How DNSSEC Works | Cloudflare

The domain name system (DNS) is the phone book of the Internet: it tells computers where to send and retrieve information. Unfortunately, it also accepts any address given to it, no questions asked.

Email servers use DNS to route their messages, which means they're vulnerable to security issues in the DNS infrastructure. In September 2014 researchers at CMU found email supposed to be sent through Yahoo!, Hotmail, and Gmail servers routing instead through rogue mail servers. Attackers were exploiting a decades-old vulnerability in the Domain Name System (DNS)—it doesn't check for credentials before accepting an answer.

The solution is a protocol called DNSSEC; it adds a layer of trust on top of DNS by providing authentication. When a DNS resolver is looking for blog.cloudflare.com, the .com name servers help the resolver verify the records returned for cloudflare, and cloudflare helps verify the records returned for blog. The root DNS name servers help verify .com, and information published by the root is vetted by a thorough security procedure, including the Root Signing Ceremony.

A Gentle Introduction to DNSSEC

DNSSEC creates a secure domain name system by adding cryptographic signatures to existing DNS records. These digital signatures are stored in DNS name servers alongside common record types like A, AAAA, MX, CNAME, etc. By checking its associated signature, you can verify that a requested DNS record comes from its authoritative name server and wasn't altered en-route, opposed to a fake record injected in a man-in-the-middle attack.

To facilitate signature validation, DNSSEC adds a few new DNS record types:

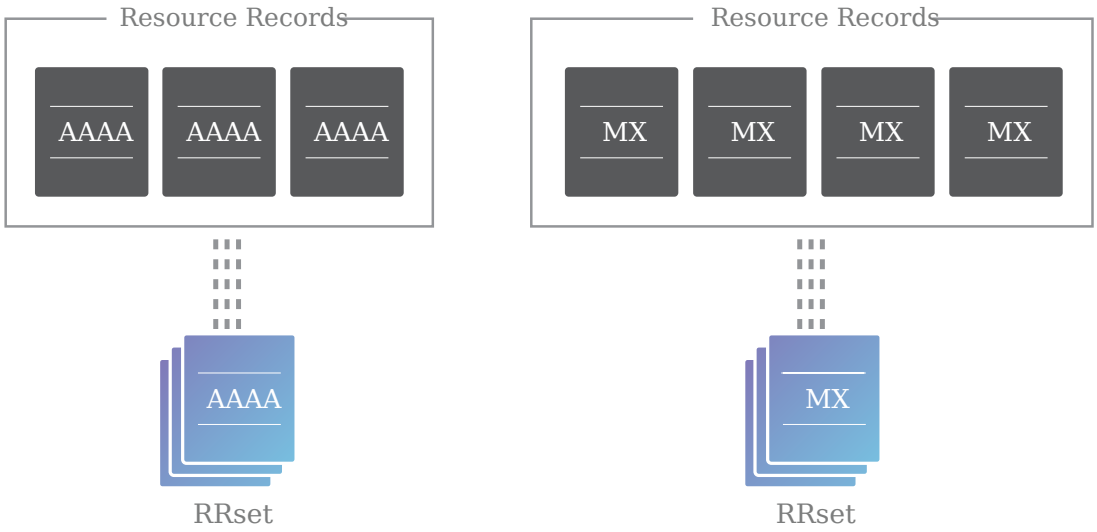
- **RRSIG** - Contains a cryptographic signature
- **DNSKEY** - Contains a public signing key
- **DS** - Contains the hash of a DNSKEY record
- **NSEC** and **NSEC3** - For explicit denial-of-existence of a DNS record
- **CDNSKEY** and **CDS** - For a child zone requesting updates to DS record(s) in the parent zone.

The interaction between RRSIG, DNSKEY, and DS records, as well as how they add a layer of trust on top of DNS, is what we'll be talking about in this article.

RRsets

The first step towards securing a zone with DNSSEC is to group all the records with

