。
- 全てのDNSKEYは検証する際 (the validation)に使われる。
- キーの種別を区別する動機(the motivations)が純粋に動作している。
- 実際に、DNSSECの運用者は実運用の中では運用者らに区別されている、鍵署名鍵(KSK)およびゾーン署名鍵(ZSK)、セキュアエントリーポイント(SEP)と呼ばれているものを利用しているよね。
- これらに対する原理と考察を下記にて議論する。(「運用上の理由から、下記の動機によって、」と記述修正)
- ゾーン再署名と鍵のロールオーバーの手順を作ることは、(※bits02では記述削除)一つあるいは複数の鍵署名鍵(KSK)鍵を使えば、簡単にできる。これらの鍵はゾーン内の一番上位のDNSSEC RRSetしか署名しない。他の鍵はゾーン内の全てのRRSetを署名するのに使われ、ゾーン署名鍵(ZSK)として参照される。
- この文書では、KSKは親との鍵交換に使われる鍵のサブセットであり、トラストアンカーとして設定する可能性があるSEP Keyだと仮定する。
- なお、この文書では、KSKとSEP keyの間は1対1に割り当てていると仮定する。また、SEP flagが全てのKSKに設定されていると仮定している。
- 共通の慣例がトラブルシューティングを容易にして以来、親サーバと鍵交換に使われ、あるいは潜在的にトラストアンカーとして設定される全ての鍵はKSKとして扱われ、SEPのフラグセットを持っていることを私たちは推奨してきた。KSKとZSKの区別がSEPフラグでは作られていないケースの場合には全ての鍵にSEPフラグをセットすべきだ。

3.1.1 Motivations for the KSK and ZSK Separations
（KSKとZSKの区分の動機付け）

- KSKとZSKの機能の区分することは、いくつかの利点を持っている。
  - ZSKが更新された時にサーバの親子間で対話が要求されない。
  - KSKはより強くなる。（言い換えれば鍵の素材にもっとbitを扱うようになっている）これはゾーンデータ内の小さな断片を署名するのに使われるのみだった運用に少し影響をもつだろうね。また、KSKはゾーンのキーセットの検証にのみ使われ、ゾーン内の他の鍵のためには使われない。
  - KSKはキーセットを署名するためにのみ使われ、ゾーン内の他の鍵よりも頻繁には更新されないであろう。したがってZSKよりも安全な場所に区分して保管するべきだ。
  - ZSKと区分することで、KSKは長い有効期間を設定することが出来る。

3.1.1 Motivations for the KSK and ZSK Separations
（KSKとZSKの区分の動機付け）

- KSKとZSKの区別は運用上の目的のためが大半の理由で行われる。
- 鍵管理とゾーン署名のほとんど全ての手法において、KSKはZSKよりも頻繁に利用しない。ひとたび、ある鍵セットがKSKによって署名されると、全ての鍵セットの中の鍵はZSKとして使用することが出来る。もしZSKの状態が悪くなっているなら鍵セットから単純に破棄することが出来る。新しい鍵はその時にKSKを使って再署名される。
- SEP機能をもった鍵の交換は第三者との対応が関わるという点において比較的人的コストが高くなる。
- 鍵が親のDSによって指し示されているのみなのであれば、親がレコード登録の対応を完了してDNSのレコードとして出現するのを待つだけでよい。
- トラストアンカーの鍵として使われるSEP鍵の場合、全てのトラストアンカーが交換し終わっているという十分な信頼をもてるまで待たなくてはならない。実際には鍵のロールオーバーについての情報をもった完全なユーザーベースには到達できないかもしれない。
- KSKのためにSEPのflagが設定されていると仮定して、KSKはDNSKEY RR のflag項目を検証することでZSKと区別することができる。もしflag項目が奇数な数値だったならそれはKSKである。もし偶数な数値ならばそれはZSKである。（※Internet Draftで別項目に記述移動）
- ZSKは定期的にゾーンの中の全てのデータの署名に利用できる。。（※Internet Draftで別項目に記述移動）
- ZSKが更新されるとき、親サーバとのやり取りは必要ない。これはその場にふさわしい署名の有効期間によって許可されている。。（※Internet Draftで別項目に記述移動）

3.1.1 Motivations for the KSK and ZSK Separations
(KSKとZSKの区分の動機付け)

- 鍵が盗難や紛失により危険に晒されることはまたひとつのリスク。
  - 直接的に更新されるレコードを署名するのに必要で、そのためにネットワークに接続されたネームサーバのファイルシステム上にインストールされている鍵のリスクは比較的高い。
  - ハードウェア署名モジュール上に保管されている鍵のリスクは比較的低い。
  - KSKとZSKの機能を分けることによって、発生する負荷に対してのトレードオフと、これらのリスクの管理を両立できる。

例えば、ゾーンデータの追加、更新のような運用上の目的のために容易に利用できる必要があるZSKよりもKSKはオフラインかさらにアクセス権限が限定的な場所に保存することができる。

さらに具体的には安全性が確保されたスマートカードの中にKSKを、日々のルーティンによってアクセス可能なファイルシステム上にZSKが保管されていると、KSKとZSKがHSM(ハードウェアセキュリティモジュール)上に一緒に保管されているよりも運用上でより柔軟でより高い能力を供給するかもしれない。

3.1.1 Motivations for the KSK and ZSK Separations
(KSKとZSKの区分の動機付け)

- 鍵署名鍵(KSK)はゾーン内のDNSKEY RRを署名するのみに使われる。もしKSKをロールオーバーする時には、そのゾーン管理者以外の他の団体と連携することになるだろう。
- これらの連携する対象にはセキュアエントリポイント(SEP)として設定されている特定の鍵を持つ親ゾーンの登録管理団体、あるいは名前解決の検証をしているリゾルバサーバの管理者が含まれる。

- そのために、これらの鍵の有効期間はさらに長くするべきであり、実際長くすることが可能である。もっとも十分な長さの鍵を指定し、複数年単位の有効期間の鍵だとしても、我々は鍵のロールオーバーが運用のルーティンワークに残るようにするために、数ヶ月単位の有効期間と計画することを提案するけどね。

3.1.1 Motivations for the KSK and ZSK Separations
（KSKとZSKの区分の動機付け）

- 最後は、鍵の素材の暗号解読のリスクについて
  - そのような解析をするための負荷は、鍵の長さと、分析者にとって利用可能な署名付きの素材の量に相関する。
  - 署名の速度とDNSパケットの長さの理由で、最小責任程度の鍵の長さを維持して、比較的頻繁に鍵を交換する一方で親サーバとの対話がない。
  - このような場合が、KSKとZSKを分離するための大きな動機となる。
  - 脆弱性を暴露されるリスクが低い場合、ここまでのZSKとKSKの区別のための弁論は弱くなる。
    - 例えばKSKとZSKが最大限の運用上の柔軟性のために許可されている、分けられたHSMに鍵が保存されている場合などでは根拠としては薄い。
    - それゆえにKSKとZSKの区分はつくられて、SEPフラグが排他的にKSKにセットされることを助言する。

3.1.2 KSK for High-Level Zones (ハイレベルゾーンのためのKSK)

- 高いレベルのゾーンは一般的に低いレベルのゾーンよりも更に繊細である。そのサブドメインの全ての権威を得ることによって、誰もがゾーンのセキュリティ(安全性)を操作したり破壊する。(サブドメインの公開鍵をローカルに設定しているリゾルバの場合を除いて、この場合、そしてこれだけ、サブドメインは親ゾーンが危険に晒されている(compromised)影響をうけることがない。)
- それゆえに高いレベルのゾーンではより一層の配慮をとるべきであり、そして強度の高い鍵を使うべきである。

- ルートゾーンは全てのゾーンにおいて最も重大なゾーンである。誰かがルートゾーンのセキュリティを操作するか侵害することは、ルートゾーンを使っている全てのリゾルバのDNSの名前空間を制御することになる。(サブドメインの公開鍵をローカルに設定しているリゾルバの場合を除いて)。 したがって、ルート ゾーンのセキュリティ保護に最大限の注意をしなければならない。最も慎重に処理される最強のキーを使用するべきだ。 ルート ゾーンの秘密鍵は常にオフラインで保持する必要がある。

- 多くのリゾルバーはルートサーバへのアクセスとDNSデータの認証を開始するだろう。世界中のリゾルバーという巨大な個体数を請け負っているトラストアンカーの安全な更新は非常に難しいだろう。

3.1.2 KSK for High-Level Zones (ハイレベルゾーンのためのKSK)

- 前のスライドの3.1.2はInternet Draftでは破棄されている。
  Internet Draft 3.1.4.5でその理由が書いてある。

3.1.4.5 高レベルゾーンの差異について

- この文書の以前のバージョンだと、DNSヒエラルキーの高層ゾーンのためのKSKと低層のゾーンのためのKSKについて区別を論じていた。

- 全ての人が誰も壊すことの出来ない鍵を使うべきだという暗号化指導の内容だったが、長い鍵はもはや有用ではない。
- また、どのゾーンが攻撃者に対して価値が高いか低いかの判断は不可能。
  - つまり、もし(経路上の)中間物が見過ごされ、かつ攻撃者が見過ごされた経路で中間者攻撃(MITM)ができるのならば、単に攻撃に利用するのみだ。
  - 例えば、「exsamle.」というドメインが侵害されて、攻撃者が「somebank.exsamle.」のDNS問い合わせ回答を偽造し、中間者攻撃としてそのレコードを送りつけた場合、攻撃が検出されると「exsamle.」が侵害されていて、KSKがロールバックされていることを証明するのが簡単になる。
- 長い期間成功しうる攻撃の定義は、全てのレベルの鍵において難しい。

ということで削除した。

3.1.2 Practical concequences of KSK and ZSK Separation
（実際の KSK とZSKの区分の結果）

- KSKにはSEPのフラグが設定されているものと仮定すると、DNSKEY RRのSEPフラグの項目を確認することで、KSKはZSKと区別することが出来る。（※RFC4641 3.1.1から項目移動してきた）
- SEPフラグのフィールドが奇数ならその鍵はKSK、偶数ならばZSKである。（※RFC4641 3.1.1から項目移動してきた）

- ZSKは定期的にゾーン内の全てのデータを署名することができる。（※RFC4641 3.1.1から項目移動してきた）
- ZSKがロールされるとき、親サーバとのやりとりは発生しない。これは署名の有効期間のために許可している。（※RFC4641 3.1.1から項目移動してきた）

- KSKはゾーン内のDNSKEYのRRを署名するためだけに使われる。
- KSKがロールオーバーするなら、そのゾーンの管理者よりも別の団体と対応することになるだろう。
- 対象が親のゾーンならば、これらは親ゾーンの登録管理者か、SEPとして設定されている特定の鍵を持っている検証をしているリゾルバーの管理者かを含んでいる。
- 対象がトラストアンカーならば、トラストアンカーを信頼している全ての人が新しい鍵にロールオーバーすることを必要としている。
- 後者は(RFC5011のような)自動的にトラストアンカーのKSKを更新してくれるメカニズムが無ければ、一定の負担に左右されるかもしれない。
- それゆえに、これらの鍵の有効期間はもっと長く生成できるし、そうするべきだ。

3.1.2.1 Rolling a KSK that is not a trust-anchor
（トラストアンカーではないKSKのロールオーバについて）

- トラストアンカーではないKSKのロールオーバーには3つの考え方がある。
  - キーのロールオーバーを定期的な運用処理に残るように頻繁かつ定常的な(場合によると2ヵ月ごと)に行われるべき。
  - 頻繁だが、不規則に行われるべき。ここでいう「頻繁」とは数ヵ月ごとを意味していて、ロールオーバ作業は実践すること、および一般的な運用処理であるべきという議論を軸にしているけども、実際は、作業は不規則なタイミングつまり第三者がKSKの信頼処理を始めていないのに、トラストアンカーとして設定することを誘惑されないようにするため、大きな揺らぎ(余裕かな?)をもっているべき。
  - 鍵が危険におかされているという強い疑いがあるか、そうであると確認されている時にのみ行うべき。 鍵のロールオーバーが正常に行われなかった際に、システムの安定性にかかわる問題が生じる確率を削減するために行う。

3.1.2.1 Rolling a KSK that is not a trust-anchor
（トラストアンカーではないKSKのロールオーバについて）

- DNSSECのそれぞれの展開のために、これらの3つの考え方のどれがより良いかことに広範な合意はない。
- アンカーではないKSKにはロールオーバごとに一定した負担があり、しかしその負担は親サーバと子サーバ間の関係が良好ならば、そのコストは低い可能性がある。
- 一方で、定期的にロールオーバー処理のテストすることのみが、関係が良い状態であることを確認をする完全に効果的な方法である。
- 従って、親とのKSKをロールオーバさせることは二つの理由でのみ行われる。
  - 一つは緊急時に備えてテストし、システムの動作を確認するため。
  - そしてもう一つは実際に緊急時の場合である。

- 最後に ゾーンの管理者はほとんどの場合、ゾーンのKSKがトラストアンカーとして使われていない確証を満足に得ないだろう。ゾーン管理者には責任がないトラストアンカーの設定にもかかわらず、ゾーン管理者がKSKを更新したときに検証する管理者がトラストアンカーを間違って設定していたら、ゾーン管理者に一定の負担がかかるかもしれない。

3.1.2.2 Rolling a KSK that is a trust-anchor
（トラストアンカーであるKSKをロールオーバーする場合）

- 前項と同じ運用上の懸念が、トラストアンカーとして使われてるKSKをロールオーバーする際にも適用される。

  トラストアンカーのロールオーバーを間違って行ったとき、そのトラストアンカーが包括しているゾーン全体は、トラストアンカーが正しく修復されるまでbogusとなってしまう、ってことを思い出してね。

- トラストアンカーの全ての利用者との理解を得ることの難しさのため、古いトラストアンカーを新しいトラストアンカーと取り替えることは論ずることが出来る。
- その鍵が危険に冒されているという強い疑惑があるかそうであると認知されている場合以外に、トラストアンカーのKSKはロールオーバーしない。

- 全ての利用者に古いトラストアンカーを新しいトラストアンカーと取り替えさせることの難しさのため、その鍵が危険に冒されているという強い疑惑があるかそうであると認知されている場合以外に、トラストアンカーのKSKはロールオーバーするべきではない。という主張をすることが出来る。

- 言い換えれば、何人かのユーザには連絡が出来なかったりするので、鍵のロールオーバーの負担はバカ高いってこと。

3.1.2.2 Rolling a KSK that is a trust-anchor
（トラストアンカーであるKSKをロールオーバーする場合）

- しかしながら、運用習慣の議論は、クライアントの検証上でのトラストアンカーの再設定にも適用する。
- 短い有効期間で利用されているなら、設定が忘れられる可能性は小さいが、トラストアンカーは定期的にその正常性を確認しないといけない。
- 実際に、それらのユーザのための負担はRFC5011にある自動ロールオーバーによって最小化するかもしれないし、定期的に鍵をロールオーバーすることで、再帰的ネームサーバの運用者は一定の負担に対処するためにそのような広報をするなどの適切な手段を講じるか、言葉を言い換えれば、予期せぬ危険を被る代わりにそのコストを予算化しておきなさい。
- それゆえにもしこれらのロールオーバーがRFC5011の自動メカニズムを使っているのならば定期的にKSKの交換をすることを推奨する。
- 検証するクライアントの管理者が変更に対応できないほど激しく変更のかかるシステムから守るために、適切な長さの鍵の有効期間を選択することが最後に残されたトレードオフである

