Zone State Revocation for DNSSEC

Eric Osterweil
UCLA
eoster@cs.ucla.edu

Vasileios Pappas
IBM
vpappas@us.ibm.com

Dan Massey
Colorado State University
massey@cs.colostate.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

DNS Security Extensions (DNSSEC) are designed to add cryptographic protection to the Internet's name resolution service. However the current design lacks a key revocation mechanism. In this paper we present Zone State Revocation (ZSR), a lightweight and backward compatible enhancement to DNSSEC. ZSR enables zones to explicitly revoke keys using self-certifying certificates, and enables DNS name-servers to opportunistically inform distributed caching resolvers of key revocations via lightweight control messages. Further, ZSR allows resolvers to distinguish between legitimate key changes and potential attacks when authentication chains are broken. ZSR is designed to work well with global-scale DNS operations, where millions of caches may need to be informed of a revocation, and where time is critical.

1. INTRODUCTION

The Domain Name System (DNS) [4] provides name resolution service for the global Internet and is collectively operated by millions of autonomous administrative domains. To secure this critical service, the DNS Security Extensions (DNSSEC) [6, 8, 7] have been developed to add cryptographic protection to DNS. DNSSEC uses public-key cryptography to authenticate DNS data. It leverages the existing DNS delegation hierarchy to create a Public-Key Infrastructure (PKI) where each zone signs the public keys of all of its children. However, after years of development efforts, a number of DNSSEC deployment issues remain open.

One of the remaining issues is emergency key revocation. DNSSEC has well defined procedures to let each zone change its public-private key pairs periodically under normal operations, but has no procedure to handle emergency situations, such as when a zone's private key is lost or compromised. With a zone's private key, an adversary (*Eve*) could hijack a zone's webservers, name-servers, or even create fictitious child zones. One could simply remove the compromised public key from the authoritative DNS servers, but this does not revoke the compromised key.

The fundamental challenges in implementing emergency key revocation come from DNS' heavy reliance on caching and its global scale. Due to caching, DNS data records, including essential DNSSEC keys and signatures, may be stored at multiple caching resolvers even after they have been removed from the authoritative servers. The unfortunate fact is that, due to the scale of DNS, there is no way to know how many and which DNS resolvers have cached the old keys and corresponding signatures that authenticate the old keys. Thus, there is no way to notify them about the key changes. *Eve* can replay stale public keys and stale authentication chains until their definitive lifetimes expire. The affected zones can be in a vulnerable state for weeks or months. What is needed is a revocation mechanism that can immediately revoke a zone's key(s), revoke the signatures generated by these keys, and inform DNS caching resolvers throughout the Internet to flush stale entries and re-fetch affected data.

In this paper, we develop the Zone State Revocation (ZSR) mechanism for handling emergency key revocations and revoking obsolete signatures at the Internet-scale. ZSR makes a simple observation about the current operation of DNSSEC that all signature lifetimes are temporal (based on an inception date and an expiration date), but emergencies reflect unplanned events that are not able to be captured by preset date ranges. In ZSR, the idea of a zone's state is formalized and used to augment DNSSEC's signing practices. Specifically, signatures include *state-lease* information in addition to dates, thus they can be invalidated if a zone's state changes beyond the specified lease. Additionally, ZSR defines a new revocation key resource record (REVKEY) that enables a DNS server to definitively revoke a key. We show that ZSR is a simple, incrementally deployable, and a backwards compatible enhancement to DNSSEC. ZSR offers the ability to i) let a server mark a public key as being revoked, ii) efficiently disseminate the information of a revoked key to the potentially large-scale number relevant resolvers, and iii) enable resolvers to distinguish between attacks and emergency key changes.

The remainder of this paper is organized as follows: Sections 2 and 3 describe the background and threat model. Section 4 describes the details of ZSR, Section 5 follows with the analysis that supports it, and Section 6 concludes.

2. BACKGROUND

DNS and DNSSEC: The DNS name space is divided into multiple sub-spaces called zones that represent sub-trees in the DNS name hierarchy. Each zone is identified by its "Start of Authority" (SOA) data record, which identifies a zone's namespace and its meta-data. A zone's data records are replicated in multiple authoritative name-servers. Typically, one authoritative server that holds a master copy of a zone resource records (RRs) and several other secondary servers replicate this zone data from the master server. After a data update, a *serial number* in the SOA record is incremented to signal to secondary servers that they should initiate a replication. The SOA record also plays an important role in our revocation scheme.

The DNS Security Extensions (DNSSEC) [6, 8, 7] use a public-key based cryptographic system to add origin authentication and data integrity to DNS services. The basic idea is relatively simple, but the process of adding cryptography to an existing large scale distributed system is non-trivial and DNSSEC's design has undergone several major revisions over the last decade. Below we review DNSSEC's central concepts, the current DNSSEC key management practices, and the revocation problem.