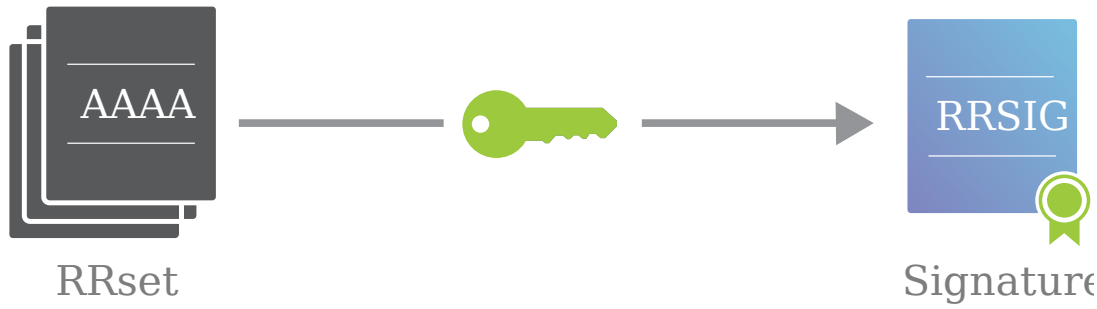
the same type into a resource record set (RRset). For example, if you have three AAAA records in your zone on the same label (i.e. label.example.com), they would all be bundled into a single AAAA RRset.



It’s actually this full RRset that gets digitally signed, opposed to individual DNS records. Of course, this also means that you must request and validate all of the AAAA records from a zone with the same label instead of validating only one of them.

Zone-Signing Keys

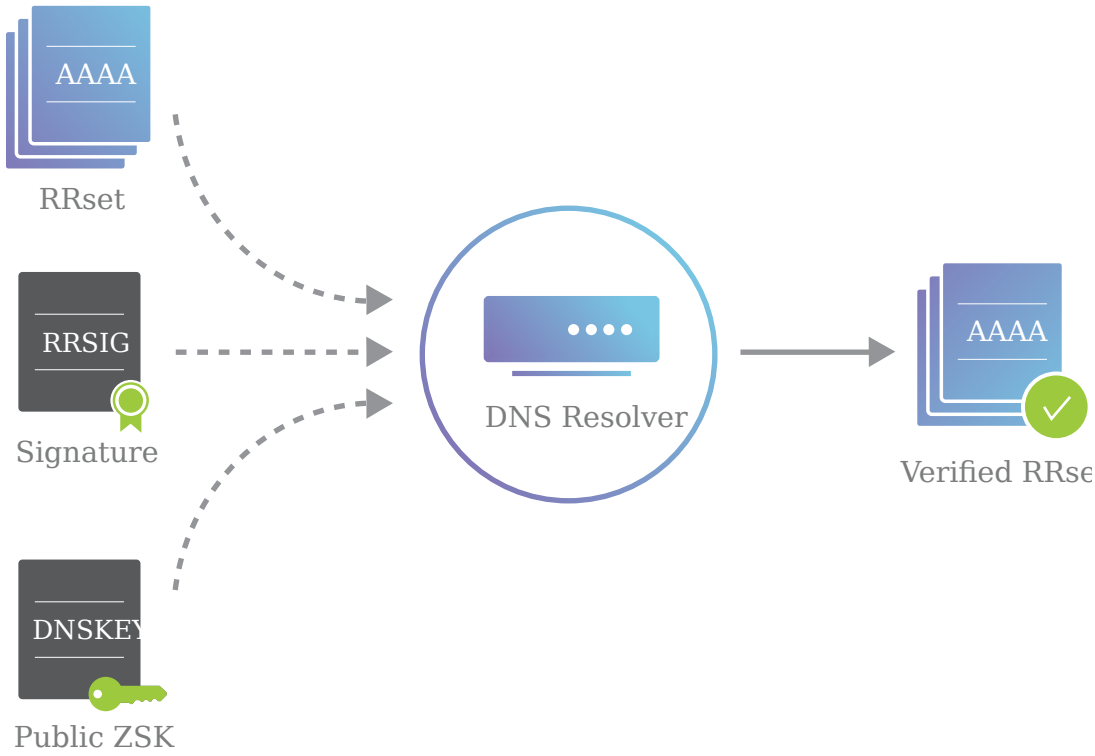
Each zone in DNSSEC has a zone-signing key pair (ZSK): the private portion of the key digitally signs each RRset in the zone, while the public portion verifies the signature. To enable DNSSEC, a zone operator creates digital signatures for each RRset using the private ZSK and stores them in their name server as RRSIG records. This is like saying, “These are my DNS records, they come from my server, and they should look like this.”



However, these RRSIG records are useless unless DNS resolvers have a way of verifying the signatures. The zone operator also needs to make their public ZSK available by adding it to their name server in a DNSKEY record.

