The use of Elliptic Curve Cryptography in DNSSEC

Francis Dupont
Internet Systems Consortium
fdupont@isc.org

3 may 2013

Abstract

The RFC 6605 introduced the modern cryptography based on elliptic curves into DNSSEC. This document explains what are the advantages, and the few disadvantages, to switch from current DSA/RSA keys and signatures to Elliptic Curve Cryptography.

Modular Group Cryptography

Some Maths

All objects here are based on an abstract algebraic structure named a *field* in English, un *corps* in French, ein *Körper* in German. In fact definitions are different but give in the finite case the same objects with p^k elements, p being a prime, noted $\mathbb{F}_{p^k}$ or $GF(p^k)$, with the common addition and multiplication modulo p^k.

In the finite field $\mathbb{F}_p$, the elements at the exception of 0 form a group for the multiplication, so for any a the set $\{1, a, a^2, \ldots\}$ is a cyclic subgroup with a number of elements which divides (by Lagrange's theorem) the number of elements of the whole group, i.e., $p-1$, so for instance $a^{p-1} \equiv 1 \pmod{p}$ when p is a prime (as known as the Fermat's little theorem).

Diffie-Hellman example

The Diffie-Hellman protocol is the simplest application of these maths to cryptography. Take a big prime p so $\frac{p-1}{2}$ is also prime, and a base g so the

group of the powers of g has $p-1$ different elements (with a simple extra condition on p, 2 (or 3 or 5) can be chosen for g).

To get x from g^x modulo p (aka the discrete logarithm) is a hard problem for large values of p. The idea is to apply the identity $(g^x)^y \equiv (g^y)^x \equiv g^{xy}$ to build a shared secret g^{xy} from private x and y, and public g^x and g^y.

Signing

The standard signing protocol over modular groups is DSA [1], an ElGamal signature scheme.

Note if RSA uses modular arithmetic, it is not based on modular groups. In fact its hard problem is the factorization of the product of two large primes, not the discrete logarithm.

Elliptic Curve Cryptography

The discrete logarithm problem on not special elliptic curves was considered for cryptography because it has no better solution than the generic (a.k.a. brute force) one.

An elliptic curve over $\mathbb{F}_p$ (EC cryptography uses too curves over $\mathbb{F}_{2^m}$ but not in the DNSSEC context) are points of coordinates x and y (in $\mathbb{F}_p$) verifying an equation like $y^2 \equiv x^3 + ax + b[p]$ (note the cube makes this equation not defining an ellipse) with a point at infinity (so the term *projective* for the plane).

An addition is defined on the curve with the point at infinity as the identity element, given a group with a number of elements (named the *order* of the group) close to the prime p (by the Hasse's theorem), and the curve can be chosen to get this number a prime (case of DNSSEC curves) or a small multiple of a prime [2].

With a base point G, all the cryptographic protocols defined on the $g^n[p]$ modular group can be translated to the $n * G$ EC group (so we have ECDH and ECDSA, but no ECRSA).

Operations over an EC group are faster than over the equivalent (in term of strength of the hard problem to solve to break the crypto) modular group, not because the power and multiply are replaced by multiply and add (these are only notations) but because the primes are far smaller (256 bits vs. 2048 or 3072 bits) and can be chosen to make modular arithmetics far simpler without an adverse impact on the discrete logarithm problem hardness.

Application to DNSSEC

The RFC 6605 defines the use of ECDSA for signing, i.e., in DNSKEY and RRSIG resource records. The used elliptic curves are the P-256 and P-384 NIST pseudo-random curves defined on $\mathbb{F}_p$ fields with p 256 and 384 bit carefully chosen primes, and base points with 256 and 384 bit prime orders. The hash algorithms are SHA-256 and SHA-384.

Public keys Q are uncompressed curve points, i.e., 512 or 768 bit strings formed by the concatenation of the two point coordinates $x \mid y$. Private keys d are 256 or 384 bit numbers between 1 and $order - 1$, with $Q = d * G$.

A signature is a pair of numbers modulo the order and must be encoded on 32 or 48 octets.