To prove that the data records in a DNS reply are authentic, each zone creates public/private key pairs to verify and sign each data record. The public key of a zone is stored in a record called a DNSKEY, and the signatures are stored in RRSIG records. In response to a query, an authoritative server returns both the requested data and its associated RRSIG records. A resolver that has learned the DNSKEY of the requested zone can use it to verify the *origin authenticity* and integrity of the reply data. To resist replay attacks, each signature carries a definitive expiration time. In order to accommodate the additional space needed for cryptographic signatures and keys, DNSSEC requires the use of Extension mechanisms for DNS (EDNS0 [10]). EDNS0 uses a meta record (called an OPT record) to allow DNS name-servers and resolvers to negotiate options such as packet sizes.

A zone's DNSKEYs can be retrieved via standard DNS requests, but their authenticity (i.e. if the key returned *indeed* belongs to the zone it claims) needs to be vouched by a parent zone's DNSKEY. This is called a *secure delegation*. In order to check the validity of a secure delegation, DNSSEC expects a resolver to construct a *chain of trust* that follows the DNS hierarchy from a trusted root to the zone in question. For example, the public key of the DNS root would be used to authenticate the public key of *edu* zone; the public key of *edu* zone would in turn be used to authenticate the public key of *ucla.edu* zone; and so forth. Although this key hierarchy seems simple in concept, managing it is challenging in practice, due to the necessary coordination *across different administrative domains.*

Secure delegations are implemented by storing only the hash (or finger-print) of a zone's public key at its parent zone. These records are called *Delegation Signing* (DS) records, and are signed by the parent's key. When a child zone changes its public key, it must request an update to the DS record in its parent's zone so that the parent can sign the new DS record.

Conceptually, each secure zone has a single public key-pair. The private key signs the zone data, the public key verifies signatures, and the public key matches a signed DS record at the zone's parent. Cryptographic keys should be changed periodically to reduce the chance of compromise, but any coordination across administrative boundaries is potentially slow, which makes key changes harder to manage. For this reason, zone operators are encouraged to use two

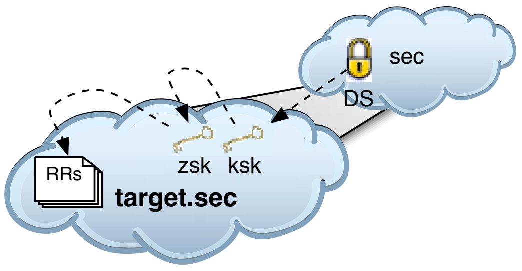

Figure 1: Here *target.sec's* DS record is stored at its parent (*sec*). Resolvers can query *sec* for *target.sec's* DS record, and then *target.sec* for its KSK. Confirming the DS record verifies the KSK completes the secure delegation from *sec* to *target.sec*. Resolvers can then verify that *target.sec's* KSK has signed for the ZSK, and that the ZSK has signed for each resource record (RR) in the zone.

distinct public key-pairs. One public key is called the Key Signing Key (KSK). The KSK is created by the zone administrator and should match the DS record, signed and stored at the parent zone. This is how a secure delegation is implemented. Anytime the KSK changes, the zone must notify its parent zone so the DS record can be changed accordingly. This KSK *could* be used to sign the zone's data records, but the addition of a second public key pair will greatly simplify operations.

In order to reduce coordination with the parent zone, administrators are encouraged to generate a second public key pair and use this second private key to sign the zone data. Since this second public key is used to sign the all the zone data, it is called the Zone Signing Key (ZSK). A resolver must first authenticate the KSK and then use the KSK to authenticate the ZSK and finally use the ZSK to authenticate the actual zone data. Figure 1 shows a portion of the authentication chain leading from a fictional top level domain (TLD), *sec* to a zone *target.sec*'s KSK and ZSK. The duality of having 2 keys complicates matters conceptually, but operationally it reduces coordination requirements between parent and child. KSKs are used less frequently (they only sign ZSKs) and are, thus, allowed to change less frequently. The ZSKs are not verified by zones' parents (like the KSKs are) but rely on a signatures produced by KSKs instead. Since zone operators control all of their own keys, ZSKs can be changed at any time without notifying the parent zone (only a KSK change requires a DS update at the parent).

DNSSEC Key Management: The DNS (and by extension, DNSSEC) relies heavily on caching for scalability and performance, but does not have any type of cache-coherence control. It is, therefore, not surprising that, at any given moment, some caching resolvers may hold stale records. This fact has greater implications in a secure system than in the current DNS. In the case of DNSSEC, stale DNSKEY records (those that a zone administrator would like to revoke) can be exploited by an adversary.