When a DNSSEC resolver requests a particular record type (e.g., AAAA), the name

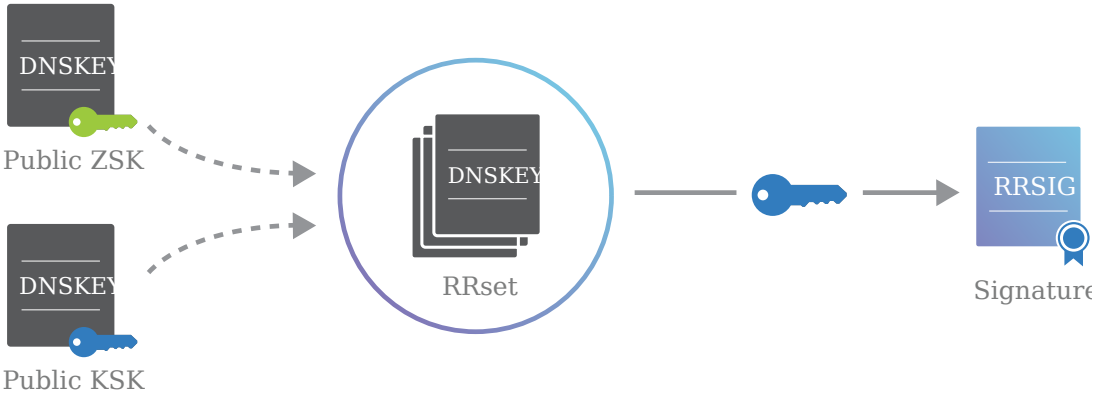
server also returns the corresponding RRSIG. The resolver can then pull the DNSKEY record containing the public ZSK from the name server. Together, the RRset, RRSIG, and public ZSK can validate the response.



If we trust the zone-signing key in the DNSKEY record, we can trust all the records in the zone. But, what if the zone-signing key was compromised? We need a way to validate the public ZSK.

Key-Signing Keys

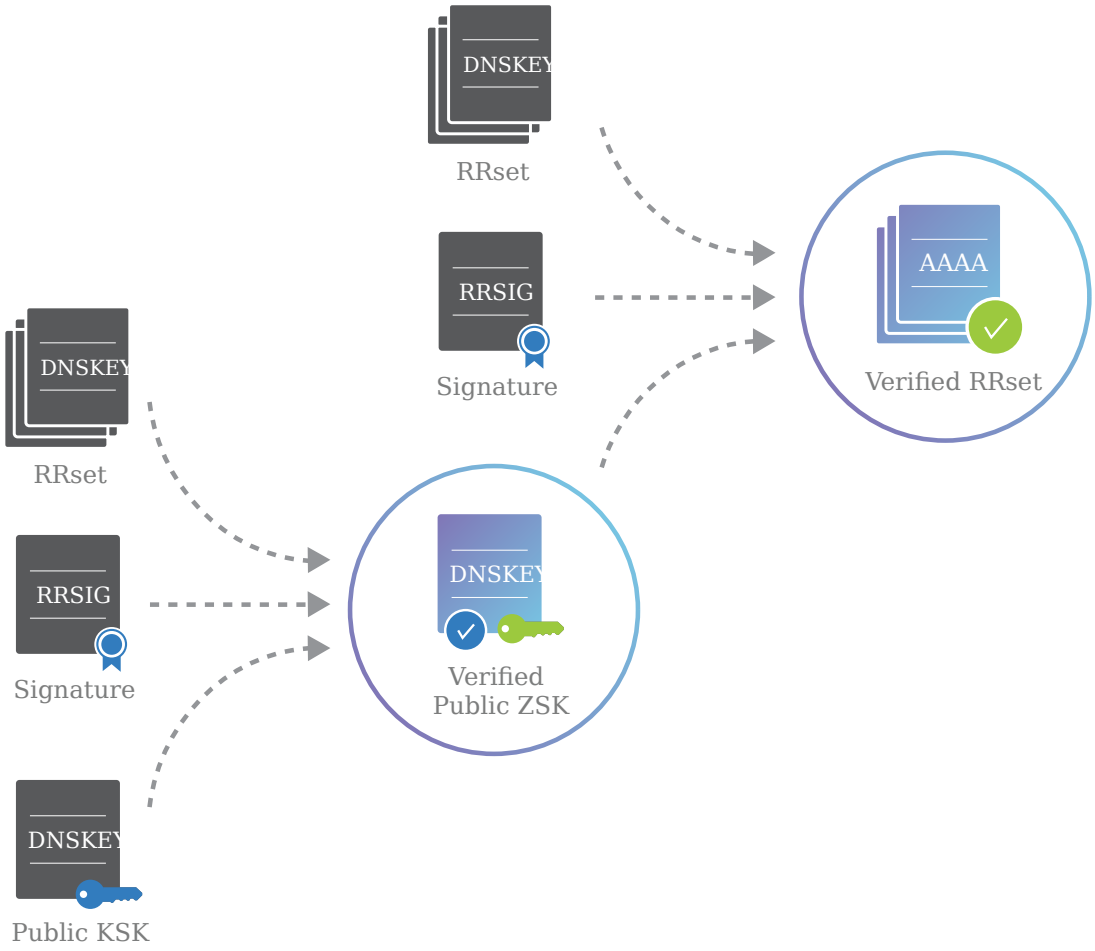
In addition to a zone-signing key, DNSSEC name servers also have a key-signing key (KSK). The KSK validates the DNSKEY record in exactly the same way as our ZSK secured the rest of our RRsets in the previous section: It signs the public ZSK (which is stored in a DNSKEY record), creating an RRSIG for the DNSKEY.