With PKCS#11 parameters are a bit more specific:

- mechanisms must have the *CKF_EC_F_P* ($\mathbb{F}_p$ support), *CKF_EC_NAMEDCURVE* (ASN.1 Object ID for the parameters, i.e., the curve) and *CKF_EC_UNCOMPRESS* (uncompressed, i.e., two coordinates, form of EC points) flags. Note they seem to be the common minimal choice.

- parameter attribute is filled with the Object ID of the curve (prime256v1 1.2.840.10045.3.1.7 or secp384r1 1.3.132.0.34) in ASN.1 DER encoding of a choice.

- point attribute is filled with the ANSI X9.62 DER encoding in the uncompressed form, i.e., the Octet String tag, the length of the string, the UNCOMPRESSED tag (4) and the two coordinates on 256 or 384 bits each.

Of course the device must support the *CKM_EC_KEY_PAIR_GEN* mechanism with the *CKF_GENERATE_KEY_PAIR* flag and the *CKM_ECDSA* mechanism with *CKF_SIGN* and *CKF_VERIFY* flags. Note there is no way to check in the PKCS#11 API the support of a specific curve but any HSM designed for the US or the international market is supposed to support the two very standard curves selected for DNSSEC.

Advantages and disadvantages

First in DNSSEC the ECDSA parameters are one of two curves, so there is no parameter generation consideration. This lack of freedom has a questionable impact on security. In our opinion, it makes things being simpler and removes

some occasions to do something the wrong way, and for sure in all cases in an expensive way.

As explained before, the numbers used in ECDSA are smaller than for RSA or DSA, so the operations are really faster. RSA can be optimized, usually RSA signing uses the Chinese Remainder Theorem (the dP, dQ and $qInv$ values you can find in the private key after the p and q secret primes), this leads to near 4 time better performance, for instance:

- 30500 1024 bit private RSA's in 10s (with CRT)
- 9341 1024 bit private RSA's in 10s (without CRT)

EC cryptography gets similar optimizations as shown by looking in OpenSSL 1.0.1e crypto/ec directory with for instance the `ecp_nistp256.c` 50kb file. By the way there is no equivalent file for the 384 bit curve? To finish with OpenSSL which if it is not the best written but is usually the fastest implementation, the magic configuration flag is *enable-ec_nistp_64_gcc_128* but it does not seem to be critical, i.e., it does not make a huge difference for signing.

If we compare ECDSA to RSA and DSA, exactly 256 bit ECDSA with 2048 bit RSA (note it should be in fact 3072 bit RSA) and 2048 bit DSA:

- 62991 256 bit ECDSA signs in 10s
- 25294 256 bit ECDSA verify in 10s
- 4778 2048 bit private RSA's in 10s
- 153062 2048 bit public RSA's in 10s
- 15233 2048 bit DSA signs in 10s
- 12622 2048 bit DSA verify in 10s

So ECDSA is 4 time faster than the equivalent DSA and here 12 time faster than RSA, so one can easily argue ECDSA signing is 20 time faster than the equivalent RSA one.

On HSMs the difference is smaller (perhaps ECC implementations are less mature on HSMs?) but still with an advantage for ECDSA, For instance the SafeNet Luna 5.0 PCI-E board can perform 1200 2048 bit RSA vs 1800 256 bit ECDSA signatures per seconds according to its product brief [4].

About the signature size, both DSA and ECDSA give a signature size 4 time the "symmetric key strength" and clearly smaller than RSA even this

advantage is not so great, i.e., for resource record reduced sizes one wins more with DNSKEYs.

ECDSA inherits from DSA its two main disadvantages. First the verification is slower than the signing, second the signing procedure requires a random number:

- you must ask a cryptographer to qualify exactly the randomization requirement
- anyway a standard ([5], [6]) pseudo-random number generator is enough to fulfill the requirement
- to reuse the same value is trivially broken, even with ECDSA (it is alleged this mistake was the base of the PS3 secret key recovery)
- in conclusion the evaluation of an ECDSA device should be done on the secret generation and the signing procedure together

DNSSEC ECDSA in practice

Recent bind 9, nsd, oubound, etc, support *ECDSAP256SHA256* and *ECDSAP384SHA384* DNSKEY and RRSIG resource records. Associated tools, for instance `dnssec-keygen` and `dnssec-signzone`, too. But the situation is not so good for the perl package `Net::DNS::SEC` which has no ECDSA nor GOST support.

