March 2020
Geoff Huston

Revocation

A Public Key Infrastructure (PKI) is a system designed to support the use of public/private keyed digital signatures through a system of structured transitive trust. The objective of a PKI is to enable trusted communications between parties who may have never met and may not necessarily even know each other at all. A PKI normally uses X.509 public key certificates, which are digital objects that contain a verifiable attestation that the certificate issuer has satisfied itself, via the application procedures documented in its Certificate Practice Statement, that the holder of a given public/private key pair has met certain criteria, as specified by the certificate issuer. The certificate issuer then publishes a certificate that associates a subject name (such as an individual's details for identity certificates or a DNS name for a domain name certificates) with the holder's public key, and then attaches a digital signature to this object, generated using the certificate issuer's private key. This act is both verifiable by any party that has knowledge of the issuer's public key and cannot be subsequently repudiated by the issuer.

These X.509 public key certificates are used for many purposes, as they support authenticity, verifiability, and attribution. For example, if an individual signs a digital document with their certified private key then anyone who refers to the associated public key identity certificate can validate (*verifiability*) that it was this particular individual who signed the document (*attribution*), and the document is unaltered (*authenticity*), as long as they are prepared to trust the integrity of the certificate issuance practices of the certificate issuer.

In the context of the Internet, we see increasing use of PKIs in the web, where web sites publish their content using Transport Layer Security (TLS) (as seen in the use of HTTPS URLs). The use of the web PKI allows a client to validate the authenticity of the remote server's identity, ensures that the transaction cannot be eavesdropped by third parties, that the contents of the transaction are not altered in any way, and that the server cannot repudiate the transaction. Obviously, this level of trust is vital for the Internet, and that implies that the foundation of this trust, the system of certification of domain name holders, is critically important. X.509 certificates come in many forms and their use in a particular context is typically defined within the parameters of a standard profile. A standard profile for X.509 certificates for use in the Internet is published in RFC 5280.

Trust is not eternal, and neither should certificates be regarded as eternal. An X.509 public key certificate has two date fields, `notBefore` and `notAfter`, specifying the time span when the certificate can be used. (Although it must be noted RFC 5280 does include the specification of a 'forever' `notAfter` date value if eternal trust really is the intended outcome!) The usual practice of certification management is to set the `notBefore` field to the date of certificate issuance and setting the `notAfter` field to some period specified by a contract or agreement between the certificate issuer and the subject. The subject is expected to apply for a new certificate before the expiration of the current certificate if they wish to continue to be certified beyond the `notAfter` date. Prior to Let's Encrypt's entry into the SSL certification market typical certification periods were one or two years. Let's Encrypt is now is a major player and their certificates have a 90-day validity period, so the average validity period for such certificates has come down.

There is always the case that the unexpected happens, and X.509 certificates are no exception. There are circumstances where the certificate should be marked as unusable before the `notAfter` expiration time. The private key may have been compromised, or the certificate was issued in error, or the subject is no longer undertaking the activity for which it was certified, or a myriad of other reasons.

How can a certificate be marked as unusable, (or be *revoked*)? The certificate issuer should remove the revoked certificate from its publication point, so that the revoked certificate is no longer available for upload and use by relying parties. But many relying parties gather and locally store copies of such certificates until their expiration date. How can these third parties be made aware that a certificate has been revoked and that it can no longer be used to support the establishment of a trust relationship? In practical terms, how can a browser be aware that a certificate being used to establish a TLS session has been revoked by the certificate issuer, and that the TLS session should not be opened?

Before looking the revocation mechanisms, I should note one rather esoteric point about revocation, or more precisely its reverse, *unrevocation*.

Revoking a certificate causes the Certification Authority (CA) to create a metadata record about the unusable status of the certificate but the CA does not issue an altered certificate that records its revoked status. A CA could, in theory, subsequently remove the metadata listing of the revoked status of the certificate, and because the certificate itself has not been altered in any way, it could restore the certificate back into its publication point. In effect, the published certificate state after this removal of the revocation metadata would the same as it was prior to the revocation action. The certificate has been unrevoked.