DNSSEC does not have any mechanism for explicitly revoking keys. A zone's administrator can remove the DNSKEY record of a public key from the authoritative servers only after it is no longer in use and its signatures *should* be no longer present in caches according to the TTL value [5]. However *removing* this DNSKEY record from the authoritative servers does not *revoke* the key and does not assure that it has been *removed* from the DNS system. The key may still be present in some caches and an attacker can continue to replay the removed DNSKEY and repopulate the caches until the signatures associated with the DNSKEY or its authentication chain expires.

In cases when emergency key revocation is necessary (such as the private portion of the key is compromised), DNSSEC currently does not offer any good operational choices to handle the situation. If the zone operator immediately removes the compromised key, resolvers and caches that have cached the old key would not be able to validate signatures, which could create a great deal of collateral damage and result in self-inflicted denial of service events.

DNSSEC's signatures specify the period during which they are valid. Forecasting this period is "optimistic," but is (arguably) used as a substitute for the missing revocation system. By assigning inception and expiration dates (a specific lifetime) for its signatures, DNSSEC essentially assumes that records will be valid in the future. Motivated by the key management problems that DNSSEC faces, we developed a Zone State Revocation mechanism that allows zones to rapidly revoke keys and to explicitly mark keys as revoked.

3. THREAT MODEL

DNSSEC's cryptographic signatures protect users against spoofing, assuming an adversary (*Eve*) is unable to produce cryptographic signatures for data that she intends to publish, and these signatures are verified by a zone's (such as *target.sec's*) DNSKEYs. However, if *Eve* is able to obtain the private (or signing) portion of *target.sec's* keys, then she can spoof data and provide seemingly valid signatures. We now enumerate several specific attack scenarios (what *Eve* may do) and attack vectors (how she may do it).

3.1 Attack Scenarios

In the event that *Eve* has obtained the private portion of a zone's ZSK, she will be able to serve potentially malicious data by injecting her own records, signed by a previously valid key.

Eve may cause noticeable service disruptions by creating signed:

1. *End Host Records:* *Eve* may create a new *A* record such as *www.target.sec*, that points to a malicious host, to get *Alice* to access it.
2. *Nameserver Records:* *Eve* may sign bogus NS+A records that redirect all DNS queries to her own servers, and cut the real *target.sec* out.

In addition, if *Eve* is able to compromise a zone's KSK, she can create new ZSKs at will and launch the above attacks. This is because ZSKs are validated by signatures from KSKs. Therefore, the ZSK attacks are a subsets of the possible attacks that can be launched with a compromised KSK.

3.2 Attack Vectors

In order to attack, *Eve* must insert forged records about *target.sec* in a caching server C. To do this, the relative locations of *Eve*, C, and *target.sec*'s name-servers in the network influence the success of the attack. Next we enumerate the

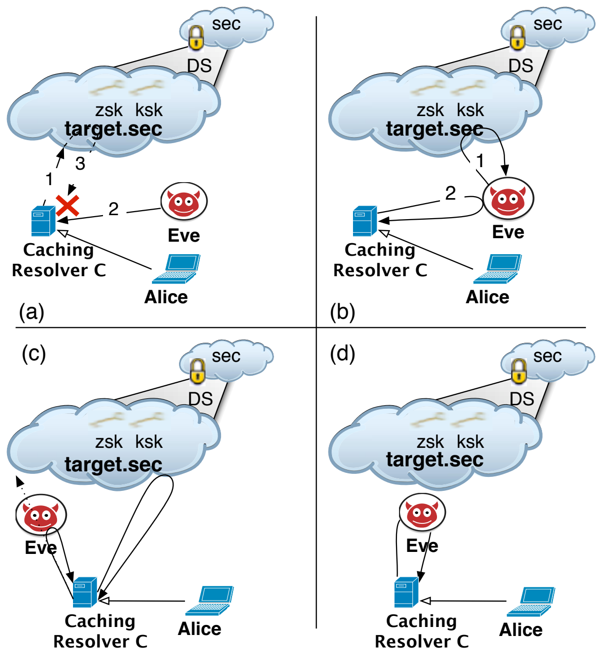

Figure 2: **Figure (a) depicts a spoofing attack whereby Eve returns a message to C before *target.sec* can. Figure (b) depicts a cache poisoning attack in which Eve inserts records into C by piggy-backing on a valid request. Figure (c) shows a man-in-the-middle attack in which Eve can intercept some of the traffic bound for *target.sec*. Figure (d) shows a man-in-the-middle attack in which Eve is able to intercept all of the traffic bound for *target.sec*.**

possible capabilities of *Eve* and her feasible attack vectors (short of full control of all network traffic):

1. *Spoofing Attack:* Figure 2(a) shows the adversary spoofing the DNS traffic between *target.sec* and C. This attack can be successful only if the adversary can predict the IDs of the DNS packets[2]. Thus, *Eve* can forge only some records of *target.sec*, and not all DNS traffic.