According to SecSpider [7] DNSSEC deployment statistics there is currently no ECDSA keys in the global DNS, but a few (23) GOST [8] keys which as the official name *ECC-GOST* suggests are based on Elliptic Curve Cryptography too.

Today we have no list of registries accepting ECDSA KSKs or associated DSs but up-to-date registry software, for instance EPP servers, should support any registered so legal algorithm value. For the possible check of KSKs, the answer is in the DNSSEC support library, i.e., it will work for bind 9 or unbound (ldns) but not for perl.

Other uses of ECC and Conclusion

As noted before, ECDSA is not the first DNSSEC signature scheme based on Elliptic Curve Cryptography. GOST (full name *ECC-GOST*) uses an elliptic curve on a 256 bit prime field and the signing process is very closed to the

ECDSA one. In fact the main technical difference between *ECC-GOST* and *ECDSAP256SHA256* is in hash algorithms (GOST R 34.11-94 vs. SHA-256).

It can be expected most national cryptography agencies will follow the Russian example and both will base the locally enforced signature on Elliptic Curve Cryptography, and try to get an algorithm code point at IANA. It is already the plan for China...

I proposed to modernize the TKEY mechanism with modern cryptography too, so to replace DH by ECDH, SHA-1 by SHA-256, etc.

To finish there are a clear tendency from national security regulators to push ECDSA in place of RSA. I have seen two medium term more than convincing arguments:

- to make minimal RSA modulus size bigger and bigger, so making ECDSA very attractive and RSA less and less praticable
- to disallow the PKCS#1 v1.5 signature mechanism in favor of better schemes (e.g., PSS).

So adapting a French expression, the years of RSA are numbered. And its successor is clearly ECDSA!

References

[1] NIST, FIPS PUB 186-3 *Digital Signature Standard (DSS)*, `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`, NIST, June 2009

[2] NIST, *Recommended Elliptic Curves for Federal Government Use*, `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`, NIST, July 1999

[3] P. Hoffman, W.C.A. Wijngaards, *Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC*, RFC 6605, IETF, April 2012

[4] SafeNet, *Luna PCI-E 5.0 Hardware Security Module (HSM) Product Brief*, `http://www.safenet-inc.com/WorkArea/DownloadAsset.aspx?id=8589946455`, SafeNet Inc., 2011

[5] NIST, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, `http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf`, NIST, January 2012

[6] D. Eastlake, J. Schiller, S. Crocker, *Randomness Requirements for Security*, RFC 4086, BCP 106, IETF, June 2005

[7] UCLA, SecSpider *Global DNSSEC deployment tracking*, *Deployment Stats*, `http://secspider.cs.ucla.edu/stats.html`

[8] V. Dolmatov, A. Chuprina, I. Ustinov, *Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC*, RFC 5933, IETF, July 2010

DSA and ECDSA signing algorithms in details

The DSA signing procedure is from the hash $H(m)$ of message m with the parameters prime p, subprime q and base g, and secret key x (public key is parameters and $y \equiv g^x \pmod p$):

0. take a new k at random with $0 < k < q$

1. calculate $r \equiv (g^k \pmod p) \pmod q$

2. in the unlikely case $r = 0$ retry in **0.** with another k

3. calculate $s \equiv k^{-1}(H(m) + xr) \pmod q$

4. in the unlikely case $s = 0$ retry in **0.** with another k

5. the signature is the pair (r, s)

The ECDSA signing procedure is from the hash $H(m)$ of message m with the parameters curve, point G of order n, and secret key d_A (public key is parameters and $Q_A = d_A * G$:

0. take a new k at random with $0 < k < n$

1. calculate the curve point $(x_1, y_1) = k * G$

2. $r \equiv x_1 \pmod{n}$

3. in the unlikely case $r = 0$ retry in **0.** with another k

4. calculate $s \equiv k^{-1}(H(m) + rd_A) \pmod{n}$

5. in the unlikely case $s = 0$ retry in **0.** with another k

6. the signature is the pair (r, s)