3.2 Key Generation (鍵の生成)

- すべてのキーを注意深く生成することは、暗号的に安全なシステムでは見落とされることもあるが、絶対に不可欠の要素である。
  - もし攻撃者が徹底的な検索をするのが可能な程度の鍵空間のサイズの小ささを十分推測できる場合には、最長の鍵を利用した最も強力なアルゴリズムですら役に立たない。
  - ランダム鍵の生成するための技術提案はRFC4086と「決定的なランダムビットジェネレーターを利用したランダム数値の生成のため推奨事項(2007年)」(※InternetDraftで記述追加)に記載されている。このRFC4086の提案に従って、鍵の生成にランダムな数値の生成を利用するならば、慎重に判断するべきだろう。

- 長い有効期間をもっている鍵は、短い有効期間の鍵よりも長い期間攻撃の対象という意味で、つまり、もっと価値の高い標的を象徴するものという意味でも特に繊細である。ネットワークから(“空気を経由した”)別な場所か、最低でもハイレベルに安全性のあるハードウェアといった、隔離された手法で長い有効期間の鍵を生成することを強く推奨する。

3.3 key Effectivity Period (鍵の有効期間)

- 一般的に、使用可能な鍵の長さは、鍵の有効期間の上限を設定する。
- 全ての実践的な目的のために、純粋に運用上の要求に基づいて、鍵の有効期間を定義して鍵の長さをそれに合わせれば十分である。
- 運用の視点を無視すれば、親ゾーンをもつKSKの合理的な有効期間は20年以上が妥当なところだ。
- ロールオーバーの手順をテストする計画がなければ、鍵は永遠に有効なままにして、緊急な事態が生じたときにだけロールオーバーするべきである。

3.3 key Effectivity Period (鍵の有効期間)

- 様々な理由で、DNSSECの鍵はしばらくすると一度交換する必要がある。より長い鍵が使用されていても、注意不足、事故、スパイ行為や暗号解読を通じて鍵が脆弱性に晒されることは非常に小さな確率では存在する。

- 更に、鍵のロールオーバーが運用上でとっても稀な定期イベントであるとき、ロールオーバーが運用の習慣の一部になっていないであろう。そして実際に鍵のロールオーバーが必要になったときに誰もロールオーバーの手順を覚えていない場合、そこにリスクが存在する。

- 純粋な運用上の展望から、鍵署名鍵(KSK)の妥当な有効期限は、12ヵ月後に置き換えられるという意味から13ヶ月であるといえる。鍵の有効期間を1ヶ月増とするのはZSKのためには合理的である。

- この有効期間と一致する鍵のサイズについては3.5を参照すること。

3.3 key Effectivity Period (鍵の有効期間)

- 前述のとおりのやり方で、この年に一度のロールオーバーは、認証を行う管理者とそのゾーンの両方のためにロールオーバーの運用に慣れさせる機会になる。
- さらに多くの環境における一年とは、計画しやすく、対応しやすいタイムスパンである。
- もし紛失や盗難、あるいは別のなにかの危険に晒されることのリスクがZSKとKSKとで同等なら、異なる有効期間を選択する根拠は薄くなる。(KSKとZSKの機能の区分はまだ利点になるけども)
- 鍵がオンラインのシステムに保管されていて、その様々なリスクに晒されることがかなり高い場合、一ヶ月という意図した鍵の有効期限は、ZSKのために合理的だ。

3.3 key Effectivity Period (鍵の有効期間)

- 3.1.2での議論として、トラストアンカーの安全な更新処理は非常に難しいであろうと論じてきた。 一方、「運用上の習慣」の議論はトラストアンカーの再設定には適用されない。 もし短い鍵の有効期間が利用され、トラストアンカーの設定の機会が定期的に訪れるなら、設定を忘れてしまいがちな確率もより小さくなるだろう。でもその分、検証するクライアントの管理者が変化に付いて行けないであろうほどの動的なシステムに対してのトレードオフである。

- 鍵の有効期間は例えば数分間くらいにとても短く生成できる。しかし、鍵を交換する時については、4.1章と4.2章から考察をするべきだね。

- KSKの鍵の有効期間よりもZSKの有効期間を短くしている動機は、運用者がKSKよりもZSKへ頻繁にアクセスするということが多いという運用上の事情に根付いている。

- もしZSKが異なる鍵の有効期限をもつための動機よりも優先して暗号化されたHSMの上で置かれたままならば、それは無力になるだろう。

3.4 Key Algorithm(鍵のアルゴリズム)

- 現在、DNSSECでは3つの異なる種類のアルゴリズムが使われている。RSA、DSA、楕円曲線暗号(Elliptic Curve Cryptography)である。 一番後者のものは比較的新しく、DNSSECで利用するためにまだ標準化する必要がある。

  → Internet DraftではRSAとDSAの二種類とされた。

- RSAはオープンで分かりやすい方法で開発されている。2000年にRSAの特許は期限が切れ、今は自由に使うことができる。
- DSAはNIST(アメリカ国立標準技術研究所)で開発されている。署名の生成時間はおおよそRSAでの生成と同じだが、検証処理は10～40倍遅い。

- 両種類とも、多くの自由に利用できる文書が存在しており、特許無料とするためによく考えられている。
- RSAとDSAを使った署名の生成は大雑把には同じくらいだが、DSAは10倍くらい署名の認証に時間がかかる。

3.4 Key Algorithm(鍵のアルゴリズム)

- 我々はRSA/SHA-1を鍵のための優先的アルゴリズムとして提案する。RSAの既知の攻撃手法は鍵を長くすることで防ぐ(負かす)ことが出来るだろう。MD5でハッシュされたアルゴリズムはクラックする手法が出現しているため、SHA-1を利用することを推奨する。

- → 私たちは優先的アルゴリズムとしてRSA/SHA-256を、代替策として RSA/SHA-1を使うことを推奨する。両方とも長所と短所がある。RSA/SHA-1は何年も前に開発されたし、RSA/SHA-256はつい最近開発されたばかりだ。一方で、両方の暗号方式に対して、効果的な攻撃をしたなら、最初に解読されるはRSA/SHA-1のほうだろうね。RSA/MD5は攻撃対象として最初に試す格好の餌食だから使用するのは辞めるべきである。

- 公開された時点で、SHA-1は暗号解読について問題を持っていることはよく知られている。これらの問題は解決にむけてまだ進行中。公開鍵のアルゴリズムのベースはSHA-1よりもより強度のあるハッシュのもの(例えばSHA-256)を、プロトコルの仕様および実装において利用可能になったらできる限り早く利用することを推奨するよ。

- → 最初に公開したときに、SHA-1のhashは暗号化に問題があると知られていた。その問題の周知活動は現在進行中である。 ということで、SHA-1よりも強いhashをベースにしたアルゴリズムの公開鍵(SHA-256など)をDNSのプロトコルで規定および実装され利用できるようになったらすぐにでも利用することを我々は推奨するよ。

3.5 Key Sizes(鍵のサイズ)

- 鍵のサイズを選択するとき、ゾーン管理者は、どの位の長さの鍵が使われるであろうか、どの位の量のデータが鍵の公開期限の間に署名されるのであろうか(※ [17](1996出版「Applied Cryptography」) の8.10章を参照すること)、そして場合によっては親の鍵のサイズはどの位の大きさなのか、考慮する必要がある。

- 信頼の鎖が本当に「一つの鎖」としてなら、その鎖の中の鍵の一つを生成するのに数回他のよりも大きいことはあまり意味がない。

- これは常に鎖全体の強さを定義した、もっとも弱い繋がりになる。違う役割を提供する鍵(ZSK vs KSK)が鍵のサイズの差異を必要とするかもしれないことについて議論した3.1.1章を参照すること。

3.5 Key Sizes(鍵のサイズ)

- 正しいサイズの鍵を生成することは難しい問題だ。RFC 3766ではこの問題に取り組んでいる。
- RFC 3766 stateのセクション1で冒頭部分では、
  - 1. アプリケーションのセキュリティ要件を満たすために必要な攻撃への抵抗力を決定する。攻撃者がシステムの安全性を弱めるためにせざるを得ないコンピューター操作の最小数を予測することによって決定され、その対数として2つの数値を採用する。この対数をnと呼ぶことにする。
  - 1996年の報告書では、システムセキュリティのための良い選択として90bitsを推奨していた。90bitの数値は、3年につき2bit程度増加するべきだ。つまり2005年には96bitsであるべきだ。 ※つまり2010年には100bit程度?!

としている。

3.5 Key Sizes(鍵のサイズ)

- [13]では、この数値nが公開鍵の鍵サイズの暗号化を計算するのにどのように使われるのかを説明している。これは左の表に結論している。(多少、我々の目的のために修正している)

- 鍵のサイズは幾分大きくなっているけども、これらの鍵が兆億長者の攻撃者に抵抗するためだ。
- 生意気でリッチな攻撃者でも、私たちが推奨事項とする以下のサイズ(低レベルドメインでは1024bits 、中レベルでは1300bits、高レベルでは2048bits)に従って、一年に一度ロールオーバーしたあなたのKSK鍵を攻撃しないだろう。

System requirement for attack resistance (bits)	Symmetric key size (bits)	RSA or DSA modulus size (bits)
70	70	947
80	80	1228
90	90	1553
100	100	1926
150	150	4575
200	200	8719
250	250	14596

3.5 Key Sizes(鍵のサイズ)

- 対象のドメインが、低レベルか、中レベルか高レベルかどうかは単にそのゾーン管理者の主観次第だ。たとえば、あるDNSゾーンのサブドメインは低価値だろうし、TLDsやroot zoneは高価値と言える。提案した鍵のサイズはこれからの5年間のために安全なサイズのはずだ。

- ZSKはもっと簡単にロールオーバーし(そしてもっと頻繁に)、鍵のサイズはより小さく生成する傾向がある。しかし、この項での冒頭で言ったように、ZSKを生成する時の鍵サイズが小さすぎるのは、(KSKのサイズとの関係で)あまり意味を為さない。 100bitsの鍵サイズとの違いの限界に挑戦してみるといい。

- 誰も未来を見たことがないし。これらの鍵サイズはガイドラインとして提示されているのみだということを記載しておく。詳細な情報は、注釈[16]と7.5章の注釈[17]に記載されている。注釈[16]が鍵サイズが安全だと定義してとても楽観的に考察していることも記載しておくべきだろう。

3.5 Key Sizes(鍵のサイズ)

- 鍵のサイズについての最後の記載は下記のとおりだ。より大きな鍵はRRSIGのサイズとDNSKEYの登録レコードを増加させるだろう。そしてそれゆえにDNS UDP パケットのオーバーフローの機会が増加するだろう。
  また、RRSIGを認証したり生成するための時間は、より大きな鍵を用いることで増加することだろう。だから、君の鍵も2倍のサイズは必要ないよ。

3.5 Key Sizes(鍵のサイズ)

- DNSSECの署名鍵は鍵が有効な間、全ての知られ得る暗号解読攻撃に抵抗するのに十分な大きさであるべきだ。
  - 現在までに、多大な努力にも関わらず、誰も1024bitの鍵を破れずにいる。
    実際に最も完璧な攻撃でも700bitの鍵と同等だと推測される。
  - 攻撃者が1024bitの署名鍵を壊すには、一個の鍵を破るために発見されていない手法で、驚くほどのネットワークコンピュータパワーの量のコストが必要だろう。
  - このため、最高のゾーンは少なくともあと10年は1024bitの鍵を利用することで安全に保てるだろう。1024bitの不均等な鍵は対称80bitの鍵とほぼ同等の強度がある。

- 究極的に高価値なトラストアンカーの鍵や、あるいはロールオーバーは難しい鍵は、1024bitより長い鍵を使いたいかもしれない

  一般的に、1024bitの次のサイズは2048bitで、対称(暗号(5/27))の112bitの鍵と同等の強度だ。標準的なCPUでは、署名あるいは検証のために1024bitの鍵の4倍の時間かかるだろう。

3.5 Key Sizes(鍵のサイズ)

- 鍵のサイズを決めるためのもうひとつの手法は、攻撃者が1024bitの鍵を破るための驚くほどの労力が鍵の使用用途に関係なく等しいことを思い出すことだ。
  - 攻撃者が1024bitを破る能力をもっていたなら、彼はまた、webブラウザにインストールされた多くの1024bitのTLSのトラストアンカーのひとつも破る能力がある。
  - 攻撃者にとって、あるDNSSECの鍵の価値がTLSのトラストアンカーの価値よりも低かったなら、攻撃者はそのDNSSECの鍵を攻撃するリソースをTLSのトラストアンカーを攻撃するために使うだろうね。

- 鍵を壊すための攻撃能力が予想外に向上する可能性はあるし、そのような攻撃なら1024bitの鍵を壊すことも考えられるが、2048bitの鍵はそうもいかないだろう。
  - もしそのような攻撃能力の向上が起こるなら、膨大な量の告知がされるだろうね。
  - なんてったって、多数の1024bitのTLS トラストアンカーが、人気のあるwebブラウザに組み込まれているからね。
  - その時には全ての1024bitの鍵は親ゾーンの鍵だろうがトラストアンカーだろうがなんでも、より大きなサイズの鍵に交換しなくてはならないだろうさ。

- 前のバージョンの文書では、頻繁に使われる特定の鍵は長い鍵を使うことを促していた。
  - あの助言は15年前にはそうだったかもしれないけども、RSAとDSAのアルゴリズムを利用して1024bit長かそれより長い鍵が使われている**今日では正しくはないね。**

3.6 Private Key Storage(秘密鍵の保管場所)

- 署名されたゾーン秘密鍵とゾーンファイルのマスターコピーは、可能であれば、オフラインで、ネットワーク接続のない、物理的に安全な機器上でのみ保存されることが推奨される。
- 定期的に、RRSIGとNSEC RRを加えたことによってゾーンの承認を追加するためにアプリケーションが起動することがある。そのとき、レコードが増加したファイルは転送されることがある。

- 署名されたゾーンの管理を動的更新処理に依存した場合、少なくとも、ゾーン内の一つの秘密鍵がマスターサーバ上に存在しなくてはならないことに注意してほしい。
- この鍵はサーバが知らないクライアントに対する露出の度合いと、そのホストの安全性と同じくらいにしか安全ではない。