2. *Poisoning Attack:* In Figure 2(b), *Eve* receives valid DNS requests for her own data from C and replies by appending forged records that belong to *target.sec*[3]. This can happen if *Eve* controls a server that C happens to use while resolving names, possibly names that are not related to *target.sec*.

3. *Man-In-The-Middle Attack (1):* In Figure 2(c) *Eve* has control over the DNS exchanges between *target.sec* and C. She can snoop, block, and insert packets in the communication pipe between C and *target.sec*. However, she can only control the communication pipes between C and some of the name-servers of *target.sec*.

4. *Man-In-The-Middle Attack (2):* In Figure 2(d), *Eve* again has control over the DNS exchanges between *target.sec* and C, but now she can control all the traffic for (just) the name-servers of *target.sec*.

3.3 Attacks

The effectiveness of these attacks is illustrated by the following examples. Let us assume *Eve* has compromised the private portion of *target.sec's* ZSK.

If *Eve* is positioned in the same network as *Alice*, then it may be possible for *Eve* to snoop *Alice's* DNS requests, and the subsequent resolver request. In such a case, *Eve* can use attack vector 1.

Alternately, if *Eve* is an operator at any authoritative zone (such as *target.eve*), any traffic to her servers (at any point in time) will allow her to respond with additional records (beyond the specified question), and *Eve* may include fictitious records from *target.sec*. This optimistic caching policy of DNS' allows *Eve* to use attack vector 2.

If *Eve* is well positioned on a network that also hosts one of *target.sec's* name-servers, then she can try to intercept requests from *Alice*, and respond with her own responses[1]. While positioned in this way, *Eve* would be able to use attack vector 3.

Finally, if *Eve* were able to intercept all outbound traffic from *Alice's* network (perhaps by being positioned at her egress router), then *Eve* could disrupt all of *Alice's* services and effectively cut off queries to any of *target.sec's* name-servers (attack vector 4).

In Section 4 we demonstrate how ZSR combats these attack vectors.

4. ZONE STATE REVOCATION

An emergency key revocation protocol for DNSSEC needs to embody several qualities: i) it must be able to *prove* that a key has been revoked, ii) it must be able to revoke signatures from the revoked key, and iii) it must be able to rapidly notify resolvers and caches that make use of the affected zone's data.

ZSR addresses these needs by: i) creating self-certifying revocation certificates for revoked keys (called REVKEYs), ii) bounding the validity of signatures so that they can be invalidated at any time by dramatically changing the zone's state (represented by its SOA serial number), and iii) including the zone's serial number state in *each* DNS response from its authoritative servers. The specific details are described below in the next 3 subsections, respectively.

4.1 Proving Key Compromise

To provably indicate that a zone's key has been revoked, we introduce a new type of DNS resource record called the *REVKEY*. The format of the REVKEY is identical to that of the existing DNSKEY and thus uniquely identifies the public key to be revoked. To ensure that only the valid owner can revoke the public key, the REVKEY *must* be signed by its corresponding private key.

The REVKEY records act as self-certifying revocation certificates. Each REVKEY is a public key from the zone that is served with the standard signature(s) that DNSSEC attaches to all records. However, the REVKEY will *also* have a signature from its own private portion attached. In this way, any resolver that receives a REVKEY can use it to verify its own attached signature. If the REVKEY verifies any of the attached signatures, the key must be considered to be revoked. If the key cannot verify any of the attached signatures, then nothing can be concluded and the REVKEY must be ignored.

As an example: suppose a zone administrator, *Alice* wants to revoke a DNSKEY *Key1*. She first creates a REVKEY record for *Key1* by simply replacing the type name "DNSKEY" with "REVKEY", then signing the REVKEY with its own private portion, and storing the signature in an RRSIG. The REVKEY and its corresponding signature records will be served from the zone's authoritative servers. Note that *Eve* cannot revoke *Key1* unless she has possessed the corresponding private key. When a resolver obtains the REVKEY it comes with its own signature and the resolver can do the verification, without any further external queries.

4.2 Revoking Data

The mechanism by which DNSSEC validates resource records is by creating companion signatures (RRSIGs) that have both the standard DNS caching TTL, and inception and expiration dates that create a lifetime during which the signature validates the resource records. In fact, DNSKEY lifetimes are dictated by the RRSIGs attached to them. The difficulty in managing emergency key revocation with this mechanism is that emergencies tend to be unplanned, and therefore, do not fit into pre-generated signature lifetimes. In other words, if a zone operator (say, *Alice*) generates signatures for her zone's DNSKEY records that span a month, and then must revoke her DNSKEY a week later, the signatures will still appear to be valid.