In practice this is a terrible idea and CA's should not attempt this. It is far more prudent to signal the re-instatement of trust by issuing a new certificate for the same subject with a new serial number. The new certificate may or may not reuse the public/private key pair, and that is a decision for the subject to make, not the CA.

The reasons why CA's should not attempt this *unrevocation* short cut is because the CA has no control of how relying parties handle revocation metadata. It is quite possible, and even quite reasonable, for a relying party to treat revocation as irreversible and remove this certificate from their local trust set for the remainder of the validity lifetime of this revoked certificate. In other words, in practical terms a relying party may interpret certificate revocation as irrevocable. This implies that there is nothing the CA can do to force such relying parties to reset their trust state relating to these revoked certificates, which implies that CAs are in effect forced to treat certificate revocation itself as an irrevocable action.

Certificate Revocation Lists (CRLs)

A conventional PKI response to manage revocation is for the Certification Authority (CA) to regularly publish a signed Certificate Revocation List (CRL). A CRL contains a list of the certificate serial numbers of all unexpired revoked certificates that have been issued by the same CA as the issuer of the CRL and the time of revocation. A CRL also contains the date of issuance of this CRL and the anticipated of the next CRL to be published by this issuer. CRLs are signed documents, signed with the private key of the CA. A standard profile for CRLs for use in the Internet is published in RFC 5280.

The CRL is intended to be complete, in that at any time within the date specified by the CRL, all unexpired revoked certificates issued by this CA in a given scope are listed in the CRL. If the scope of the CRL is the entire set of certificates issued by this CA, then the corollary is that if an unexpired certificate is not listed in the CRL, then it can be trusted until the next CRL issuance time.

Relying parties can consider a CRL itself to be valid if the current time is between the CRL time and the CRL's next update time, and the CRL's signature can be validated.

Having a certificate listed in a CRL effectively curtails a certificate's validity, but the action is not necessarily immediate across the broader domain of use. A relying party may hold a local copy of an issuer's CRL until the CRL's `nextUpdate` time, and therefore revocation will not necessarily be noted by relying parties until the next CRL is published.

The more the number of unexpired revoked certificates the larger the CRL will get. For a large CA the workload associated with CRLs can be significant. Delivering a complete list of all revoked certificates seems to be a case of over-answering, particularly if all the querier wanted to know was the revocation status of a single certificate. Also, the generation of CRLs is not a mandatory requirement for CAs. A CA may elect to regularly publish CRLs, or may elect to publish CRLs and update them with delta CRLs, or may not publish CRLs at all! For these reasons CRLs are typically not used by end clients when setting up a TLS session.

Online Certificate Status Protocol (OCSP)

OCSP is an alternative to the omnibus style of CRLs. In OCSP a client generates an OCSP request that contains a certificate serial number and sends it to the CA that issued the certificate. The CA responds with a signed certificate status report indicating whether the certificate is good, or whether the certificate has been revoked. The protocol is documented in RFC 6960.

An OCSP request is an ASN.1 object, containing one or more certificate serial numbers of certificates that the client wants the CA to check.

An OCSP response is digitally signed by the CA who issued the certificate, or a CA-designated responder. The response includes the identity of the responder, the time of the response, responses for each certificate in the request and optional extensions. The response codes used by OCSP indicate that the certificate is either:

- *good*, which actually means no certificate with this serial number issued by this CA is revoked. As RFC 6960 explains: "This state does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval."
- *revoked*, which indicates an unexpired revoked certificate, but may also indicate that this CA has not issued a certificate with this serial number.
- *unknown*, which indicates that the CA does not recognise this certificate serial number.

The extensions in the response may include a *nonce* that cryptographically links a request to the response, preventing replay attacks. It may also include a reference to a CRL. OCSP responses also may include four times:

- `thisUpdate` – the time that the responder generated this status information
- `nextUpdate` – the time by when updated information will be available
- `producedAt` – the time the responder signed this response
- `revocationTime` – the time when the certificate was revoked

These fields allow the client to cache an OCSP response, as the response can be cached until the `nextUpdate` time. OCSP responses can be generated in advance, with the time of the generation of the response indicated by the `producedAt` time.