3.6 Private Key Storage(秘密鍵の保管場所)

- 必須ではないが、次のような手法でDNSを管理できる。
  - 動的更新を処理するマスターが、インターネット上の一般のホストから使用できなくし、SOA RRsのMNAME の項目には記載されているのに、NS RRsetに記載されていないようにする。
  - NS RRSetにあるネームサーバは、NOTIFY、IXFR、AXFR、あるいはある外向きに送出されるメカニズムを通じてゾーンの更新を受信することが可能である。
  - このアプローチは“隠されたマスター”の構築手法として知られている手法だ。
  - 理想的な状況は、ネットワークから改ざんされる可能性を避けた一方向の情報フローのネットワークを持つことである。ネットワーク上のオンラインのゾーンマスターファイルを維持し、シンプルにオフラインでの署名をしたものを引き渡すことは実現できない。
  - ホストに漏洩・侵害の可能性が存在する場合には、このオンラインバージョンは、まだ改ざんされる可能性がある。最高の安全性のために、ゾーンファイルのマスターのコピーはオフネットにするべきだし、安全ではないネットワークを介した通信上で更新するべきではない。

3.6 Private Key Storage(秘密鍵の保管場所)

- リスクとコストの間の経済バランスをとるために、理想的な状況は実現しないかもしれない。例えば、ゾーンをオフラインに保ち続けるのは、実際的ではないしDNSゾーンを運用するコストを増加させることになるだろう。だから実際に、ゾーンファイルが置かれている機器はネットワークに繋がっていることだろう。
- 署名したDNSデータの改変を防ぐために、マスターコピーへの不正なアクセスを防ぐためのセキュリティー対策をとることを助言しておく。
- 同様に、HSMに秘密鍵を収納する選択は、様々な要因とのトレードオフによる影響を受けるだろう。
  - 承認されていない人が秘密鍵を気づかれずに読みとるリスク
  - 攻撃者のために開きっぱなしで隙があるウィンドウ
  - 可能な攻撃の経済的影響
    (多くのケースで個人のユーザよりもTLDが攻撃された時の影響はコストが高くなるだろう)
  - (危険に晒された)鍵を交換するコスト:
    この場合、ZSKの交換するコストは最も低く、
    トラストアンカーとして広く使われているKSKを交換するコストは最も高い。
  - HSMを買って、維持するためのコスト

- 動的更新されるDNSSECゾーン(RFC 3007参照)のために、RRを更新する際に署名するのに使われるマスター鍵のコピーと秘密鍵はオンライン上に存在している必要があるだろう。

3.6 Private Key Storage(秘密鍵の保管場所)

- 一般的に、ゾーンファイルをオフラインに維持することは実践的ではないだろうし、メンテナンス対象のゾーンファイルをもつ機器はネットワークに接続しているだろう。マスター コピーへの不正アクセスを保護するセキュリティ対策を講じることを運用者にお勧めする。

- 安全なゾーンを動的更新処理をするために、RRを更新する署名に使われたマスターコピーと秘密鍵は、オンラインにあることが必要とされるだろう。

4. Signature Generation, Key Rollover, and Related policies

4.1. Time in DNSSEC

- DNSSECを使わなければ、DNSにおける時間は、すべて相対的。
  - REFRESH、RETRY、EXPIRATION、minimum TTL、(RRの)TTL
- 署名の有効期限の概念が登場したので、DNSSECはDNSに絶対時刻を導入した。
  - 過ぎると署名は無効になり署名されたデータはbogusになる(検証に失敗する)。
- draft-morris-dnsop-dnssec-key-timingで、さらに実質的な議論。

4.1.1. Time Considerations

- 署名の失効という概念に関係して、以下のことを考慮すべき。

4.1.1. Time Considerations

- ゾーンデータに登場する最大のTTLは署名の有効期間のfraction(小数、分数???約数では?(5/27))にすることを提案する。
  - 桁レベルで小さくしろ、という意味らしい。
  - 整数分の1で余りが出ないように、ということでは?(5/27)
  - TTLが署名の有効期間と同じオーダだと、その間にqueryされたRRsetは、署名が失効するまでキャッシュにつかまれてしまう。
  - RFC 4033のsection 7.1は、署名が失効するときをキャッシュの有効期間の終わりと見なしてもよい、と言っている。

4.1.1. Time Considerations

- 署名が失効したタイミングでキャッシュ上のデータが一斉に無効になって、ドッとqueryがやってくる。
- これを避けるに、どのRRのTTLも、署名の有効期間の何分の1かにしておくことを提案する。

4.1.1. Time Considerations

- 署名の開示期間は、署名の有効期限よりもゾーン中で最大のTTLだけ前に満了させることを提案する。
  - 署名の有効期限ぎりぎりで再署名すると、データがキャッシュ上で一斉に無効になって、authoritativeサーバの負荷が上がる。

4.1.1. Timing Considerations

- ゾーン中で最小のTTLでも、信頼の連鎖に関係するすべてのRRの取得と検証に必要な時間よりは　十分な余裕を持っておくことを提案する。
- 実験環境では、5分とか10分とかを下回る短いTTLで2つの問題により破綻をきたすことが観測された。

4.1.1. Timing Considerations

- 検証が完結するより前に、必要なデータがキャッシュ上で無効になってしまった。
  - validatorは、検証が完結するまで、必要なデータを保持しなければならない。
  - 信頼の連鎖を完成させるために必要なすべてのRRについて言える。
    - DS, DNSKEY, RRSIG, (検証の対象の)最終的な応答
    - つまり受けたqueryに対して応答するために必要なすべてのRR。
- 頻繁に検証すると、recursiveサーバの負荷が上がる。
  - delegation pointのデータ、つまりDS、DNSKEY、RRSIGはキャッシュの恩恵を受けるのでTTLは長く取るべし。

4.1.1. Time Considerations

- スレーブサーバは、サーブしているゾーンについて、RRSIGが失効する前に新しい署名の入ったゾーンデータを取得できなければならない。
  - マスタと同期がとれなくなって署名が失効したら、スレーブは何も応答しない方がいい。
  - 通常、マスタサーバと長い間通信できないスレーブは、そのデータをexpireさせる。
  - サーバは(実装によって、という意味?)そのゾーンについてさまざまな応答をする。
    - SERVFAILを返したり、AA bitを倒して応答したり。

4.1.1. Time Considerations

- いつexpireするかはSOAに書いてあって、マスタとスレーブが最後にちゃんとrefreshできたときを起点とした相対時間。
  - refresh: SOAのserialの次の数字。スレーブがマスタにゾーンデータが更新されたかどうかの手がかりとしてserialをqueryする間隔。
- RRSIGのexpire(ここでは「失効」をあてている)とSOAのexpireとの間には何ら関係はない。
  - expire: SOAの最後から2つ目の数字。スレーブが、どれだけの間、refresh動作に失敗し続けたらゾーンデータをexpire状態に遷移させるか。

4.1.1. Time Considerations

– サーバがDNSSECを有効にしたゾーンをサーブしていると、SOAのexpireを迎える前に署名が失効してしまうこともあり得る。
  • SOAのパラメータではどうしようもない。

– SOAのexpireを署名の有効期間と同じか短くしておけば、影響は最小で食い止められる。

4.1.1. Time Considerations

- authoritativeサーバがゾーンデータを更新できないと、そのゾーンが失効した署名を含んでいる間は、non-secureレゾルバはどこかのスレーブが提供しているデータで名前解決を継続できるのに、security-awareなレゾルバは、応答がbogus(署名の検証に失敗した)となってしまうので、問題に遭遇することになる。
- SOAのexpiration timerを署名の有効期間のおよそ1/3とか1/4とかにしておくことを提案する。
- そうすれば、署名が失効してしまう前にゾーン転送の異常に気づくことができる。

4.1.1. Time Considerations

- スレーブサーバの運用者は、スレーブとしてサービスしているゾーンの署名の失効を監視するwatch dogを導入して、適切なアクションをとることを提案する。
- (プライマリの管理者が)expireタイマの値を決めるときには
  - すべてのスレーブがexpireするのはどんな場合か?
  - スレーブの管理者に正しいデータを読み込んでもらうのに、どれだけ時間がかかるか?
- を考慮しなければいけない。
- DNSSEC固有のことではないが、署名の有効期限を決めるのに影響する。

4.2. Key Rollovers

- あるゾーンが、非常事態への備えとして定期的に鍵を更新しているか、非常事態が起きてしまったときだけに更新しているのかに関わらず、鍵のロールオーバ(更新)はDNSSECを使うにあたっての宿命である。
- 鍵を更新するときは、古いゾーンデータは、まだキャッシュの中にあることを忘れてはならない。
- クライアントが、そのゾーンの名前を解決できなくなるかもしれないので、DNSSECを導入するときには重要。

4.2. Key Rollovers

- 最も急を要する例は、古い方の鍵をキャッシュしていないレゾルバが、古い方の鍵で施した署名を検証しようとするときに起こる。
  - DNSKEY(旧を含む)はキャッシュされておらず、RRSIG(by旧DNSKEY)はキャッシュされているとき。
- もし、古い方の鍵が既にゾーンデータからなくなっていると、検証に失敗する。
- 古い方の鍵だけをキャッシュしているレゾルバが、それを使って、新しい鍵で施した署名を検証しようとしても、同様に失敗する。

4.2.1. Zone Signing Key Rollovers

- 鍵の新旧とキャッシュ上のデータがからむ問題を起こさずにZSKをロールオーバする方法は2つある。
  - double signatures(4.2.1.2)
  - pre-publication(4.2.1.1)

4.2.1.1. Pre-Publication Key Rollovers

- すべてのデータに2回署名しなくていい。
- 鍵が漏洩したときに利点がある。
- 古い方の鍵が漏洩した時点で、既に新しい方の鍵はDNSで配布されている。
  - キャッシュに撒かれている
- ゾーンの管理者は、即座に新しい鍵に切り替え、漏洩した方の鍵はゾーンから削除できる。
- もう1つの利点は、ゾーンデータの寸法が倍にはならないこと。

Appendix B. Zone Signing Key Rollover How-To

- キャッシュにデータが残っていないことを保証した上で、pre-published signature方式を適用する方法。

Appendix B. Zone Signing Key rollover How-To

- Step 0: The preparation
  - 鍵を2つ生成して、両方ともDNSKEY RRsetに含めて公開する。
  - 片方を”active”、もう片方を”published”と呼ぶ。
  - publishedの方の秘密鍵はオフラインで保管するのが望ましい。
  - (DNSSECの)プロトコルには、ある鍵(DNSKEY)がactiveなのかpublishedなのかの属性はない。
    - ので、ノートに書き留めるとか鍵管理ツールを使うとか。

Appendix B. Zone Signing Key Rollover How-To

- Step 1: Determine expiration
  - ロールオーバを開始する時点で、ゾーンデータに登場する、そのときのactiveの鍵で生成したRRSIGのうち、TTLが一番大きな物がキャッシュから消えるのはいつかを把握する。
  - その時間が過ぎるのを待つ。

Appendix B. Zone Signing key Rollover How-To

- Step 2
  - publishedの方の鍵でゾーンデータに署名する。
    - その鍵はpublishedからactiveになる。
  - activeだった鍵を使うのをやめる。
    - その鍵で施した署名を消す。
    - その鍵はactiveからrolledになる。

Appendix B. Zone Signing Key Rollover How-To

- Step 3
  - 署名の有効期間を1回経過すれば、次のロールオーバ(Step 1)を始めても安全である。
  - It is safe to engine in a new rollover (Step 1) after at least one signature validity period.
  - signature validity periodではなく、~~DNSKEY~~ RRのTTLでは?

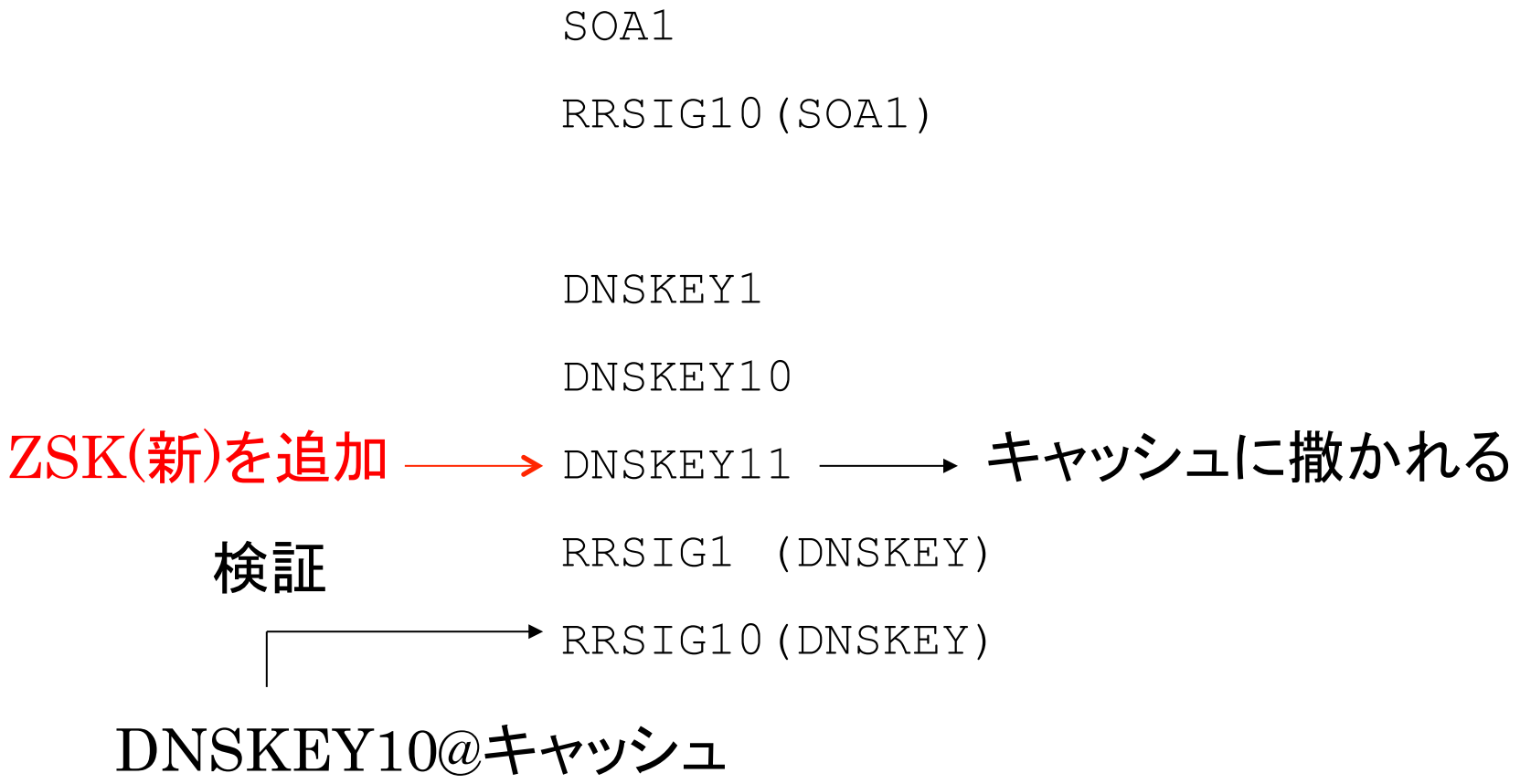
validity periodは変。
DNSKEYではなくRRSIGでは?(6/24)

4.2.1.1. Pre-Published Key Rollover

- initial

4.2.1.1. Pre-Published Key Rollover

- new DNSKEY



4.2.1.1. Pre-Publish key Rollover

- DNSKEY 11をkey setに追加。
- まだこの鍵では署名しない。
- でも、ゾーンデータに載せちゃった以上、公開鍵に対するbrute force attackには安全ではない。
- 最短でも、セカンダリに伝播するための所要時間+key setのTTLの間は、この状態でなければならない。