This creates a logical dilemma for *Alice*, as she must now weigh the costs and benefits of signing frequently and incurring larger computation overhead (and perhaps needing to acquire more robust server hardware), or signing infrequently and leaving her zone vulnerable during emergencies.

ZSR augments the RRSIG inception/expiration values with a numerical *lease* (a number that is based on the zone's SOA serial number) that compliments the lifetime. The lease simply adds a notion that indicates if the zone has not changed its state by very much since the signature was generated, then resolvers may use the RRSIG's lifetime. If, however, the zone's state (serial number) has transitioned, and exceeded the lease specified by the signature, the record must be flushed from the cache and re-fetched and the resolver should query the zone for any REVKEYs.

In ZSR the RRSIG record format is modified to include the serial number lease. This new field in the signature record adds another dimension for validating a zone's signatures so that resolvers can confirm if any of their records are dirty during an emergency. If an adversary (*Eve*) has launched an attack using a zone's ZSK, *Alice* may feel secure by creating a REVKEY for the compromised key, breaking the lease on her current signatures by immediately advancing her zone' s serial number, and using a new key that is signed by her zone's KSK(s).

If *Eve* has compromised *Alice's* KSK, it is important for *Alice* to create a REVKEY (as above), and to break the secure delegation from her parent zone (*sec*). For this reason, we propose that DS records also include lease periods so that *Alice* can break her DS lease when she needs to. With a DS lease, resolvers who are walking the chain of trust can decide if a secure delegation is still valid.

Whereas the signature lifetimes in RRSIGs are temporal,

[1]DNS best practices mandate that zones position their name-servers in distributed locations, so it may be infeasible for *Eve* to be located on all of networks that host *target.sec's* name-servers.

this serial number lease can be broken at *any time* if an operator wants to signal an emergency revocation.

Using ZSR, *target.sec* can use its current serial number + x to create RRSIG records with leases. Then the zone can tell *sec* (its parent) to sign a DS record that includes a serial number that is x larger than the current value in *target.sec*[2]. Operators should choose x in such a way so that during typical operations of *target.sec* it is unlikely that the serial number will exceed its current value + x before the RRSIGs or the DS expire. For this reason, x should be a relatively large value and as a result, when a zone does intend to break its lease, it should increase its serial number by 2^{31}. We note that this is largest increase that is permissible for a zone's serial number at any given time. In Section 5.2 we provide evidence, through a sampling of active zones, that over 99% of SOA serial number changes are below this threshold. Thus, a large change of this kind is exceedingly uncommon in modern DNS practices.

For example, suppose on April 28th *target.sec's* SOA serial number were 2007042801 and an operators (*Bob*) signed the zone's records with a expiration date of May 5th, 2007 and a lease of 2008042801. If a day later *Bob* wished to revoke the zone's key, he would simply create a REVKEY and increase the zone's SOA serial to something large, such as 4154526449 (the zone's serial number + 2^{31}). This signals caches that a key revocation may be in progress, and they can then request the REVKEY for *target.sec* and see this is the case.

4.3 Notifying Resolvers

Informing resolvers about a revoked key must be done as quickly as possible in order to minimize the amount of time that a zone is vulnerable to attacks.

ZSR disseminates revocation notifications via lightweight control message that are embedded in *every* DNS response that is sent only from a zone's authoritative servers. By ensuring that control messages are only sent from authoritative servers (instead of intermediate caches), ZSR avoids attack vector 2. The messages sent indicate the instantaneous state of a zone and signal "revocation" state transitions.

In the event of a state transition message, caches should verify which keys have been revoked (by requesting REVKEYs), and then flush records that are now stale, and re-fetch any affected records (such as DNSKEY records, NS records etc.).

One of the key benefits of this protocol is that it allows name-servers to inform resolvers of emergency key revocations with every query. Without ZSR's notification messages, resolvers may try to observe a zone's state through examining its SOA record. However, this approach has only limited usefulness as caching can cause stale records to mask revocation conditions. For example, during a revocation, when *target.sec* is queried for its SOA record, the resolver will be able to inspect the serial number and identify that it has increased beyond the lease period of cached signatures. In addition, whenever *target.sec* is queried for a record that does not exist, it *may* return its SOA[3]. Unfortunately, a resolver may be served a stale SOA from another resolver's cache and it may, therefore, be difficult to rely on getting fresh SOAs in a timely fashion when their TTLs allow them to remain cached during a revocation. For this reason, ZSR uses the EDNS0 [10] OPT record.

The OPT record has 2 relevant requirements: i) caches are not allowed to cache OPT records, so resolvers can be confident that they are fresh, and ii) servers can embed arbitrary code/value pairs in them.

[2]*Alice* may choose different lease periods for RRSIGs and DS records because of their different implications.