There are some privacy concerns with OCSP in having the client contact the CA, in that the CA is then aware of the identity of clients using this certificate via the source of the OCSP request and also aware of the when the client is using the certificate. There are also performance issues with the additional time taken to generate the OCSP request and waiting for the response. This is not necessarily a single request, as a prudent client would check not only the revocation status of the certificate used by the server, but also check the revocation status of all the CA certificates used by the client to assemble the validation chain from a Trust Anchor to this certificate.

One response to these concerns related to on-demand certificate status checking is to package the OCSP response with the certificate, in a framework called *OCSP stapling* (described in RFC 6961). As the OCSP response is already signed and dated by the CA, and the server knows which certificate it has passed to the client, it also knows what OCSP requests the client will make to confirm that the certificate has not been revoked by the CA. The server uses OPCSP stapling to attach the OCSP response to the TLS material used in the handshake and may also attach the OCSP response of other CA certificates that will form the validation chain used by the client to validate the certificate.

Testing Certificate Revocation and Browsers

To experiment with the various ways in which browser applications handle revoked certificates I used Let's Encrypt to generate a certificate and then revoked it. I confirmed the revocation via an OCSP query:

```
$ openssl ocsp -issuer lets-encrypt-x3-cross-signed.pem.txt -serial
0x03DEAA6ADA0286BB5D733188CDB1EBF3cc9B -url http://ocsp.int-x3.letsencrypt.org  -text
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: 7EE66AE7729AB3FCF8A220646C16A12D6071085D
          Issuer Key Hash: A84A6A63047DDDBAE6D139B7A64565EFF3A8ECA1
          Serial Number: 03DEAA6ADA0286BB5D733188CDB1EBF3CC9B
    Request Extensions:
        OCSP Nonce:
            0410F996916B40E625BEAC68513525FA1593
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
    Responder Id: C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3
    Produced At: Mar 10 23:39:00 2020 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 7EE66AE7729AB3FCF8A220646C16A12D6071085D
      Issuer Key Hash: A84A6A63047DDDBAE6D139B7A64565EFF3A8ECA1
      Serial Number: 03DEAA6ADA0286BB5D733188CDB1EBF3CC9B
    Cert Status: revoked
    Revocation Time: Mar 10 23:39:49 2020 GMT
    This Update: Mar 10 23:00:00 2020 GMT
    Next Update: Mar 17 23:00:00 2020 GMT

    Signature Algorithm: sha256WithRSAEncryption
         73:88:aa:a1:1d:ff:5f:5b:eb:30:9d:43:ca:76:b8:3e:70:9f:
         d7:d2:3f:6e:dd:dd:fb:69:0f:16:e4:b3:3c:f3:75:d3:6b:37:
         f4:fa:cc:10:15:8c:cf:59:e0:f4:2a:60:d9:4c:5e:b2:df:24:
         d9:a1:20:b2:7e:14:d8:d0:03:92:97:9e:be:b3:e5:4e:c9:6c:
         db:96:8c:ff:6c:c7:4f:cf:88:35:bb:52:90:4f:6e:b9:51:70:
         f1:51:93:9d:de:b0:91:44:69:12:47:15:b2:18:c3:0d:bd:d5:
         af:01:ff:c3:8d:c0:31:94:87:e0:0e:06:18:35:7a:a8:4a:dd:
         2a:e4:61:2a:6d:db:6e:9f:87:d4:9f:79:25:17:f5:7a:e3:4d:
         7b:44:95:56:d4:ff:b2:38:50:f6:58:7c:d3:97:0c:e6:ab:2c:
         f6:2b:9a:55:a8:63:c6:f4:b9:97:2b:21:a2:bf:38:0d:91:e6:
         af:64:22:b8:50:b4:e8:70:27:ee:60:0d:fd:96:6e:b2:54:f8:
         38:ed:14:31:ca:1e:3f:c3:7f:ae:f5:d3:ff:9b:75:bf:4d:12:
         e7:1b:9a:62:a8:d9:c0:a4:8f:48:33:b2:f5:ea:d0:e9:27:e3:
         4f:ed:c3:1a:80:3a:1b:94:27:7c:90:56:c3:b3:65:7a:6e:5f:
         94:a9:56:79
WARNING: no nonce in response
Response verify OK
0x03DEAA6ADA0286BB5D733188CDB1EBF3cc9B: revoked
       This Update: Mar 10 23:00:00 2020 GMT
       Next Update: Mar 17 23:00:00 2020 GMT
       Revocation Time: Mar 10 23:39:49 2020 GMT
```

The Let's Encrypt server has an OCSP service that uses a 7-day OCSP response. A nonce was requested to protect the response against replay, but the Let's Encrypt OCSP server did not use a nonce in the response.