Just like the public ZSK, the name server publishes the public KSK in another DNSKEY record, which gives us the DNSKEY RRset shown above. Both the public KSK and public ZSK are signed by the private KSK. Resolvers can then use the public KSK to validate the public ZSK.

Validation for resolvers now looks like this:

- Request the desired RRset, which also returns the corresponding RRSIG record.
- Request the DNSKEY records containing the public ZSK and public KSK, which also returns the RRSIG for the DNSKEY RRset.
- Verify the RRSIG of the requested RRset with the public ZSK.
- Verify the RRSIG of the DNSKEY RRset with the public KSK.



Of course, the DNSKEY RRset and corresponding RRSIG records can be cached, so the DNS name servers aren't constantly being bombarded with unnecessary requests.

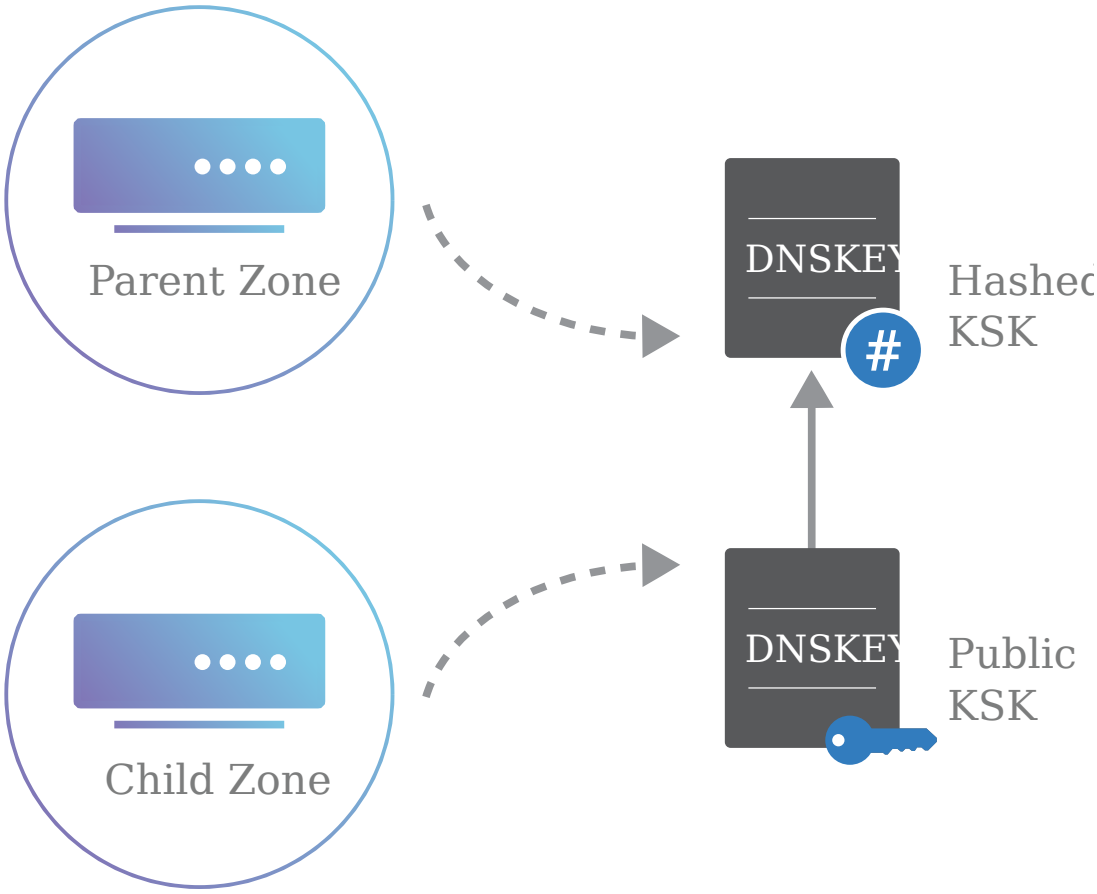
Why do we use separate zone-signing keys and key-signing keys? As we'll discuss in the next section, it's difficult to swap out an old or compromised KSK. Changing the ZSK, on the other hand, is much easier. This allows us to use a smaller ZSK without compromising the security of the server, minimizing the amount of data that