In ZSR, authoritative name-servers embed the following information in the OPT record of all response traffic: *soa-serial-number* and a 4-dimensional tuple value. The format of this tuple is:

$$\langle zone\_name, serial\_number, timestamp, signature \rangle$$

This signature is used by resolvers to verify a serial number that indicates a revocation. This requirement keeps *Eve* from falsely signaling a revocation. During a revocation, this signature must be from the compromised key (REVKEY). This ensures that an adversary has not used a fictitious key with either of attack vector 3 or 4 to falsely signal a revocation.

4.4 ZSR Examples

ZSR's mechanisms are able to address all attack vectors specified in Section 3 short of an adversary that intercepts every query to *target.sec's* name-servers.

In the event that *Eve* were to attempt attack vector 1, her initial record may become cached. However, any subsequent queries to *target.sec* (for any record at all, not just the previous query) would allow *target.sec* to inform the cache that the zone's state has transitioned. The cache could then request REVKEYs and see that *Eve's* record is dirty and it would be flushed.

If *Eve* were to attempt attack vector 2, caches would be able to flush her record with exactly the same efficiency as attack vector 1.

Attack vector 3 has *Eve* positioned so that she can intercept all traffic to one of *target.sec's* name-servers. In this situation, *Alice's* cache may be unfortunate enough to encounter *Eve's* name-server some of the time. However, the very first time *Alice's* cache encounters one of *target.sec's* real name-servers for any query, she will receive a message that the zone's state has transitioned. The cache will then fetch the REVKEYs from *that* name-server, and see that the other signatures and key are invalid.

Attack vector 4 allows *Eve* to intercept all traffic bound for any of *target.sec's* name-servers. In this case, *Alice* will not be able to receive messages indicating a rollover state with the same frequency that she queries for names from *target.sec*. However, by periodically walking the chain of trust (from the root or *sec*) she will see when *sec* has created a new delegation to *target.sec* and realize the old KSK has changed. In this case, the response time will not be proportional to the query rate, but *Alice* will be able to observe a key change.

5. FEASIBILITY ANALYSIS

Below we analyze a sample of actual DNS information to justify that simply waiting for key expiration will not suffice for an emergency rollover plan and show how ZSR outperforms this approach.

5.1 Signature Lifetime

The current deployment of DNSSEC is fairly small. Using an existing DNSSEC monitoring project [1], we observed that DS signature lifetimes currently span from 3 days to 30 days, with an average of 17.03 days. In addition to the DS records, 31 zones signed their DNSKEYs for periods ranging

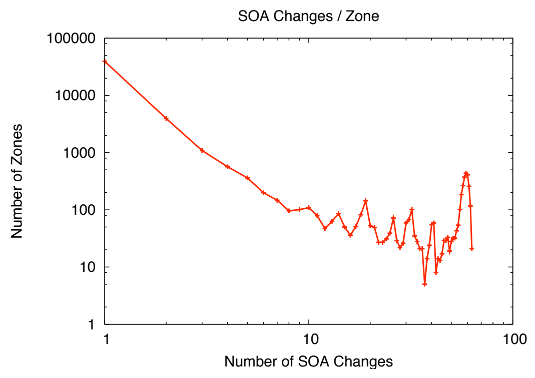

Figure 3: **This figure shows the number of zones that underwent SOA serial number changes. The x-axis reflects the number of changes, and the y-axis is the number of zones that changed with that frequency. Note that this figure is drawn to log-scale.**

from 3 to 30 days, with an average of 26.45 days.

The average lifetimes of signatures is several weeks. ZSR offers key revocation notification on a timescale that directly correlates to the query traffic of a zone. Caching resolvers visit popular zones frequently, thus can learn about the zone's key revocation quickly once it occurs.

5.2 SOA Serial Number Analysis

The SOA serial number is treated very carefully by ZSR because it is already used DNS zones. For example, it signals the need for a zone transfer from a master to a slave.

We conducted a month-long study to justify the feasibility of ZSR, and its use of SOA serial numbers. Due to the fact that DNSSEC has not been widely deployed yet, we studied the behavior of 50,000 typical insecure DNS zones' SOA serial numbers during May 2006. These zones were randomly chosen from a set of 2.5 million unique zones.

Figure 3 shows that while more than 80% of these zones did not change their SOA serial numbers during the monitoring period, some zones, in the tail, show a large amount of activity (seen as a spike in the curve).

Our results also show that most zones have very little dynamism in their SOAs, and only a small percentage show a high degree of activity. Figure 4 shows that 95% of the serial number changes that occured did so within 27 hours of previous changes. We can see that only a small number of zones change very frequently, and that among these zones the average period is 13.5 hours. We can, therefore, reason that the total number of changes needed (even highly active zones) is still relatively low considering the very large precision of the serial number.