4.2.1.1. Pre-Published Key Rollover

- new RRSIG

4.2.1.1. Pre-Publish Key Rollover

- DNSKEY 11だけで署名する
  - DNSKEY 10で施した署名は削除する。
- DNSKEY 10はkey setに残す。
- 既にキャッシュ上に載っているver.1(==SOA1)のゾーンデータを、ver.2(==SOA2)から取得したkey setで検証できるようにするため。
- ver.1のデータがキャッシュ上から消えるまでは、この状態でなければならない。
  - セカンダリに伝播する所要時間+ver.1のゾーンデータの最大のTTL

4.2.1.1. Pre-Published Key Rollover

- DNSKEY removal

```
SOA3

RRSIG11(SOA3)


DNSKEY1


DNSKEY11

RRSIG1 (DNSKEY)

RRSIG11(DNSKEY)
```

DNSKEY10は、もはやキャッシュ上にはない。

4.2.1.1. Pre-Publish key Rollover

- ロールオーバ直後に、いつも未来の鍵を公開し始めるように単純化もできる。

SOA0	SOA1	SOA2
RRSIG10(SOA0)	RRSIG10(SOA1)	RRSIG10(SOA2)
DNSKEY1	DNSKEY1	DNSKEY1
DNSKEY10	DNSKEY10	
DNSKEY11	DNSKEY11	DNSKEY11
		DNSKEY12
RRSIG1 (DNSKEY)	RRSIG1 (DNSKEY)	RRSIG1 (DNSKEY)
RRSIG10(DNSKEY)	RRSIG11(DNSKEY)	RRSIG11(DNSKEY)

4.2.1.1. Pre-Publish key Rollover

```
SOA3                    SOA4
RRSIG10(SOA3)           RRSIG10(SOA4)

DNSKEY1                 DNSKEY1

DNSKEY11
DNSKEY12                DNSKEY12
                        DNSKEY13
RRSIG1 (DNSKEY)         RRSIG1 (DNSKEY)
RRSIG12(DNSKEY)         RRSIG12(DNSKEY)
```

4.2.1.1. Pre-Publish Key Rollover

- new DNSKEYフェーズで追加された鍵はまだ署名には使われない。
  - 物理的にセキュアな方法で保存してよい。
  - ゾーンデータを署名するには必要ない。

4.2.1.2. Double Signature Zone Signing Key Rollover

- new DNSKEY stage
  - 新しいゾーンデータがすべてのauthoritativeサーバに行き渡らなければならない。
  - validatorのキャッシュに載っているデータは消えなければならない。
    - 最低でもゾーン中の最大のTTLだけは待たなければならない。

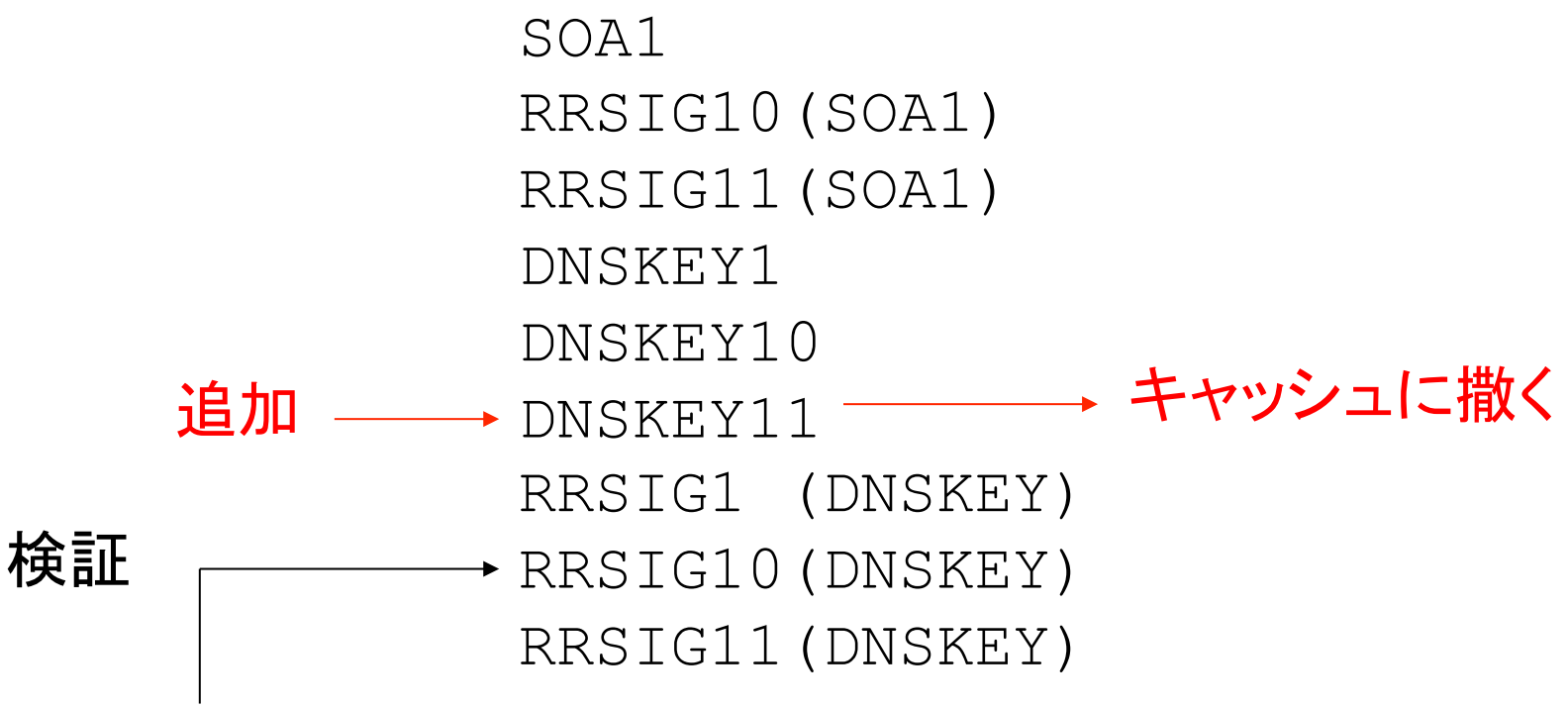
4.2.1.2. Double Signature Zone Signing Key Rollover

- initial

```
SOA0
RRSIG10(SOA0)

DNSKEY1
DNSKEY10

RRSIG1 (DNSKEY)
RRSIG10(DNSKEY)
```

4.2.1.2. Double Signature Zone Signing Key Rollover

- new DNSKEY



DNSKEY10@キャッシュ

4.2.1.2. Double Signature Zone Signing Key Rollover

- validatorのキャッシュから、ver.0(initial)のデータが全部消えるまで続けなければならない。
  - DNSKEY11を含んでいないDNSKEY RRSet
  - RRSIG10(*)
- ver.0に登場する一番大きなTTL

4.2.1.2. Double Signature Zone Signing Key Rollover

- DNSKEY removal

```
SOA2

RRSIG11(SOA2)
DNSKEY1

DNSKEY11
RRSIG1 (DNSKEY)

RRSIG11(DNSKEY)
```

DNSKEYをキャッシュしていれば
必ずDNSKEY11も載っている

4.2.1.2. Double Signature Zone Signing Key Rollover

- どの瞬間でも、前のバージョンに含まれていた(キャッシュから索きあてた)RRSIGは今のバージョンのゾーン(キャッシュにヒットせずrecursive queryで取ってきた)のDNSKEY RRSetで検証できなければならない。
  - 逆も
- new DNSKEYフェーズの持続期間も、次のロールオーバまでの間隔も、最低でもゾーン中の最大のTTLは必要。

4.2.1.2. Double Signature Zone Signing Key Rollover

- new DNSKEYフェーズはver.0に含まれている署名が失効する前に終わらせることを勧める。
- これで古い署名はすべてのキャッシュから消える。
  - This way all caches are cleared of the old signatures.
  - ???
- が、ゾーン中の最大のTTLよりは長くなければならない。
- この例ではロールオーバ中にゾーンデータは変更されないことを仮定している。
  - ちゃんと両方の鍵で署名しておけば、新しいデータを追加しても大丈夫。

4.2.1.3. Pros and Cons of the Schemes

- Pre-Publish Key rollover
  - 2回署名しなくていい。
  - ロールオーバに先立って、次に使う鍵を公開しなければならない。
    - 暗号的手法による攻撃に利用され得る。
  - 4段階の手順を踏まなければならない。
  - (このセクションはZSKのロールオーバを論じているが)KSKのロールオーバ(4.2.3)にこの手法を使うと、親ゾーン側での作業が多くなる。

4.2.1.3. Pros and Cons of the Scheme

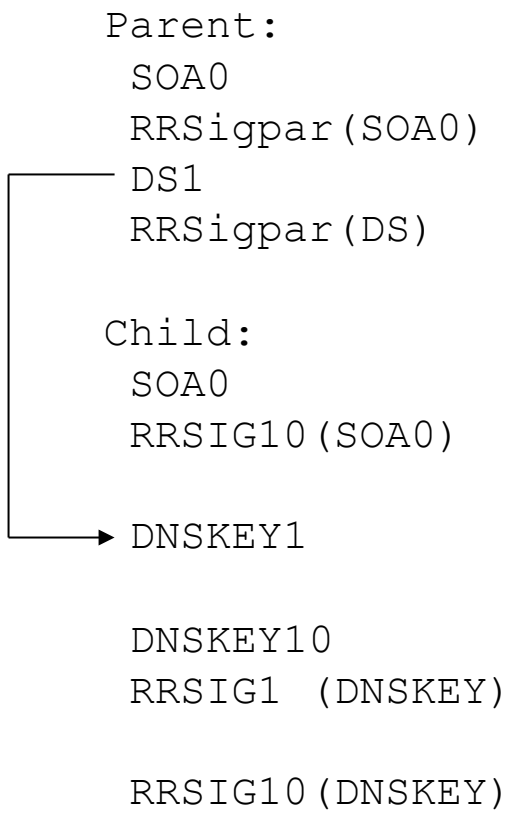
- Double signature ZSK rollover
  - 難点はゾーンに含まれる署名の量が倍であること。
  - ゾーンデータがとても大きいと、高くつく。
    - prohibitive: 禁止の、はなはだ高価な
  - 利点は手順が3段階で済むこと。

4.2.2. Key Signing Key Rollovers

- KSKのロールオーバには、ZSKのロールオーバと同じことが言える。
- キャッシュに載っている古いデータ(zone apexのkey setだけだが)を新しいkey setで検証できることと、その逆ができることを保証するために、double signature schemeを使う。
- KSKで署名されるのはkey setだけなので、ゾーンの寸法については、気にしなくていい。

4.2.2. Key Signing Key Rollovers

- initial

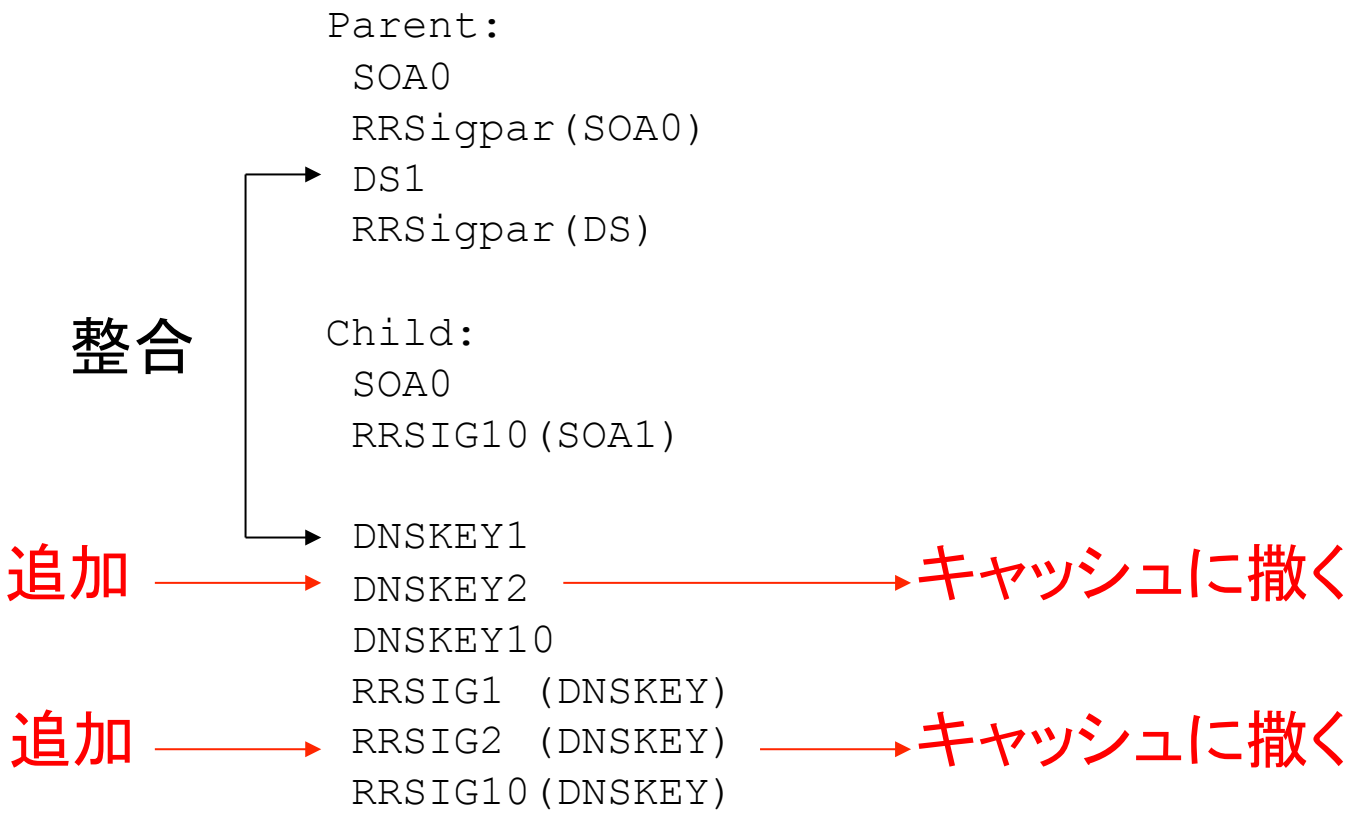


TTL_DS ::= TTL(DS1)