the server has to send with each response.

We've now established trust within our zone, but DNS is a hierarchical system, and zones rarely operate independently. Complicating things further, the key-signing key is signed by itself, which doesn't provide any additional trust. We need a way to connect the trust in our zone with its parent zone.

Delegation Signer Records

DNSSEC introduces a delegation signer (DS) record to allow the transfer of trust from a parent zone to a child zone. A zone operator hashes the DNSKEY record containing the public KSK and gives it to the parent zone to publish as a DS record.



Every time a resolver is referred to a child zone, the parent zone also provides a DS record. This DS record is how resolvers know that the child zone is DNSSEC-enabled. To check the validity of the child zone's public KSK, the resolver hashes it and compares it to the DS record from the parent. If they match, the resolver can assume that the public KSK hasn't been tampered with, which means it can trust all of the records in the child zone. This is how a chain of trust is established in DNSSEC.

Note that any change in the KSK also requires a change in the parent zone's DS

record. Changing the DS record is a multi-step process that can end up breaking the zone if it's performed incorrectly. First, the parent needs to add the new DS record, then they need to wait until the TTL for the original DS record to expire before removing it. This is why it's much easier to swap out zone-signing keys than key-signing keys.

Explicit Denial of Existence

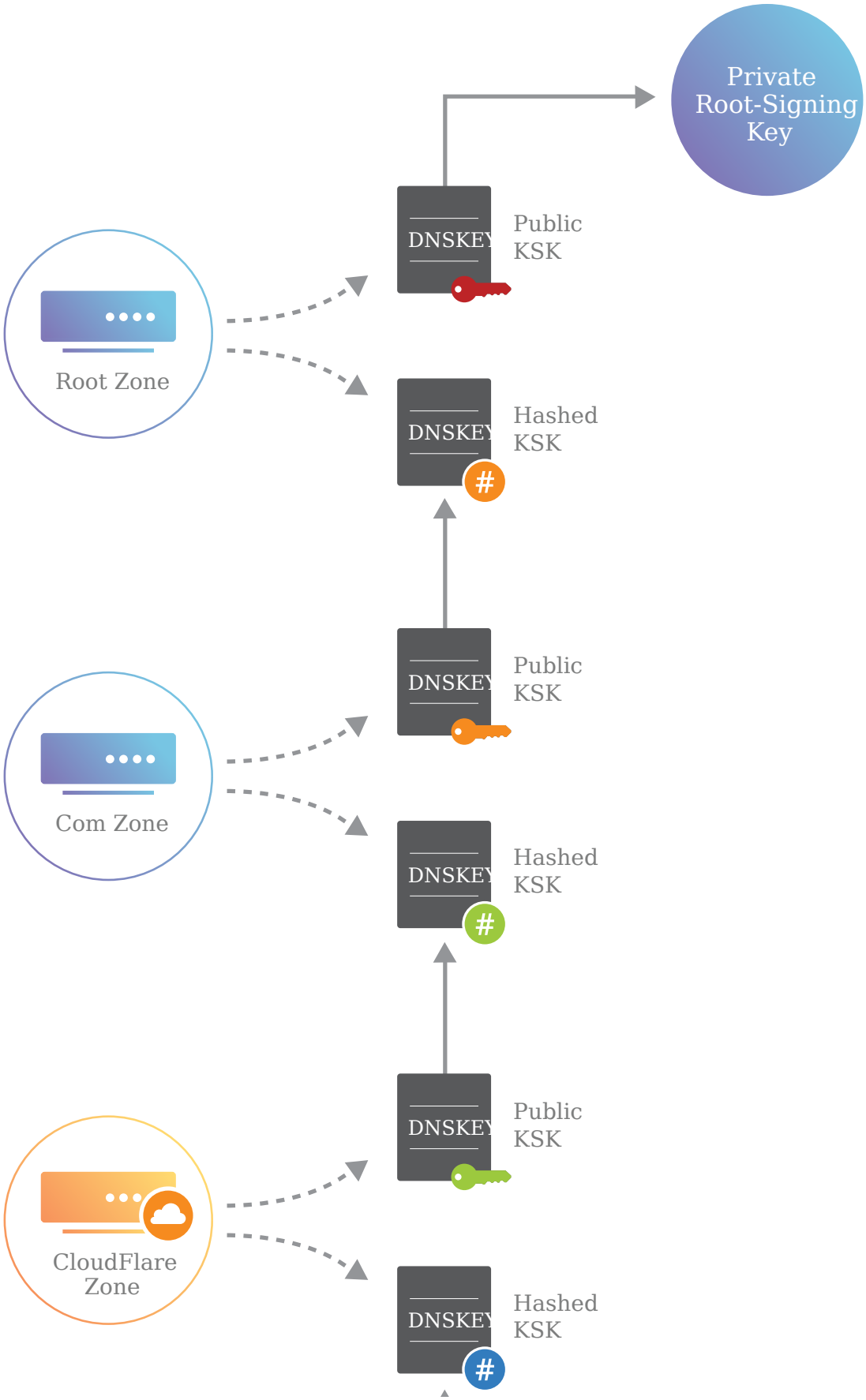
If you ask DNS for the IP address of a domain that doesn't exist, it returns an empty answer—there's no way to explicitly say, "sorry, the zone you requested doesn't exist." This is a problem if you want to authenticate the response, since there's no message to sign. DNSSEC fixes this by adding the NSEC and NSEC3 record types. They both allow for an authenticated denial of existence.