Further, analysis of the serial number gaps (i.e. the relative change in the value of the serial number at every stage) is shown in Figure 5. We can see from this figure that there is some symmetry between positive and negative changes. Inspection of those serial gaps shows that many zones miscalculate their SOA serial numbers in such a way that the numeric representation of the date[9] is not zero-padded. To describe this, let us take an example of a real zone from our traces. We call the zone *missconfig.example*, and it had initial SOA serial number of 20065275 (which should, perhaps, have been 2006052705). Upon re-visiting

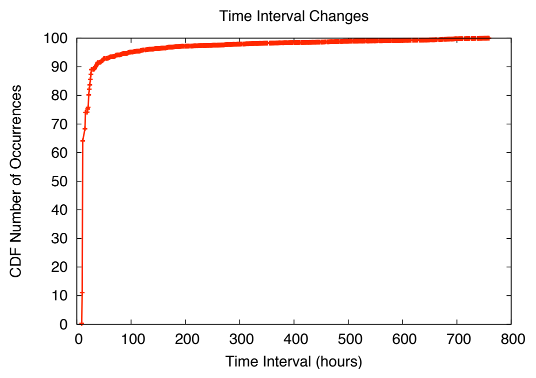

Figure 4: **This figure shows the CDF of time gaps between SOA changes. The x-axis reflects the number of hours between any 2 changes, and the y-axis is the CDF of the number of zones that changed SOAs at this period.**

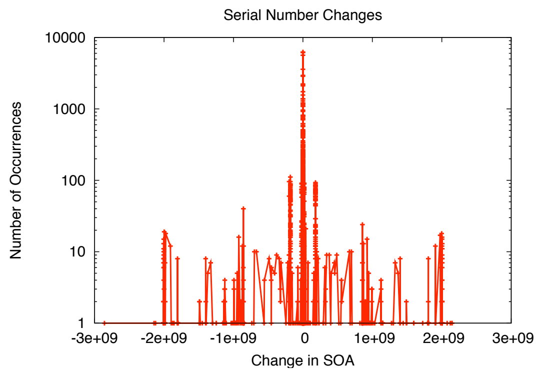

Figure 5: **This figure shows the sizes of serial number gaps between observations. Note that this figure's y-axis is drawn to log-scale.**

this zone, we observed that its serial number had changed to 200652716. The difference between these values was: $200652716 - 20065275 = 180,587,441$. Our next visit to the zone showed a serial number of 20065284. The difference this time was: $20065284 - 200652716 = -180,587,432$. Failure to properly zero-pad the computed serial number results in oscillating precision that explains the strange symmetry seen in Figure 5.

We can see from Figure 5 that there are large peaks between about -2.2^8 and 2.2^8, which is the precision of the oscillation in the preceding example. When counting the number of increments from all points between these peaks (inclusive), we account for approximately 99.2% of all SOA changes seen. This indicates that most SOA changes are noticeably less than the maximum allowed 2^{31} value, and if a zone using ZSR were to change its serial number by this much, it would stand out as being in less than 1% of zone changes.

5.3 Zone Access Patterns

In this section we describe the results of overlaying ZSR on top of actual DNS traffic taken from a north-American university. The traffic captures all the questions issued by the stub resolvers to the local caching server, for the duration of one month. There are 821 unique stub-resolvers that query 117,540 different names belonging to 55,632 DNS zones. This trace allows us to simulate the client traffic to a local caching server, and by extension, the outbound traffic from the caching server. In the simulation we model the effects of an adversary (*Eve*) that has compromised the KSK of *target.sec*. The adversary is capable of mounting any of the attack scenarios described in Section 3.

Window of Vulnerability: When considering the effects of *Eve*'s attacks, we must first examine the window during which caches are vulnerable to attack. We used our DNS trace to construct access patterns to each zone. As one can see from Figure 6, unmodified DNSSEC has a window of vulnerability that is independent of the type of the attack (the red solid line). In contrast, the window of vulnerability for DNSSEC with ZSR varies for the different types of attacks. It is shorter for the first and second type of attack compared to the third and forth type of attack.

The window of vulnerability is much shorter if ZSR is deployed. For example, the window is more than 6 days for around 88% of the zones without ZSR, and only for 17% of the zones with ZSR. However, the results seen in Figure 6 are heavily biased by those zones that receive an initial hit, and are not queried thereafter. Such zones appear to have longer windows of vulnerability, but actually are just unpopular and do not receive traffic during our simulated attack.