Various browsers and operating system platforms were tested for their behaviour when connecting to a site that uses this revoked certificate, shown in Table 1. The server was not configured to perform OCSP stapling, so it was left to the client browser to detect revocation using OCSP. In the table below **YES** indicates that the browser detected that the host certificate has been revoked, while **NO** indicates that the browser connected to the site and marked the connection as secure. The version numbers of both the platforms and the browsers used in this small experiment are also shown in this Table.

Platform	Chrome	Firefox	Opera	Safari	Edge
Mac OS X 10.15.3	**YES** 80.0.3987.132	**YES** 73.0.1	**YES** 67.0.3575.53	**YES** 13.0.5	
iOS 13.3.1	**YES** 80.0.3987.95	**YES** 23.0	**NO** 16.0.15	**YES** 13.3.1	
Android 10	**NO** 80.0.3987.132	**NO** 68.6.0	**NO** 56.1		
Windows 10	**NO** 80.0.3987.132	**YES** 74.0	**NO** 67		**YES** 44.18362

Table 1 – Browser Revocation Status

The diversity in behaviour observed here is in part to the distinction between platform services and the applications' choice of API options when interfacing with each platform's security services. These days the overwhelming majority of the Internet's user base uses the Chrome browser on an Android platform, so it's evident that the majority of the Internet's users don't perform revocation checks.

Is Revocation worth it?

This seems like an odd question. A compromised private key should not be accepted. An attacker might use this compromised private key to impersonate a site, and this vulnerability needs to be prevented. The way to stop a compromised key from being accepted is to disseminate the information that the key pair is now no longer usable, and this is achieved by revoking the public key certificate. Or so goes the conventional thinking on X.509 certificates and revocation.

But revocation is not always used or even useful. For example, DNSSEC, which uses public/private keys, has no revocation capability. The remedy for a compromised private key is to re-sign the zone with a fresh key and rely on the DNS TTL cache management data to flush out the old key values from the various DNS caches. DNS typically uses TTLs of hours of days, compared to the use of months, years and even multiple years in Web PKI certificates. That means that the window of vulnerability in DNSSEC from a compromised key is far shorter than that of the Web PKI, and perhaps this is the major reason why revocation is a far bigger issue in Web PKI certificates than it is in DNSSEC.

This means that there are cases in the use of public/private keys where the mechanisms of revocation are not considered to the necessary. What about a conventional use of certificates in the Web PKI? Is revocation useful? Or is it just another means of adding risk through additional points of potential vulnerability?

Both on-demand CRLs and OCSP require CAs' service points to be available. In the case of CRLs it is possible to rely on local caching of CRLs and to some limited extent alleviate this availability requirement, but for on-demand OCSP there is simply no way out. The OCSP server has to be available, and available at a speed commensurate with the tight time constraints of a TLS session setup in an application.

This is not always going to be achieved for OCSP so we need to consider the failure case. What should the client do if the OCSP request is not answered? Proceeding is a *soft-fail* that exposes the client to the potential perils that OCSP was intended to avoid. A cautious denial, or *hard-fail* may simply generate unnecessary blocking. The OCSP service point becomes a single point of potential failure and in a world of various forms of constrained and unreliable access adding one more point of connection failure is just adding to the burden. A *hard-fail* framework also runs the risk of making these OCSP servers yet another point of vulnerability in a hostile DOS scenario. It appears that many client applications and operating system service libraries use *soft-fail* if an OCSP query elicits no response. This has consequences, as an attacker who is on the path between the user and the CA may see the unencrypted OCSP query and simply block it. The client's certificate validation function will then *soft-fail* and regard the revoked certificate as valid. The client is then placed into the vulnerable position of trusting a revoked certificate.