NSEC works by returning the "next secure" record. For example, consider a name server that defines AAAA records for api, blog, and www. If you request a record for store, it would return an NSEC record containing www, meaning there's no AAAA records between store and www when the records are sorted alphabetically. This effectively tells you that store doesn't exist. And, since the NSEC record is signed, you can validate its corresponding RRSIG just like any RRset.

Unfortunately, this solution allows anybody to walk through the zone and gather every single record without knowing which ones they're looking for. This can be a potential security threat if the zone administrator was counting on the contents of the zone being private. You can read more about this problem in DNSSEC: Complexities and Considerations, as well as Cloudflare's unique solution in DNSSEC Done Right.

The Chain of Trust

Ok, so we have a way to establish trust within a zone and connect it to its parent zone, but how do we trust the DS record? Well, the DS record is signed just like any other RRset, which means it has a corresponding RRSIG in the parent. The whole validation process repeats until we get to the parent's public KSK. To verify that, we need to go to that parent's DS record, and on and on we go up the chain of trust.


Private
Root-Signing
Key
DNSKEY
Public
KSK
Root Zone
DNSKEY
Hashed
KSK
#
DNSKEY
Public
KSK
Com Zone
DNSKEY
Hashed
KSK
#
DNSKEY
Public
KSK
CloudFlare
Zone
DNSKEY
Hashed
KSK
#

However, when we finally get to the root DNS zone, we have a problem: there's no parent DS record to validate against. This is where we get to see a very human side of the global Internet.

In the Root Signing Ceremony, several selected individuals from around the world come together and sign the root DNSKEY RRset in a very public and highly audited way. The ceremony produces an RRSIG record that can be used to verify the root name server's public KSK and ZSK. Instead of trusting the public KSK because of the parent's DS record, we *assume* that it's valid because we trust the security procedures around accessing the private KSK.

The ability to establish trust between parent and child zones is an integral part of DNSSEC. If any part of the chain is broken, we can't trust the records we're requesting because a man-in-the-middle could alter the records and direct us to any IP address they want.

Summary

Similar to HTTPS, DNSSEC adds a layer of security by enabling authenticated answers on top of an otherwise insecure protocol. Whereas HTTPS encrypts traffic so nobody on the wire can snoop on your Internet activities, DNSSEC merely signs responses so that forgeries are detectable. DNSSEC provides a solution to a real problem without the need to incorporate encryption.

Cloudflare's goal is to make it as easy as possible to enable DNSSEC. Right now, customers with Cloudflare paid plans can add DNSSEC to their web properties by flipping a switch to enable DNSSEC and uploading a DS record (which we'll generate automatically) to their registrar. Learn more about how to get DNSSEC.

We've also published an Internet Draft outlining an automated way for registries and registrars to upload DS records on behalf of our customers. This will enable Cloudflare to automatically enable DNSSEC for our entire community. Stay tuned for updates.