An implementation of the Boneh-Lynn-Shacham Short signature scheme

Michael Scott

School of Computing
Dublin City University
Ballymun, Dublin 9, Ireland.
mike@computing.dcu.ie

Abstract. Herein we briefly document our implementation of the Boneh-Lynn-Shacham [?] short signature scheme. The implementation uses the MIRACL library.

1 Introduction

The Boneh-Lynn-Shacham short signature scheme, as originally described [?], uses the Weil pairing to instantiate a scheme which, for the approximate RSA-1024-bit level of security, generates a signature of size of about 160 bits. This is half the size of signatures based on standard methods such as ECDSA. The security of the scheme is based on the Computational Diffie-Hellman problem over certain pairing-friendly elliptic curves, in the random oracle model. Note that this is a classic digital signature – it is not an identity-based signature. It does not support message recovery.

The signature itself is the x coordinate of an elliptic curve point over the prime field $\mathbb{F}_p$ with $p \approx 160$ bits. To obtain the required level of security we assume the use of a pairing-friendly curve with an embedding degree of $k = 6$, which acts as a "security multiplier", and raises the index calculus strength of the scheme to 960 bits. Since the security of the method depends on the difficulty of the discrete logarithm problem over a sextic extension field of this size, we are assuming that this gives approximately the same level of security as 1024-bit RSA.

We choose to implement the method using the faster Tate pairing over non-supersingular MNT curves [?], [?]. For our particular chosen curve p is actually 159 bits in length and the group size is 158 bits. In one small variation from the method as described in [?], we fix the actual signature point unambiguously by storing in the signature an extra bit, so that the y coordinate of the point can be reconstructed (as this enables a certain optimization – see below). Therefore our signature size is exactly 160-bits in length.

When using non-supersingular curves, while one parameter to the pairing may be placed on the elliptic curve over the base field $E(\mathbb{F}_p)$, the other must be placed on a larger extension field. For an MNT curve the best we can do is to place it on the quadratic twist $E'(\mathbb{F}_{p^3})$. Since the signature (to be short) must

be on $E(\mathbb{F}_p)$, this implies that the public key must be over the "bigger" curve. So the public key will be relatively large. On the other hand the secret key will be just 158 bits.

We note that potentially faster methods have also been published, for example see [**?**], based on different (and arguably less compelling) security assumptions. As described these require the use of the modified pairing over a supersingular curve, which for an embedding degree of 6 requires a fast implementation of characteristic 3 arithmetic, which we do not have to hand.

Note that the timings given in the original paper are not representative of the potential of the method. We are aware of implementations (using a characteristic 3 supersingular curve) which are much faster than those reported in the paper.

2 The implementation

The fixed domain parameters consist of the pairing friendly MNT curve with an embedding degree of 6, and a fixed point P on the twisted curve $E'(\mathbb{F}_{p^3})$. The private signing key is a 20 byte randomly generated number s. The public key is the point $R = sP$. R will be 120 bytes in size, although using point compression this could be reduced to 60 bytes.

To sign a message it must first be hashed to a point Q on the curve $E(\mathbb{F}_p)$. Basically the message is hashed to an x coordinate, and if the RHS of the elliptic curve equation is a quadratic residue (tested by calculating a jacobi symbol), then a modular square root determines the y coordinate. If the RHS of the elliptic curve equation is not a quadratic residue, the hash value is incremented until it becomes one. The total cost will thus be one modular square root and one or more jacobi symbol calculations. The signature is then the point $S = xQ$, compressed using standard methods to its x coordinate and a single bit, to a size of 20 bytes.

To verify the message we reconstruct S from its compressed form, hash the message again to the point Q and test that $e(Q, R).e(-S, P) = 1$. This should be the case as

$$e(Q, R).e(-S, P) = e(Q, xP).e(-xQ, P) = e(Q, xP)/e(Q, xP) = 1$$

Doing it this way the tricks for calculating a product of pairings [**?**] can be used, with some savings.