DSは親のauthoritativeデータなので、TTL_DSは子には制御できない。

4.2.2. Key Signing Key Rollovers

- new DNSKEY

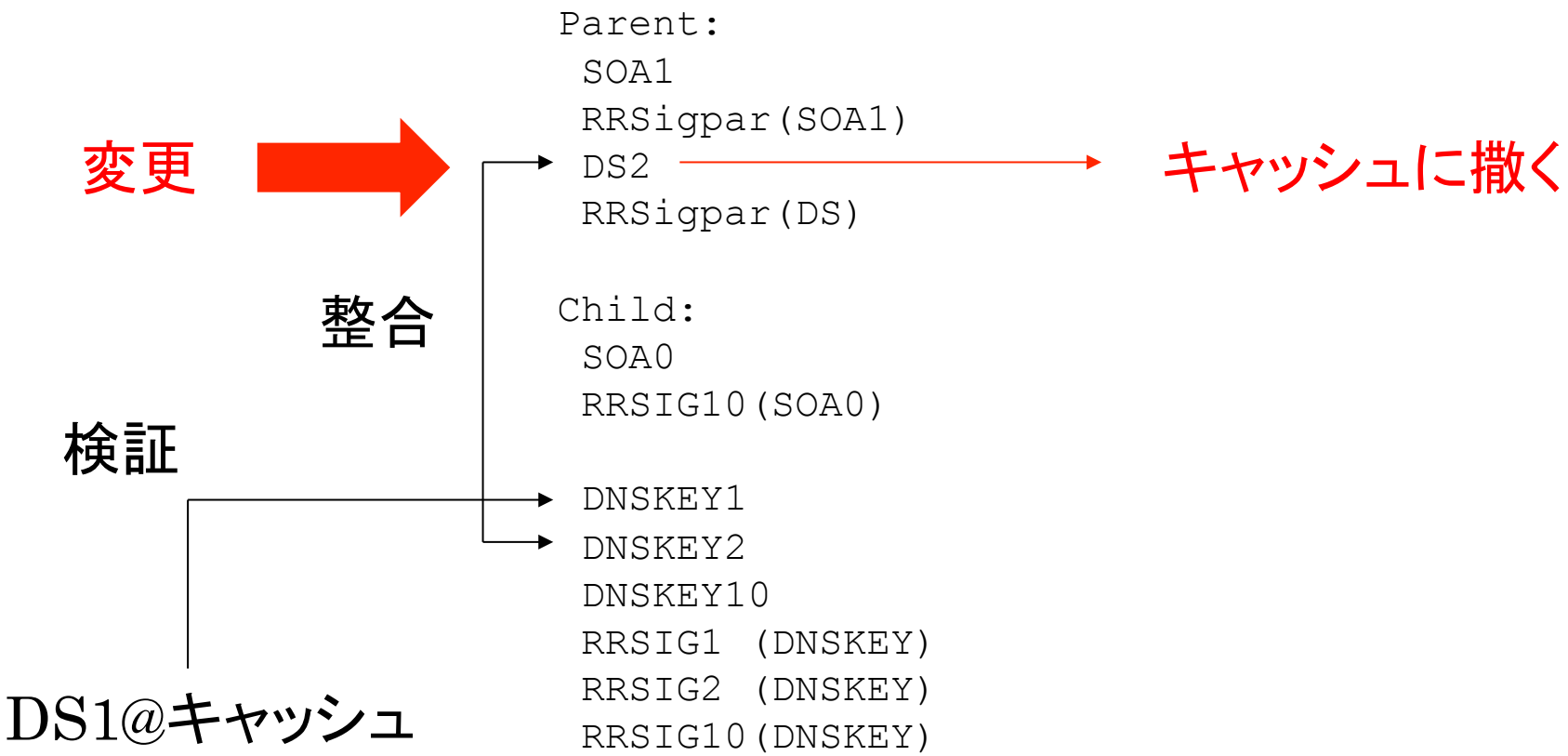


4.2.2. Key Signing Key Rollovers

- 2番目のKSKであるDNSKEY2を生成する。
- この鍵を親に送る。
- DNSKEY2を指すDSを載せてくれるのを待つ。
- 新しいDSが親ゾーンに載ったら、 親ゾーンの全部のauthoritativeサーバに行き渡るのを待つ。
- さらに、古いデータがvalidatorのキャッシュから消えるように、TTL_DSの間だけ待つ。

4.2.2. Key Signing Key Rollovers

- DS change



4.2.2. Key Signing Key Rollovers

- DNSKEY removal

```
Parent:
 SOA1
 RRSigpar(SOA1)
 DS2
 RRSigpar(DS)

Child:
 SOA2
 RRSIG10(SOA2)


 DNSKEY2
 DNSKEY10

 RRSIG2 (DNSKEY)
 RRSIG10(DNSKEY)
```

4.2.2. Key Signing Key Rollovers

- 子に至る信頼の連鎖を維持する責任が…。
  - The scenario above puts the responsibility for maintaining a valid chain of trust with the child.
- 親には子ゾーン1つにつき(アルゴリズム毎に)1つのDSだけが載っていることが前提。
- 別解
  - 既存の信頼の連鎖を使うと、親子間のやり取りはin-bandでできる。
    - 新しいKSKを親がDNS経由で取得する、ということか?
  - 古いKSKの削除は親から合図する。
  - この方法だと、親にDSが2つ載るタイミングがある。
  - 執筆時点では、このやり取りをするプロトコルはない。

4.2.3. Difference Between ZSK and KSK Rollovers

- KSKのロールオーバ
  - 親とのやり取りが発生
  - trust anchorの変更もあり得る
  - それが完了するのを待たなければならない。
- ZSKのロールオーバ
  - pre-publish(4.2.1.1)
  - double signature(4.2.1.2)
- KSKはkey setの検証に使われる。
- ZSKのロールオーバ中にKSKは変更されない。
  - キャッシュは新しいkey setの検証に使える。

4.2.3. Difference Between ZSK and KSK Rollovers

- KSKのロールオーバにもpre-publish方式を使うことはできる。
- pre-publishするレコードは親のDS。
- KSKに適用するには、pre-publish方式は不利な点がある。

4.2.3. Difference Between ZSK and KSK Rollovers

initial	new DS	new DNSKEY	DS/DNSKEY removal
Parent:			
SOA0	SOA1	-------->	SOA2
RRSIGpar(SOA0)	RRSIGpar(SOA1)	-------->	RRSIGpar(SOA2)
DS1	DS1	-------->	DS2
	DS2	-------->	
RRSIGpar(DS)	RRSIGpar(DS)	-------->	RRSIGpar(DS)
Child:			
SOA0	-------->	SOA1	SOA1
RRSIG10(SOA0)	-------->	RRSIG10(SOA1)	RRSIG10(SOA1)
	-------->		
DNSKEY1	-------->	DNSKEY2	DNSKEY2
	-------->		
DNSKEY10	-------->	DNSKEY10	DNSKEY10
RRSIG1 (DNSKEY)	-------->	RRSIG2(DNSKEY)	RRSIG2 (DNSKEY)
RRSIG10(DNSKEY)	-------->	RRSIG10(DNSKEY)	RRSIG10(DNSKEY)

4.2.3. Difference Between ZSK and KSK Rollovers

- 子ゾーンがKSKをロールオーバしようとするときは、新しい鍵(か対応するDS)を親に送ってnew DSフェーズに突入する。
- 親はDS1とDS2を公開する。
- 新しいDSがDNSに行き渡ったら(セカンダリ/キャッシュ)、子はDNSKEY1をDNSKEY2に交換してよい。
- 親に古いDSを削除していいと連絡する。

4.2.3. Difference Between ZSK and KSK Rollovers

- 不利な点
  - new DSのフェーズでは、DNSKEY2はまだ公開していないのでDS2とDNSKEY2との整合性はDNSでは確認できない。
  - 4.4.3で論じるsecurity lameを引き起こす鍵を載せることになってしまう。
  - double signature方式では、親とのやり取りは1回で済むのに、pre-publish方式だと2回発生してしまう。

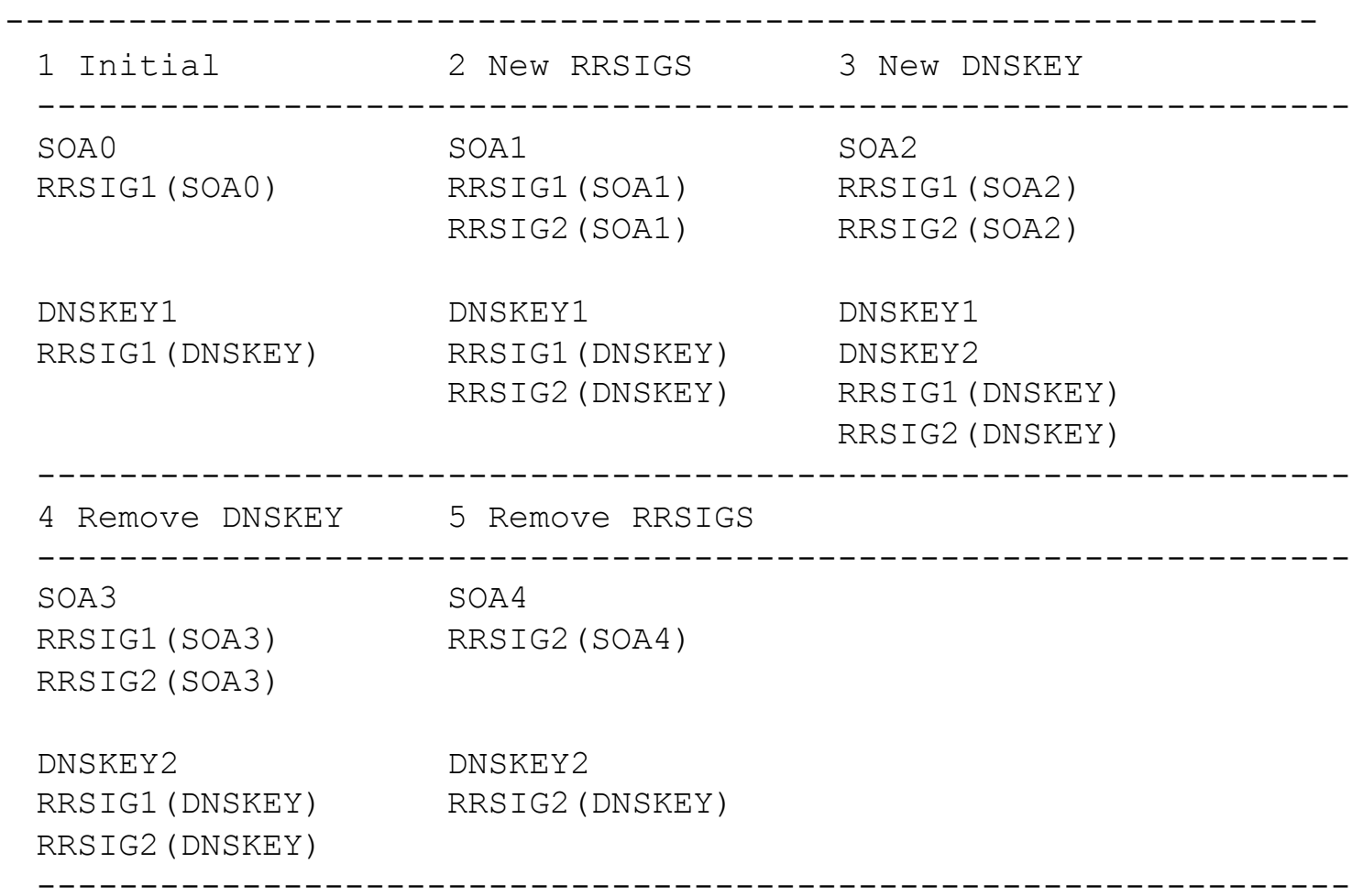
4.2.4. Key algorithm rollover [-bisで追加]

- 特別なロールオーバとして、鍵アルゴリズムのロールオーバがある。
  - 新しいアルゴリズムの追加
  - 古いアルゴリズムの削除
  - 両方
- ロールオーバ中の完全性を維持するため、追加の手順が必要になる。

4.2.4. Key algorithm rollover

- RFC 4035のセクション2.2で言及されているalgorithm downgrade protectionのために、署名に使っていないアルゴリズムの鍵は持たないべき。
  - There MUST be an RRSIG for each RRset using at least one DNSKEY of each algorithm in the zone apex DNSKEY RRset.
- アルゴリズムを追加するときは、まず署名を追加する。
- TTLが経過し、キャッシュ上に載っている既存アルゴリズムだけによる署名が付いたデータが消えたら、新しいアルゴリズムのDNSKEYを載せてよい。
- 古いアルゴリズムをやめるときは、まずDNSKEYを削除する。

4.2.4. Key algorithm rollover



4.2.4. Key algorithm rollover

- step 2
  - 新しいアルゴリズムで施した署名を追加する。
  - でも鍵自体はまだ載せない。
  - key setに対する署名はkey set自体と同期していなければならない。
  - RRSIGだけを単独で要求されることもあり得るので、DNSKEYにも新しいアルゴリズムによる署名が必要。
- キャッシュ上からデータが消えたら、新しい鍵を追加してよい(step 3)。

4.2.4. Key algorithm rollover

- 次の手順は古いアルゴリズムを削除すること。
- 署名に先立って鍵を削除しなくてはならない。
- step 4として鍵を消す。
- キャッシュからデータ(旧アルゴリズムのDNSKEY)が消える。
- step 5として署名を削除する。
- これで作業中も完全性が損なわれない。

4.2.5. Automated Key Rollovers

- 鍵は定期的に変更しなければならない。
  - 自動化したい。
- ZSKのロールオーバには、子ゾーンだけが関係するので、自動化は簡単。
- KSKのロールオーバには、親と子とのやり取りが発生する。
- 親に新しい鍵を送らなければならない。
- ロールオーバ中にアタックを受けないように、親に鍵を送るときは認証と完全性の確保が必要。

4.3. Planning for Emergency Key Rollover

- 鍵が漏洩しちゃったときへの備えについて論じる。
- 鍵が漏洩しちゃった(ことが判明した | かもしれない)ときにとるべき手順をドキュメントにしておくのがお勧め。
- ある鍵(の組)の秘密鍵が漏洩してしまうと、有効な信頼の連鎖が存在する限り、その秘密鍵を使われてしまうかもしれない。
  - 不正なゾーンデータに、その秘密鍵で署名を施すと、DNSSEC的には検証に成功してしまう。

  →不正なデータであることを検知できない。

4.3. Planning for Emergency Key Rollover

- 信頼の連鎖は
  - 信頼の連鎖の途中に登場する、漏洩した鍵で施された署名が有効か
  - 親ゾーンに設定されたDSが漏洩した鍵を指しているか
  - 問題の鍵がresolver(validator)にtrust anchorとして設定されていて、検証の初期情報として使われるか
    - 一般には、これを更新するのは大変

である限り、機能(intact)してしまう。

4.3. Planning for Emergency Key Rollover

- Zone operatorは、validatorのキャッシュに載ってしまったデータの検証に失敗するという犠牲を払ってでも、漏洩してしまった鍵の不正使用を食い止めるべきか否かを決断しなければならない。
- 漏洩してしまった鍵へ至る信頼の連鎖を断ち切れば、既にキャッシュ上に載っている正当なデータの検証に失敗する。
- 正規の鍵交換の手順を踏むと、漏洩した鍵が有効である間は、不正に使われ得る。

