Abe-Okamoto signature scheme

EPK uses Abe-Okamoto signature scheme with message recovery to validate the product key and embed information, such as expiration date and type of license.

Abe-Okamoto signature scheme with n bits of security and n bits payload, produces a $4n$ bits signature. The signature scheme works with any cyclic group. EPK uses the 128-bit elliptic curve secp128r1, and bcrypt as random oracle.

Abe-Okamoto signature scheme was constructed by Masayuki Abe and Tatsuaki Okamoto, as presented in the paper A Signature Scheme as Secure as Discrete Logarithm.

The signature scheme is provably secure in the random oracle model, against an adaptive chosen-message attack, under the (elliptic curve) discrete logarithm assumption.

The proposed scheme is outlined below, following the same notation as used in the original paper. Given a group formed by the elements of an elliptic curve, g denotes a base point on this curve with order p.

Let F_1, F_2, F_3 and H be random oracle functions, in our case bcrypt with the salt `0xc90fdaa22168c234c4c6628b80dc1cd1` and a work factor of 6, except F_2 which has a work factor of 12 (due to security reasons mentioned below). F_1, F_2 and F_3 should output a hash truncated to 64 bits. H should output a hash truncated to 128 bits.

The signer picks a random 128-bit number x as private key and calculates the public key $Y = -x \times g$.

To sign a message m, the signer does the following:

1. $m' = F_1(m) || (F_2(F_1(m)) \oplus m)$
2. Pick a random 128-bit number ω.
3. $r = (\omega \times g)_x \oplus m'$
4. $c = H(r)$
5. $z = \omega + cx \bmod p$

Let t be the SHA2 hash of the tag truncated to 128 bits or 0 if no tag was supplied. The signature is the tuple $(r \oplus t, z)$. Since both r and z are 128-bit numbers, the size of the signature is 256 bits.

To verify the signature (r_t, z) and recover the message m, the verifier does the following:

1. Calculate the SHA2 hash t of the tag truncated to 128 bits or 0 if no tag was supplied. Restore $r = t \oplus r_t$. Let $c = H(r)$
2. $m' = r \oplus (z \times g + c \times Y)_x$
3. Recover $m = [m']_{k_2} \oplus F_2([m']^{k_1})$.
4. The signature is valid if $[m']^{k_1} = F_1(m)$ holds.

Using a 128-bit elliptic curve for signing, gives us 64 bits of security, which would make the implementation vulnerable to brute-force attacks. An adversary can always pick a signature (r, z) at random, and repeat until a valid signature is found, which on average would require 2^{63} tries. For each such signature, a valid product key could easily be constructed by encoding the bits of the signature. To prevent such an attack, we choose a high work factor for F_2, thus making signature verification slow. Unfortunately, this also make product key generation slower.