Query-Based Vulnerability: The actual vulnerability of a site is reflected by its query traffic. The preceding analysis simply outlined that caches may contain data from *Eve*. However, active zones will benefit from ZSR in that caches will receive messages that inform them of emergencies. To illustrate this, Figure 7 shows that in the case of DNSSEC with ZSR even if some zones have a window of vulnerability they are not affected by the attack. The reason is that the first query that they send out is able to invalidate the signatures of compromised keys. Indeed, the figure shows that only 5% of the zones are vulnerable to the first or second type of the attack, and around 10% of the zones are vulnerable to the second type of attack. Furthermore, the number of queries that end up using the signatures of the compromised keys are much smaller compared to the case of the DNSSEC without ZSR.

6. CONCLUSION

In this work we have described ZSR, a simple enhancement that is completely compatible with DNS' protocol, and requires very few modifications to current DNSSEC records and mechanisms. These changes are: modification of resolvers' caching behavior, the introduction of new type of key (the REVKEY), and a simple extension to RRSIG and DS record types. We note that the typical handling of record-type modifications is subject to much operational debate, and the results are often to create new types (such as the evolution from NSEC to NSEC3). However, the actual changes require a very small development footprint and solve the very large, and real, operational problem of emergency key revocation in DNSSEC.

Our analysis shows that without ZSR, zones that suffer a key compromise are vulnerable to attack for days. With ZSR, operators are able to revoke a ZSK and rollover to a new key within minutes to hours. In addition KSKs can be revoked just as fast and are done so in such a way so that the chain of trust can be allowed to recover within the

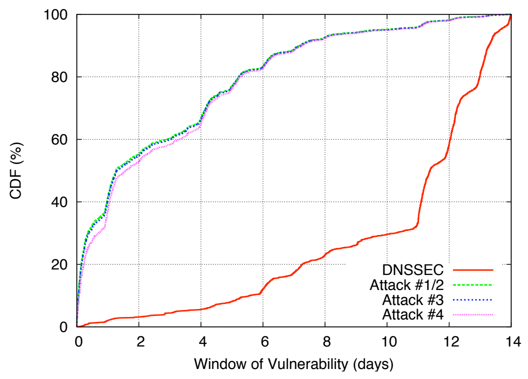

Figure 6: **This figure shows the CDF of zones that are vulnerable to attacks over the course of days. The curve labeled DNSSEC represents the behavior of DNSSEC under all of the attach scenarios. The remaining curves show the performance of DNSSEC + ZSR under the different attack vectors.**

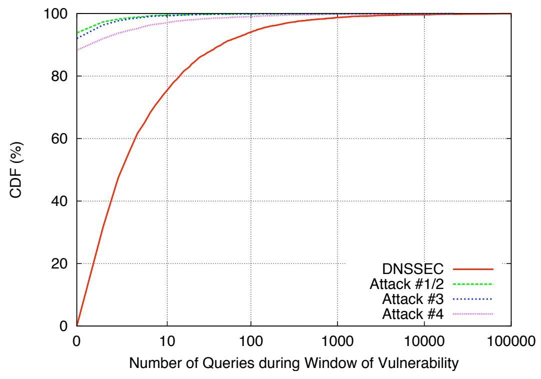

Figure 7: **This figure shows the number of queries needed to be able to successfully get fresh data to a caching resolver that is under attack. The curve labeled DNSSEC represents the performance of unmodified DNSSEC under all scenarios. The remaining curves show DNSSEC + ZSR under the different attack vectors.**

operational cycles that exist across administrative domains today. During this time, resolvers can be made aware of threats.

Though ZSR does not directly solve the problem of verifying new KSKs after compromises, it does allow client resolvers to recognize that an emergency revocation is in progress at extremely large scales. The relatively small number of modifications needed to support ZSR and these procedures make the idea ideal for prototyping and real-world implementations.

We intend to investigate the development of ZSR and analyze its performance. Following successful prototyping, we hope that engaging the operational community for feedback and support will lead to hardening and deployment of ZSR in DNSSEC.

7. REFERENCES

[1] Secspider. http://secspider.cs.ucla.edu/.
[2] Steven M. Bellovin. Using the domain name system

for system break-ins. pages 199–208.

[3] P. Mockapetris. Rfc 1035. RFC 1035, IETF, November 1987.

[4] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM '88*, pages 123–133, 1988.

[5] R. Gieben O. Kolkman. Dnssec operational practices. Internet Draft, DNSOP, March 2006.

[6] M. Larson D. Massey S. Rose R. Arends, R. Austein. DNS Security Introduction and Requirement. RFC 4033, March 2005.

[7] M. Larson D. Massey S. Rose R. Arends, R. Austein. Protocol Modifications for the DNS Security Extensions. RFC 4035, March 2005.

[8] M. Larson D. Massey S. Rose R. Arends, R. Austein. Resource Records for the DNS Security Extensions. RFC 4034, March 2005.

[9] R. Bush R. Elz. Rfc 1982. RFC 1982, August 1996.

[10] P. Vixie. Rfc 2671. RFC 2671, IETF, August 1999.