Can we make OCSP more robust and counter these OCSP stripping attacks? If OCSP query or response interception is the problem, perhaps mandatory OCSP stapling might be better approach. This is documented in a now long-expired internet draft from 2013 (draft-hallambaker-muststaple-00.txt), where the server that serves the certificate during TLS setup is also required to staple current OCSP information by virtue of an extension in the certificate. Given that this *must stable* flag sits inside the signed certificate an attacker cannot strip the flag out during the TLS setup exchange. Here the CA (or possibly the original subject) is mandating that all servers that use this certificate must also use OCSP stapling. This approach addresses the threats of OCSP stripping and privacy compromise while also avoiding additional delays in the TLS handshake to perform the revocation check as a distinct exchange between the client and the CA OCSP server. The outcome is a reduction in the window of vulnerability in a compromised certificate from the validity lifetime of the certificate itself (commonly a number of years) to the validity lifetime of the OCSP response (commonly a number of days).

There are other issues with revocation, in that revocation will not stop attacks that exploit compromised keys. An attacker can use the compromised private key to generate new credentials that would permit the attacker to re-certify the compromised service with the attacker's keys. By the time the original key compromise is detected the problem now is that the attacker is using different keys and a different certificate, and the onus is now placed on the original service owner to convince the replacement-issuer CA that the replacement certificate was incorrectly issued and get that certificate revoked. An agile attacker could repeat this re-certification process a number of times, further frustrating the efforts to remove these fraudulently obtained certificates from the PKI.

Maybe the problem with revocation is best solved by avoiding long-lived certificates in the first place. As Google's Adam Langley pointed out some nine years ago:

```
A much better solution would be for certificates to only be valid for a few days and to
forget about revocation altogether. This doesn't mean that the private key needs to change
every few days, just the certificate. And the certificate is public data, so servers could
just download their refreshed certificate over HTTP periodically and automatically (like
OCSP stapling). Clients wouldn't have to perform revocation checks (which are very complex
and slow), CAs wouldn't have to pay for massive, DDoS proof serving capacity and revocation
would actually work. If the CA went down for six hours, nobody cares. Only if the CA is down
for days is there a problem. If you want to "revoke" a certificate, just stop renewing it.
```

https://www.imperialviolet.org/2011/03/18/revocation.html

Are there other approaches to certificate revocation?

As always, whatever the problem might be, the answer is "just use the DNS," and OCSP is no exception. A recent internet draft (from 2017) proposes OCSP as a DNS resource record type (draft-pala-odin-02.txt). The OCSP query is encoded into the DNS query name, combining the query and response. For example, a query for the OCSP RR type with a query name of `123456.ca1.example.com` would respond with the OCSP status of the certificate with serial number 123456 issued by the CA with the publication point name of `ca1.example.com`. Presumably, a prudent implementation would also require this DNS zone to be DNSSEC-signed, although as the OCSP record it itself signed by the CA the major purpose of the DNSSEC signature appears to be assured presence and currency in this case.

This approach invokes the DNS to mitigate the vulnerability of the CA's OCSP service point, allowing DNS recursive resolvers to cache this data. Where the query is widely used, resolver-based DNS caching can provide exceptional improvements in query speed, even with DNSSEC validation. However, when the data is not present in the local DNS resolver's cache the process can be far slower than a conventional OCP query and response.

Mozilla has revived examination of CRL based approaches, addressing the issue of the size of the CRL. Their approach is to use Bloom Filters to compress the effective size of the CRL, using a technique they call CRLite (https://blog.mozilla.org/security/2020/01/09/crlite-part-2-end-to-end-design/). With this approach they have shown a reduction on CRL data from a list of all enrolled and unexpired certificate serial numbers from 6.7G to a filter of just 1.3M. Their approach is to use this technique on the larger CAs and fall back to OCSP where CRL data has not been set up as a filter. The approach attempts to strip out additional delays in on-demand OCSP and not rely on the piecemeal implementation of *must staple* server-side OCSP support.