4.3.1. KSK Compromise

- 漏洩したKSKをDNSKEYとして含んでいるゾーンは、その鍵がtrust anchorに設定されていたり、親ゾーンにその鍵を指すDSが載っている限り、脆弱である。
- 漏洩したKSKで、攻撃者の手中にあるゾーンのkey setに署名されてしまうかもしれない。
  - そのkey setに含まれるZSKで残りのゾーンデータに署名すれば、そのゾーンデータは検証に通る。
- そのゾーンはDNSを汚染するのに使われるだろう。

4.3.1. KSK Compromise

- というわけで、KSKが漏洩したときには、できるだけ早くtrust anchorとか親ゾーンのDSとかを交換しなければならない。
- 緊急で鍵を交換するときに、信頼の連鎖を維持しながらやるか、犠牲にするかはローカルポリシー。
- 親ゾーンに載っているDSが問題のKSKを指している物しかないときに、そのDSがある内にKSKを削除してしまうと、信頼の連鎖は切れる。

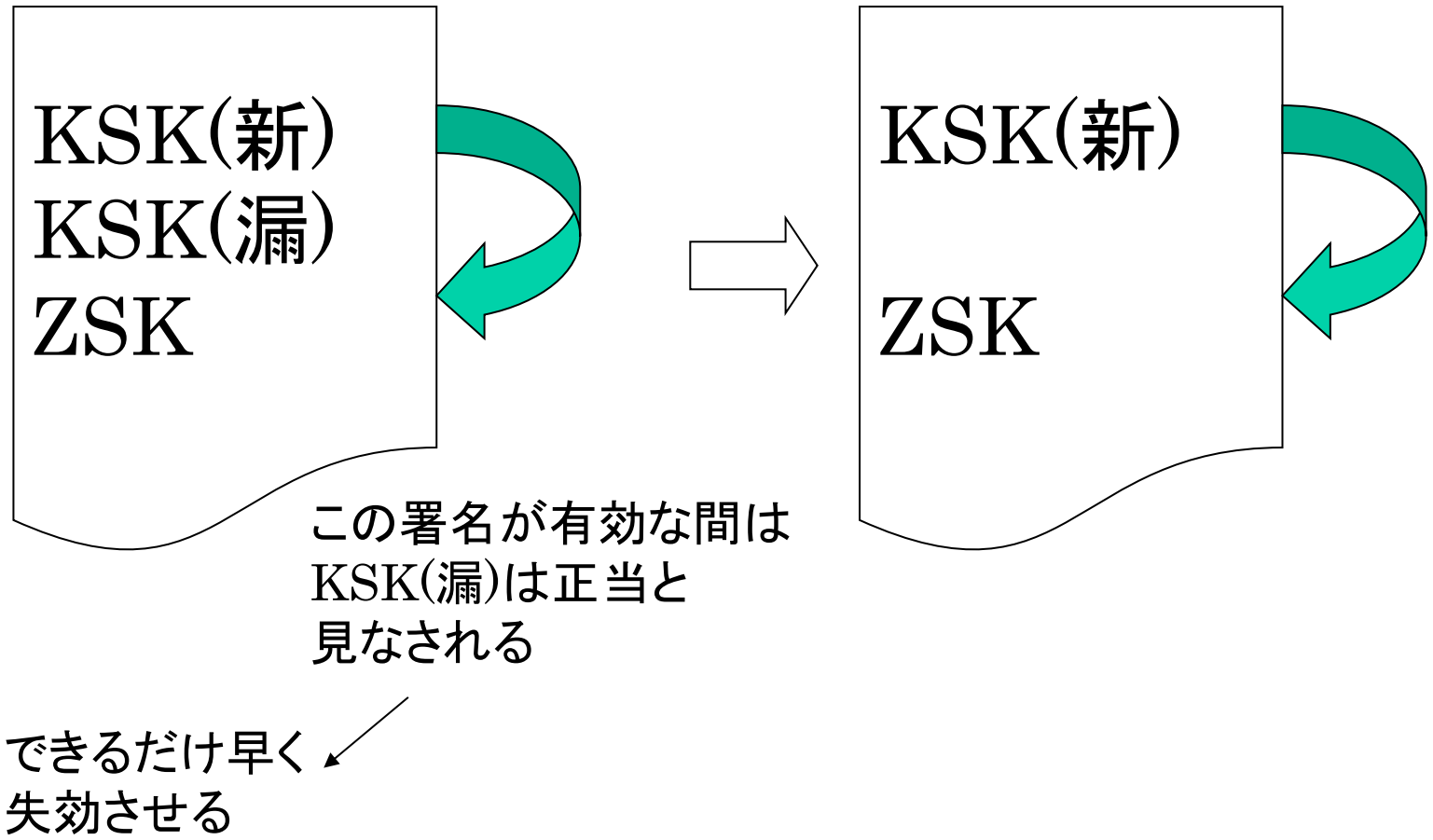
4.3.1. KSK Compromise

- 親のゾーンに漏洩したKSKを指すDSが載っている間は、攻撃者のゾーンがそのKSKを使っていると、その(攻撃者が用意した)ゾーンは検証に通ってしまうことに注意。
- 慌てて漏洩したKSKのDNSKEY RRを削除しても、親ゾーンにDSが残っている間は、本当のゾーンは検証に失敗するわ、攻撃者のゾーンは検証に通ってしまうわで、ロクなことはない。
- 親ゾーンに新しいDSが載るまで、漏洩したKSKのDNSKEY RRは消さない方がいい。

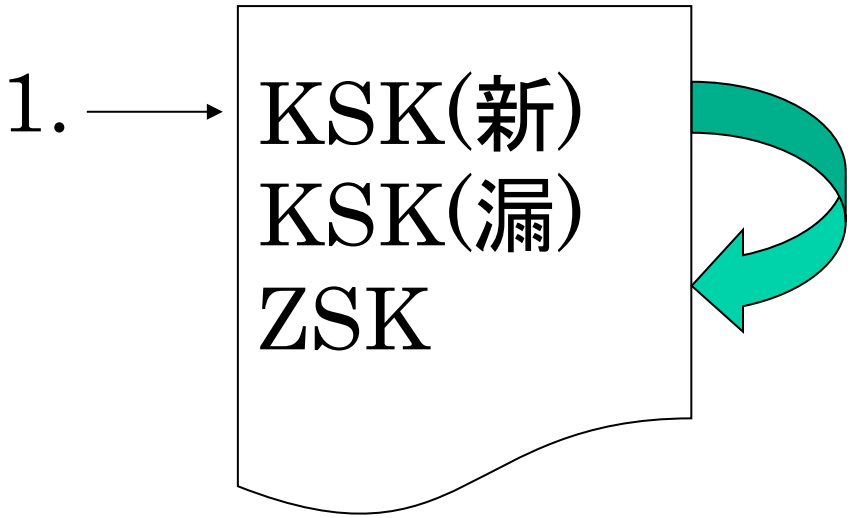
4.3.1.1. Keeping the Chain of Trust Intact

- このアドバイスに従うなら、KSKを交換するタイミングはちょっとクリティカル。
- ゴール
  - 親ゾーンに新しいDSが載り次第、漏洩したKSKを削除すること。
  - 親ゾーンに新しいDSが載り次第、新しいKSKを使って漏洩したKSKを含むkey setに施した自己署名を失効させる。

4.3.1.1. Keeping the Chain of Trust Intact



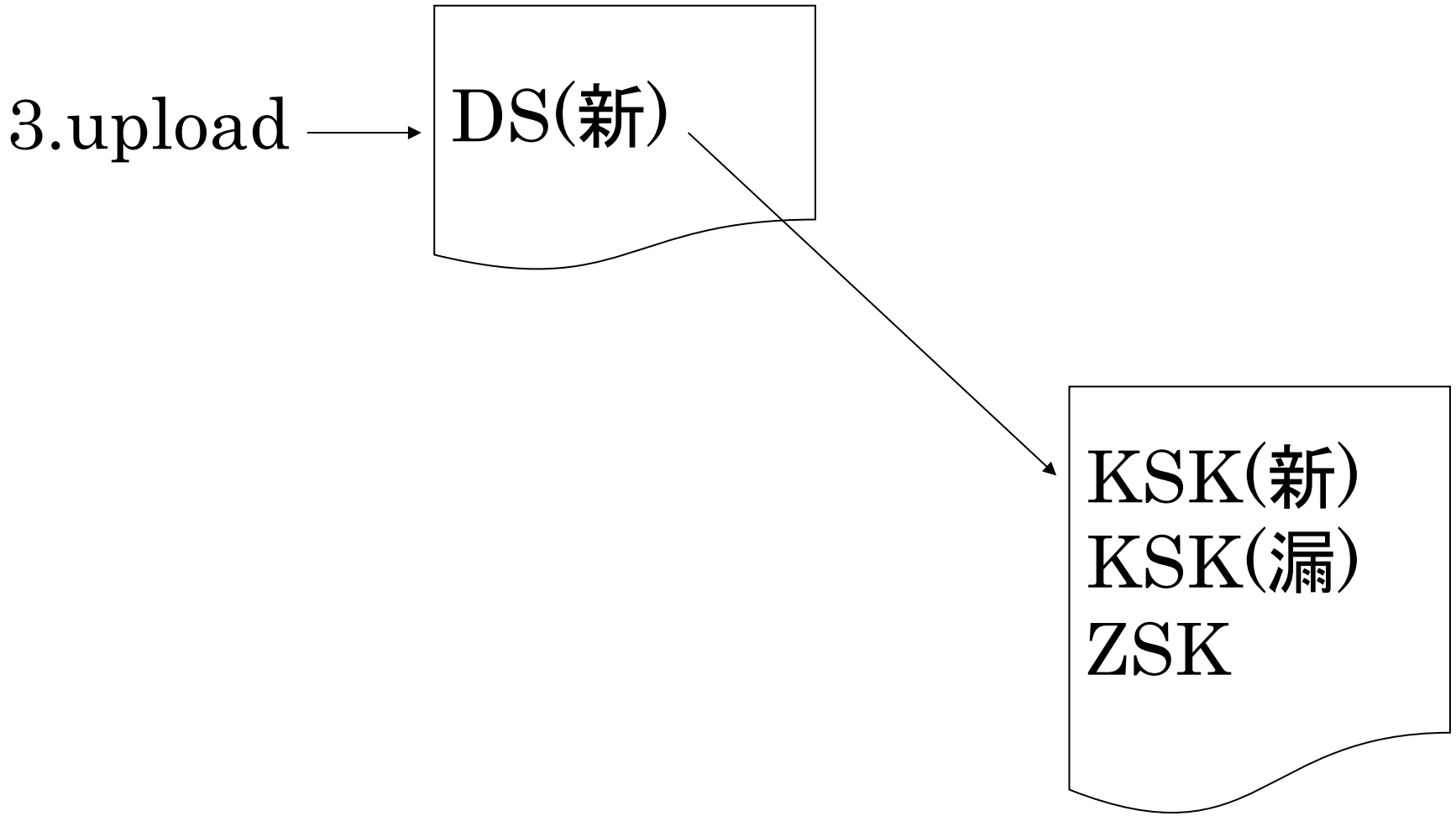
4.3.1.1. Keeping the Chain of Trust Intact



2.署名する

親ゾーンにDS(新)が載って、世の中のキャッシュからDS(漏)が消えたら、すぐ無効になるように。

4.3.1.1. Keeping the Chain of Trust Intact

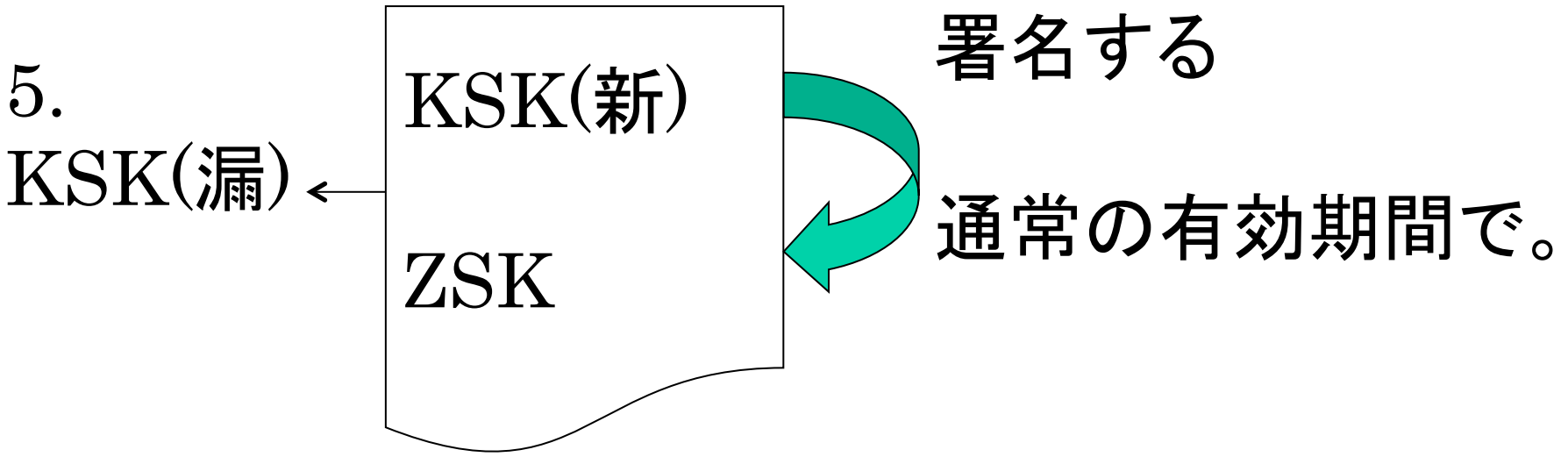


4.3.1.1. Keeping the Chain of Trust Intact

4.通常のKSKのロールオーバーの手順を踏む。

- 3.でアップロードしたDS(新)がauthoritativeサーバ群(secondaryにも)行き渡るのを待つ。
- DS(漏)のTTLが過ぎるのを待つ。
  - これで、DS(漏)をキャッシュに握っていたvalidatorもDS(新)を取りに行く。
- 2.の工程で見積もったより所要時間がかかっているようなら、再署名して有効期間を延ばす。

4.3.1.1. Keeping the Chain of Trust Intact



4.3.1.1. Keeping the Chain of Trust Intact

•An additional danger of a key compromise is that compromised key could be used to facilitate a legitimate DNSKEY/DS rollover and/or name server change at the parent.

