MSKey Readme

Abstract

Microsoft Windows Server 2003 VLK requires a VLK key to install. Commonly, for illegal users, this key is a leaked key, and thousands of piracy users use the same key to install their Windows. The problem is that the piracy users can use the product now, but not forever, because Microsoft would probably include the leaked key list in the further service packs (e.g. Microsoft prohibited several Windows XP VLK keys in service pack 1). So, it is necessary to install Windows with different keys for different illegal users.

By tracing Windows product key verification program, I successfully extracted the algorithm MS uses (some Public Key Infrastructure), and broke the private key uses to generate product keys.

Validation Process

Decode

The following computations are based on this product key:
JCF8T-2MG8G-Q6BBK-MQKGT-X3GBB
The character "-" does not contain any information, so, the MS product key is composed of 25-digit-character. Microsoft only uses "BCDFGHJKMPQRTVWXY2346789" to encode product key, in order to avoid ambiguous characters (e.g. "I" and "1", "0" and "O"). The quantity of information that a product key contain is at most $\log_2 24^{25} \approx 114 bits$. To convert a 25-digit key to binary data, we need to

a. convert "**JCF8T2MG8GQ6BBKMQKGTX3GBB**" to "6 1 3 22 ......", where 'B'=0, 'C'=1, 'D'=2 … we call the array "6 1 3 22…" *base24[]*

b. compute *decoded* = $\sum_{i=0}^{24} 24^{24-i} \times base24[i]$, the result is: **00 C5 31 77 E8 4D BE 73 2C 55 47 35 BD 8D 01 00** (little-endian) *c.* The decoded result can be divided into 12bit + 31bit + 62bit + 9bit, and we call theses 4 parts 12bit: *OS Family*, 31bit: *Hash*, 62bit: *Signature*, and 9bit: *Prefix*.

Verify

If you want to understand what I am talking about in this section, please refer to some Elliptic Curve Cryptography materials.
Before verifying a product key, we need to compute the 4 parts mentioned above: *OS Family*, *Hash*, *Signature*, and *Prefix*.

Microsoft Product-key Identification program uses a public key stored in PIDGEN.DLL's BINK resource, which is an Elliptic Curve Cryptography public key, which is composed of:
p, *a*, *b* construct an elliptic curve $y^2 = x^3 + ax + b \pmod{p}$

G(x,y) represents a point on the curve, and this point is so called "generator"
K(x,y) represents a point on the curve, and this point is the product of integer *k* and the generator *G*.

Without knowing the private key *k*, we cannot produce a valid key, but we can validate a key using public key:{*p*, *a*, *b*, *G*, *K*}

1. compute *H*=SHA-1(5D *OS Family,Hash*, *prefix*, 00 00) the total length is 11 byte. H is 160-bit long, and we only need the first 2 words. Right lift H's second word by 2 bits. E.g. if SHA-1() returns FE DC BA 98 76 54 32 10, H= FE DC BA 98 1D 95 0C 04.
2. compute *R(rx,ry)*= *Signature* * (*Signature***G* + *H***K*) (mod p)
3. compute SHA-1(79 *OS Family*, *rx*, *ry*) the total input length = 1+2+64*2=131 bytes. And compare *Hash* and result, and if identical, the key is valid.

Producing A Valid Key!

We assume the private key *k* is known (sure, Microsoft won't public this value, so we have to break it by ourselves).
The equation in the product key validation system is as below:

Hash=SHA(*Signature**(*Signature**G+SHA(*Hash*)**K*) (mod *p*))

What we need is to calculate a *Signature* which satisfies the above equation.

1. Randomly choose an integer *r*, and compute *R(rx,ry)*=*r* * *G*
2. Compute *Hash*= SHA-1(79 *OS Family*, *rx*, *ry*) the total input length = 1+2+64*2=131 bytes, and we get the first 62bit result.
3. compute *H*=SHA-1(5D *OS Family,Hash*, *prefix*, 00 00) the total length is 11 byte, and we need first 2 words, and right lift H's second word by 2 bits.

And now, we get an equation as below:

*Signature**(*Signature**G+*H***K*) = *r* * *G* (mod *p*)

By replacing *K* with *k* * *G*, we get the next equation:

*Signature**(*Signature**G+*H***k***G*) = *r* * *G* (mod *p*)

$Signature^2 + H \cdot k \cdot Signature - r = 0 \pmod{n}$, where *n* is the order of point *G* on the curve

$$Signature = \frac{-H \cdot k \pm \sqrt{(H \cdot k)^2 + 4r}}{2} \pmod{n}$$

Note: not every number has a square root, so maybe we need to go back to step 1 for several times.

Get Private-key From Public Key

I've mentioned that the private key *k* is not included in the BINK resource, so we need to break it out by ourselves.

In the public key:

K(x,y) = *k* * *G*, we only know the generator *G*, and the product *K*, but it is hard to get *k*.

The effective method of getting *k* from *K(x,y)* = *k* * *G* is Pollard's Rho (or its variation) method, whose complexity is merely $O(\sqrt{n})$, where *n* is the order of *G*. (*n* is not included in public key resource, so, we need to get *n* by Schoof's algorithm)

Because a user cannot suffer a too long product key, the *Signature* must be short enough to be convenient. And Microsoft chooses 62 bit as the length of *signature*, hence, *n* is merely 62-bit long. Therefore, the complexity of computing the private key *k* is O(2^31).