Let's remind ourselves of where we are here.

CA's issue long-lived certificates and that means that when a key pair is compromised, this vulnerability may persist for years. We'd like to annul the validity of the certificate in a faster timeframe. Certificate Revocation Lists provide such a service, but they can become large and pose issues in delivering this high volume of data on a just-in-case basis to clients. We turned to use OCSP, so we could query the revocation status of a single certificate but because of doubts as to the resilience of OCSP the client systems turned to a *fail safe* approach which allows invalid certificates to continue to be trusted.

Trust

There is no panacea here, and every approach to certificate revocation represents some level of compromise.

We can have long-lived certificates with high enrolment costs but only if we can support a robust and fast revocation mechanisms, which to date has been an elusive goal. CRLs and OCSP are not instant responses but they can drastically reduce the timeframe of vulnerability arising from a compromised private key, even though there are some clear issues with robustness with both CRLs and various flavours of OCSP.

We can head down the path forged by Let's Encrypt and only use short lived certificates generated by highly automated processes. Given their short lifetime revocation is not a major issue, and it's possible to contemplate even shorter certificate lifetimes. If revocation lists are refreshed at one-week intervals, then a one-week certificate could simply be allowed to expire as revocation would not substantially alter the situation for relying parties.

But if we head down this path of short-lived certificates, then why do we need to bother with the X.509 wrapper at all? Why not just use DANE and store the keys into the DNS and rely on DNSSEC to provide the necessary authenticity and use DNS TTL controls to control cached lifetime of the public key? With DNSSEC Chain Extensions it is possible to perform a security association handshake by having the server provide the client with the server's public key response, a DNSSEC signature and the associated DNSSEC validation responses without performing any query in the DNS at all. So, just like OCSP stapling, it's possible to use DANE and DNSSEC Chain stapling and avoid the DNS query delay completely.

This story is by no means over. We are seeing faster attacks that perform the entire process of infiltration, deception and data exfiltration in just a few hours rather than days or weeks. The responses we rely on today, including certificate transparency logs and the piecemeal use of OCSP, introduce time lags in the currency of credential data of a scale of days rather than seconds.

The situation points to the uncomfortable conclusion that as far as the security of the Internet is concerned, we are placing undue reliance on a security framework that at best offers same week service in a nanosecond world.

References and Further Reading

RFC 5280, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", D. Cooper et.al, May 2008.
https://tools.ietf.org/html/rfc5280

RFC 6960, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP", S. Santesson, et.al., June 2013.
https://tools.ietf.org/html/rfc6960

RFC6961, "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", Y. Pettersen, June 2013.
https://tools.ietf.org/html/rfc6961

"X.509v3 Extension: OCSP Stapling Required", Internet Draft, P. Hallam-Baker, April 2013
https://tools.ietf.org/id/draft-hallambaker-muststaple-00.txt

"OCSP over DNS (ODIN)", Internet Draft, M. Pala, November 2017.
https://tools.ietf.org/id/draft-pala-odin-02.txt

"Revocation Doesn't Work", Adam Langley, March 2011
https://www.imperialviolet.org/2011/03/18/revocation.html

"No, don't enable revocation checking", Adam Langley, April 2014
https://www.imperialviolet.org/2014/04/19/revchecking.html

"OCSP Stapling: How CloudFlare Just Made SSL 30% Faster", Matthew Prince, October 2012
https://blog.cloudflare.com/ocsp-stapling-how-cloudflare-just-made-ssl-30/

"High-reliability OCSP stapling and why it matters", Nick Sullivan, July 2017
https://blog.cloudflare.com/high-reliability-ocsp-stapling/

"Revocation is Broken", Scott Helme, July 2017
https://scotthelme.co.uk/revocation-is-broken/

"The Problem with OCSP Stapling and Must Staple and why Certificate Revocation is still broken", Hanno Böck, May 2017
https://blog.hboeck.de/archives/886-The-Problem-with-OCSP-Stapling-and-Must-Staple-and-why-Certificate-Revocation-is-still-broken.html

Disclaimer

Author

Geoff Huston AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net