•漏れちゃった鍵は、正当なDNSKEYやDSをロールオーバーしたり(本当のゾーンデータを検証に失敗させる?)、親ゾーンでネームサーバ(委任先のこと?)を変更するのに(ゾーン乗っ取り?)、都合よく使えてしまったりもするかもしれない。

4.3.1.1. Keeping the Chain of Trust Intact

- そのような場合、問題のドメインは泥仕合(dispute)になる。
- contact personにout-of-band(DNSに依存しない)でセキュアに通報できる機構が必要。
- これは、DNSKEYやDSが親ゾーンでの認証に使われているときにだけ問題になり得る。

4.3.1.2. Breaking the Chain of Trust

- Bogusに(検証に失敗)させる方法
  - まずKSKを交換してしまう。
  - おもむろに、親にDS(新)を送る。
  - 親がDS(新)をサーブし始めないと、信頼の連鎖は復旧しない。
- Insecureに(検証できなく=DNSSECで正当性を担保できなく)する方法
  - 親ゾーンからDSを抜く。

4.3.2. ZSK Compromise

- 親(一般には別組織)とのやり取りが発生しない分、ZSKの漏洩はKSKほど深刻ではない。
- やっぱり、できるだけ早く新しいZSKで再署名しなければならないことに変わりはない。
- 親子間のやり取りは発生せずローカルの操作だけなので、さっさとできる。
- 漏洩した鍵を慌てて即座に抜いてしまうと、通常のロールオーバーと同様に検証に問題を生じ得ることは考慮しなければいけない。

4.3.2. ZSK Compromise

- 漏洩したZSKで施した署名のRRSIGがexpireするまでは、そのゾーンは依然リスクにさらされている。

そう
(6/24)

  - expireってvalidatorのキャッシュから消えること?
  - どういうケースでどういう風に危険?

cache poisoningやman-in-the-middle attackとの
合わせ技がアブナイ(6/24)

4.3.3. Compromises of Keys Anchored in Resolvers

- 鍵はレゾルバにも設定され得る。
- DNSSECが順調に普及したら、rootの鍵は大抵のsecurity aware resolverに設定されるだろう。
- trust-anchor keyが漏洩したら、その鍵を使っているレゾルバに通知しなければならない。
- ゾーンの管理者は、SEP keyのロールオーバーを通知するメーリングリストを作ることを検討すると良い。
- その通知は、当然ディジタル署名か何かで認証しなければいけない。

.seはメーリングリストを作っているらしい。スウェーデンには、メジャーなISPが4つしかないので、それでうまくいっている。(6/24)

4.3.3. Compromises of Keys Anchored in Resolvers

- trust anchorに設定している鍵を更新することになったエンドユーザは、必ず新しい鍵が正当な物か検証(DNSSEC的な意味ではなくsocial engineering的な意味で)しなければならない。
- 新しい鍵は、例えばSSL/TLS経由のWebページなど、out-of-bandの手段で認証(authenticated)されなければならない。

4.4. Parental Policies

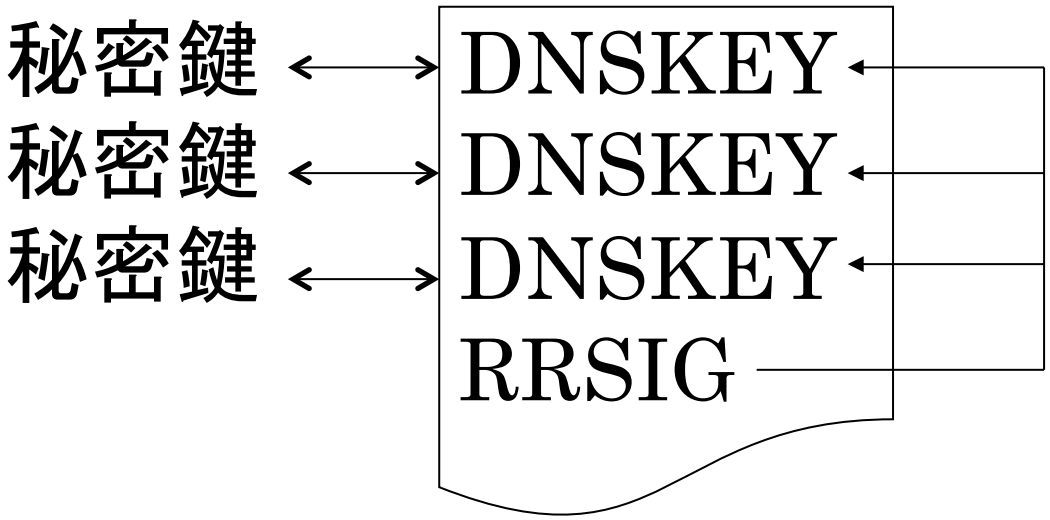
4.4.1. Initial Key Exchanges and Parental Policiies Considerations

- 最初の鍵交換(exchange:だけど子→親だけでは?)は、親の方針に依存する。
- 鍵交換の方法を決めるときには、鍵交換に適用される認証とauthorizationの強度は親子間の委任情報の交換に適用される認証とauthorizationの強度と同等であるべき。
- つまり、DNSSECだからといって、これまでDNSに関する情報のやり取りに使っていた以上に強固な認証は要らない、ということ。

4.4.1. Initial Key Exchanges and Parental Policies Considerations

- DNSKEYの受け渡しに、out-of-bandの確認を併用した上でDNS自体を使うと、ユーザのミスを削減できる。
- SEP bit(DNSKEYのflagが257)を使えば、DNSKEY RRsetの中から間違った鍵を取ってきてしまう可能性を減らせる。
- DNSKEY RRsetへの自己署名が、どのDNSKEY RRで検証できるかを調べると、どれが署名に使われている秘密鍵と対応するDNSKEYかがわかる。

4.4.1. Initial Key Exchanges and Parental Policies Considerations



どの公開鍵で検証できるか?

検証に成功したDNSKEYは、そのときに使われているKSK。

4.4.1. Initial Key Exchanges and Parental Policies Considerations

- DNS経由でDNSKEYを受け取れば、親がそのDNSKEYに対応するDSを公開すると、信頼の連鎖が機能することが保証できる。
- 親はDNSKEY RRが偽造されていないことを確認するために、DNS以外の手段での確認は、依然必要。

4.4.2. Storing Keys or Hashes?

- レジストリシステムを設計するときには、DNSKEYと対応するDSのどれを登録させることにするかを決めなければならない。
- 子ゾーンが、レジストリがまだ対応していないメッセージダイジェストアルゴリズムを使ったDSを使おうとする可能性があるので、レジストリは、子にDNSKEYを登録してもらい、そこからレジストリ側でDSを生成できることを仮定してはいけない。
- だから、少なくともDSは登録できるようにするのがお勧め。

4.4.2. Storing Keys or Hashes?

- トラブルシュートの役に立つかもしれないし、子が選んだメッセージダイジェスト(アルゴリズム)をレジストリ側がサポートしていれば、DNSKEYからDSを生成するオーバーヘッドは大したことないので、DNSKEYも登録することは意味がある。
- whoisみたいなレジストリディレクトリかなにか、DNS以外のアクセス手段で、DSがどの鍵から生成されているのかを確認できる仕組みもトラブルシュートの役に立つ。

4.4.2. Storing Keys or Hashes?

DNSKEYかDSか、という意味(6/24)

- ストレージはカスタマインターフェースやデータの受け渡し手段とも関係する。
- レジストリが対応していないアルゴリズムのDSも登録できるようにするのか、DNSKEYだけを登録できるようにするのか。
- Extensible Provisioning Protocol(EPP)へのDNSSEC拡張[RFC 4310]を使ってもいい。

4.4.3. Security Lameness

- 親に、子に存在しないDNSKEYを指すDSが登録されていること。
- 子はBogus(署名の検証に失敗した)と見なされるかもしれない。
  - lameではないDSも載っていたらO.K.なのでは?
- 鍵を登録するときの確認の一環として、子が登録を要求してきた鍵が本当にDNSに載っていることを確認することも考えられる。
- 子が、レジストリが対応していないハッシュアルゴリズムを使っているときには、key idの比較しかできない。

4.4.3. Security Lameness

- 子はDSが載っているSEP keyを抜くときは、慎重になるべし。
- 一旦security lameになると、修復にはDNSでの伝播時間を要する。

4.4.4. DS Signature Validity Period

- DSは署名の有効期限まではreplay攻撃に使われ得るので、署名の有効期間を短くしておけば、KSKが漏洩してしまったときに危険な時間を短くできる。
- 署名の有効期間が極端に短いと、署名に失敗したときに、そのゾーンがBogusになってしまうかもしれない。
  - 直す前に失効してしまう。
- 週末をはさむことを考慮すれば、DSの有効期間は最低2日必要。
  - 最低でも数日は確保するのがお勧め。

4.4.4. DS Signature Validity Period

- 長い方に関しては、鍵が漏洩したときに、どれぐらいまで危険を許容できるかに依存する。
- でも、DSの有効期間を短くすると、親側での運用上のリスクが増大する。
  - しょっちゅうやらせると、オペミスされる可能性が上がる、ということ?
- 親には、子が望むであろうより長い期間に対応できるポリシーが必要。

4.4.4. DS Signature Validity Period

- 親の運用上の要請と子が危険にさらされる期間との折り合いをつけると、DSの有効期間は1週間から数ヶ月といったところ。
- DSのTTLも、zone ownerがゾーンデータに署名しなければいけない回数の下限と、鍵が漏洩したときに危険にさらされる時間の上限にも関係する。
- TTLを短くすると、authoritativeサーバがqueryを受ける頻度は上がる。
- でも、DSを更新したときの伝播は早くなる。

原文の論旨はよくわかりませんね(6/24)

4.4.5. Changing DNS Operators

4.4.5.1. Cooperating DNS operators

- 親と子との関係は、しばしば(thin) レジストリモデルという言葉で表現される。
- レジストリは親ゾーンを管理し、registrant(子ドメイン名のユーザ)は、レジストラという仲介者を経由してレジストリとやり取りする。
- registrantがレジストリや他の委託先にDNSSECの鍵の管理を含めたDNSシステムの維持をアウトソースすることはよくある。ここでは、そのアウトソース先をDNSオペレータと呼ぶ。

4.4.5.1. Cooperating DNS operators

- ゾーンデータと鍵を管理しているDNSオペレータは、registrantがタイムリーに別のDNSオペレータに移るときの支障になることがある。
- registrantがloosing operator(出て行かれる方のDNSオペレータ)からgaining operator(移る先のDNSオペレータ)へ移るときを考える。
- まず協力的な関係にあるときを論じる。
- loosing operatorはgaining operatorに秘密鍵を引き渡さないことを前提にする。
  - ありがちな状況。

4.4.5.1. Cooperating DNS operators

pre-publish方式のZSKロールオーバー

- loosing operatorはgaining operatorのZSKをpre-publishする。

と

double signing方式のKSKロールオーバー

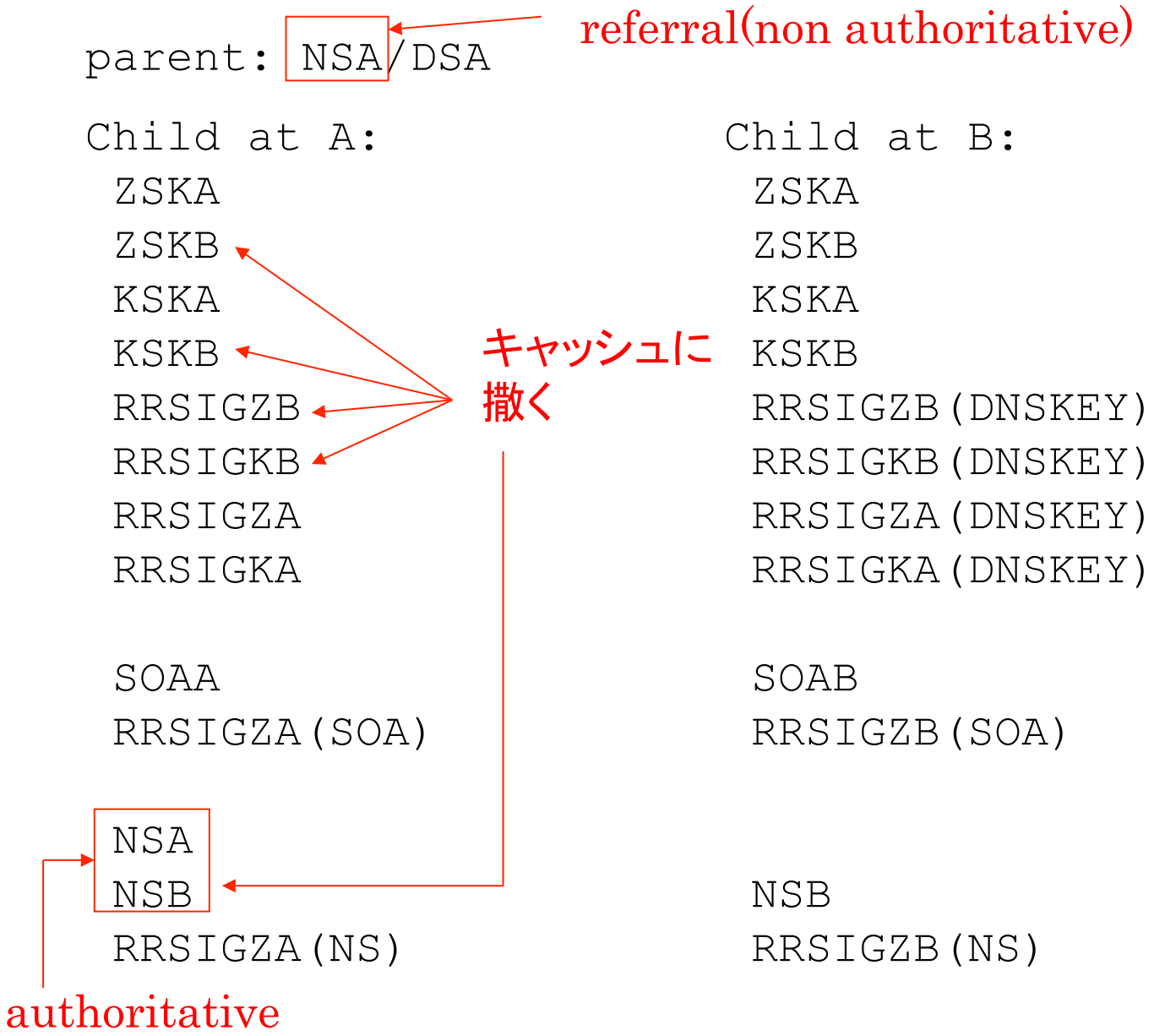
- 両者がKSKの公開鍵を交換し、自分のと相手のと両方のKSKを含んだkey setにそれぞれ自分の秘密鍵で署名し、自分のところのゾーンデータに含めて公開する。

の合わせ技。

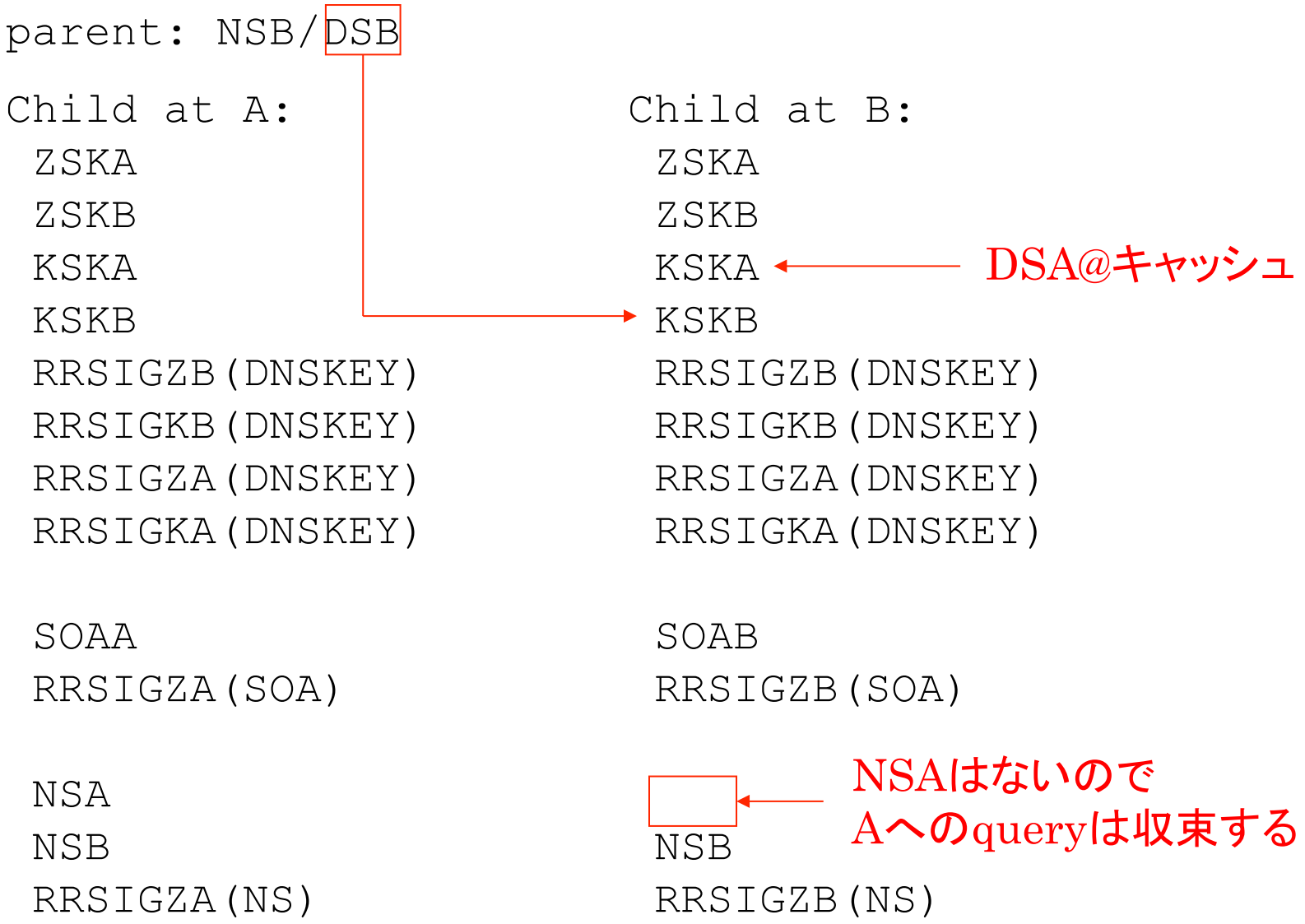
4.4.5.1 Cooperating DNS Operators(initial)

```
parent: NSA/DSA

Child at A:
 ZSKA

 KSKA



 RRSIGZA(DNSKEY)
 RRSIGKA(DNSKEY)

 SOAA
 RRSIGZA(SOA)

 NSA

 RRSIGZA(NS)
```

4.4.5.1 Cooperating DNS Operators(pre-publish)



4.4.5.1 Cooperating DNS Operators(Redelegation)



4.4.5.1 Cooperating DNS Operators(post migration)

parent: NSB/DSB

Child at B:

ZSKB

KSKB
RRSIGZB(DNSKEY)
RRSIGKB(DNSKEY)

SOAB
RRSIGZB(SOA)

NSB
RRSIGZB(NS)

4.4.5.2. Non Cooperating DNS Operators

- 非協力的な関係のときは、話はもっとややこしくなる。
- loosing operatorはDNSのデータを触ろうとしないだろう。
- ヒドいときには、TTLと署名の有効期間を長くしてDNSKEYを公開して、loosing operatorのDNSKEYが長い間キャッシュに残るようにするだろう。
  - 原理的には何十年も。

4.4.5.2. Non Cooperating DNS Operators

- 委任の変更が済んだ後にvalidatorがloosing operatorの鍵で署名を検証しようとし(キャッシュにつかんでいるから)、委任をたどっても、的確な署名が見つからないと、問題が起こる。
- gaining operatorがloosing operatorのRRSIGを全部コピーして、自分の秘密鍵で生成した署名と一緒に公開したとしても、ゾーンデータには一切の変更ができない。
  - 変更したら、loosing operatorの秘密鍵で再署名しなきゃいけないから。

4.4.5.2. Non Cooperating DNS Operators

- だいたい、委任先を変えるにはNS RRを変更しなきゃいけないから、この方法はうまくいかない。
- それに、loosing operatorがゾーン転送を許可せず、NSEC3を使っていたら(zone enumerateができない)、RRSIGを全部コピーすることはできない。

4.4.5.2. Non Cooperating DNS Operators

- registrantが踏み得る途中経過の唯一の選択肢は…
  - 一旦、署名をやめる。
  - レジストリに、loosing operatorのDSを消してもらう。
  - loosing operatorのDNSKEYと、それで生成した署名がキャッシュから消えるのを待つ。
    - でも何十年もかかり得る。

4.4.5.2. Non Cooperating DNS Operators

- いくつかの実装では、検証に失敗したDNSKEYをキャッシュしておく期間を制限して、復帰を試みる。
- これはプロトコル上の要請ではないが、非協力的なレジストラの問題を軽減はするだろう。
- 運用上の完璧な回避策はなく、根本的には契約で縛るしかない。

5. Next Record type

- DNSSECの開発において、その場で署名する代わりに、事前に署名しておいて提供する方法がトレードオフとして作成された。
- これを実現するためには、新しいレコードが必要だった。
- 大量の存在しないドメイン名を、存在するドメイン名の間に入れるレコードである。

5. Next Record type

平文を使う方法と難読化された文字列を使う方法の2つがある。

どちらも、以下の特徴を持っている。

- NSEC/NSEC3のオーナに存在する、全てのRRTYPEを含む
- そのゾーンが権威である名前のみを保存している(glueは削除される)
- 特定のRRTYPEが情報の保存に使われる。平文の場合はNSEC、難読化した文の場合はNSEC3である。

5.1. Differences between NSEC and NSEC3

NSECとNSEC3の違いは以下である。

- NSECでは、ゾーン名がソートされたリンクリストで実装されている。
- NSEC3では、一方向性ハッシュ関数でハッシュされたゾーン名が、ハッシュの文字列のソート順になっている。

5.1. Differences between NSEC and NSEC3

- NSECのレコードは暗号処理を要求しない。例外は、対応する署名の検証のみである。
- このレコード名は人間が読める内容で、手動でのクエリに使用可能である。
- 欠点は、”zone walking”をされることである。
- zone walkingとは、ゾーンの全てのエントリに対して、
- 次のNSECレコードを指している、next RRlabelを辿ることである。

5.1. Differences between NSEC and NSEC3

- しかし、全てクエリ可能なDNSのデータだとしても、NSECの副作用としてゾーンデータ全体に連続クエリをすることができてしまう。
- オペレータの中には、この挙動は受けいれられる、もしくは好ましいと思うものもいる。
- 一方、ポリシーとして好ましくない、規制したいと考える者もいる。

これが、NSECとNSEC3の1つ目の相違点である。

5.1. Differences between NSEC and NSEC3

NSECとNSEC3の2つ目の相違点を以下に示す。

- NSECはゾーンファイル内の各RRにサインを要求する。
- そのため、否定応答もサインを保証する。
- しかし、大きいゾーンファイルで、沢山の委任ゾーンがあり、少数のサインされたゾーンがある場合、受け入れがたいオーバーヘッドが発生する。ここではセキュアで無い委任は頻繁に更新されることを仮定している。典型例としては、TLDのオペレータで、数個のregistrantsがセキュアな委任を使用している場合かもしれない。

5.1. Differences between NSEC and NSEC3

- NSEC3では、2つのセキュアな委任に間隔を設けることができる。
- この間隔には、1つ以上のセキュアで無い委任を入れてもよい。

- NSEC3だと、(全てには署名しないために)ゾーンのサイズを削減できる。
- また、特定の間隔に名前が存在するかを暗号的な否定応答を返すコストの観点で、ゾーンの暗号的複雑さを減らすことができる。

5.1. Differences between NSEC and NSEC3

- NSEC3のレコードはリクエストされたRRlabelのハッシュを使用する。

- (zonewalkを行う人が)ゾーンのエントリ推測が必要である為、負荷を高くするために、NSEC3レコードではハッシュのIterationを指定できる。
- また、Saltを指定することができる。

- NSEC3はゾーンの情報漏れを完全に防ぐ手法ではない。

5.2. NSEC or NSEC3

NSEC3を開発する動機の1つ目はzone enumeration の防止である。

zone enumerationとは、zone walkingともいい、ゾーン内にデータが存在しないこと(不存在)を証明する際に、ゾーン内のデータの前後のデータを提示する形式をとるため、あるゾーンに登録されている全ての情報を誰でも取得できる状態になることを言う。

参考:

http://jpinfo.jp/event/2005/0825IETF.html

5.2. NSEC or NSEC3

- zone enumeration は、ゾーンの内容が高度に構造化されている場合や、簡単に推測できる場合に有効である。
- ここでいう高度に構造化されているゾーンとは、in-addr.arpaや、ip6.arpaや、 e164.arpaなどである。これらのゾーンでは、IPアドレスや電話番号などによって、簡単にenumerateできる。

- 一方ゾーンAPEXのレコードと、wwwやmailなどの推測しやすい一般的なRRlablelだけを持つ小さいゾーンは、NSEC3を使用した場合でも、簡単にenumerateできる。

5.2. NSEC or NSEC3

- 前述した、高度に構造化されたゾーンや、小さいゾーンの場合は、NSECの使用がオススメである。署名する人と、validateするリゾルバの両方に利点がある。

- 一覧を見る場合に、NDAへのサインが必要であるなど、容易に扱えないRRlabelを持った大きいゾーンの場合は、NSEC3の使用がオススメである。

- NSEC3を開発する2つ目の理由は、以下の5.3.4章で議論する。

5.3. NSEC3 parameters

- NSEC3のハッシュアルゴリズムは、FQDNを圧縮せずそのまま処理する。
- これは、他のFQDN RRlabelに対するbrute force attackで再利用できないことを意味する。なぜなら、攻撃目標であるドメイン名のエントリが一意であるから。

Q: 圧縮してしまうと再利用攻撃が可能?

→ ホスト部だけ(例: www)などを利用する攻撃の場合は利用可能かも。FQDNではないけど。

DNS name compressionの話ではないみたい。

5.3.1. NSEC3 Algorithm

NSEC3のアルゴリズムはDNSKEYのアルゴリズムと同じである。

現在のところ、以下の選択肢がある。

- アルゴリズム6: DSA-NSEC3-SHA1
  （これは、アルゴリズム3のDSAのエイリアス）
- アルゴリズム7: RSASHA1-NSEC3-SHA1
  （これは、アルゴリズム5のRSASHA1のエイリアス）
- next recordのアルゴリズム選択は、使用するDNSKEYのアルゴリズムだけに依存する。

アルゴリズム3,5ならば、NSEC、
アルゴリズム6,7ならば、NSEC3

5.3.1. NSEC3 Algorithm

Note:

3.1.4.1章と同様、RSASSA-PKCS1-v1_5や
RSASSA-PSSもSHA256も選択肢には無い。

5.3.2. NSEC3 Iterations

- NSEC3では、IterationsとSaltの2つの機構が、辞書攻撃に対する耐性を向上させている。
- 攻撃者は予めドメイン名からハッシュ文字列を作成しておくことで、ゾーンにあるドメイン名が存在するか否かを知ることができる。(辞書攻撃)
- しかし、RFC5155で考察されているように、署名のためのコストと、ネームサーバによるvalidationの為のコスト増は、辞書攻撃に対策する為のコストとのトレードオフである。

5.3.2. NSEC3 Iterations

- Iterationの便利に使用できる限界を、RSAの鍵長を元に目安として示す。
- Iteration 150回の場合は、RSAだと1024bit程度。
- Iteration 500回の場合は、RSAだと2048bit程度。
- Iteration 2500回の場合は、RSAだと4096bit程度となる。

最大の2/3程度がコスト的に十分だと考えられる。

例. RSA 1024bits程度 -> Iteration(MAX) 150回
-> その150回の2/3である100回がコスト的に十分なIterationの回数(目安)

5.3.3. NSEC3 Salt

- NSEC3では、Iterationパラメータを増やして辞書攻撃への耐性を高めている。
- 一方、NSEC3 Saltは事前に計算されたハッシュの使用できる生存時間を減らしている。
- Saltの値を変更することで、攻撃者が事前に計算したデータを無意味にすることができる。

5.3.3. NSEC3 Salt

- FQDN RRlabelの場合、これはハッシュ文字列の一部分になるのだが、既に使用されたRRlabel が brute force attackにおいて再利用されることは無い。
- これはFQDNが一意だからである。

- ゾーン内のNSEC3レコードにおいて、全てのSaltは同じである。
- Saltを変更した場合、全てのNSEC3レコードは再作成される。

5.3.3. NSEC3 Salt

- NSEC3レコードの再作成後に、新しいRRSIGが、NSEC3レコードを用いて作成される。Saltの変更は、ZSKの変更と連携するとよい。一連の作業で、RRSIGを再作成する必要があるためである。

- incremental signingの致命的な依存が無い場合や、ゾーン全体が簡単にサインできる場合は、この様な連携は不要である。
- ZSKの変更と異なり、NSEC3のSaltの変更は、特別なrolloverの手順は不要である。
- ゾーンの更新ごとにSaltを変更することも可能である。

5.3.4. Opt-out

- “Opt-out”の機構はゾーンの中身がほぼ委任されたレコードを持っている場合に、署名されたレコードをゾーンに徐々に入れる方法である。
- OPT-OUTフラグを使用することで、NSEC3スパンの意味を変更できる。
- Opt-outしないとき: NSEC3スパンで、そのスパンに存在する名前の権威ある否定
- Out-outしないとき: そのスパンに存在する委任にDNSSECは使用できない

[Editors Note:]

この議論を更に厳密にして、ハッシュ名にまで拡張する提案も出たが、編集者はそれは過剰だと信じている。

5.3.4. Opt-out

- これにより委任の追加/削除時に、NSEC3のRR chainに出てくる、RRの再計算やRRの再署名なしでNSEC3のRR chainができる。
- Opt-outの使用は委任ポイントと多数のゾーンとセキュアな委任の数が少ない場合に限る。
- この考えは、典型的にはTLD(トップレベルドメイン)や、それに似たゾーンに適用できる。
- その他のほとんどの状況では、Opt-outは適用すべきではない。

以降の検討はRFC5155 DNS Security (DNSSEC) Hashed Authenticated Denial of Existence Section 12.2 Opt-Out Considerations を参照のこと。

6. Security Considerations

(snip)

7. IANA considerations

(snip)

8. Acknowledgements

(snip)

9. References

(snip)

Appendix A. Terminology

本資料 p2-p8を参照

Appendix B. Zone Signing Key Rollover How-To

本資料 p68-72を参照

Appendix C. Typographic Conventions

(snip)

Appendix D. Document Editing History

(snip)