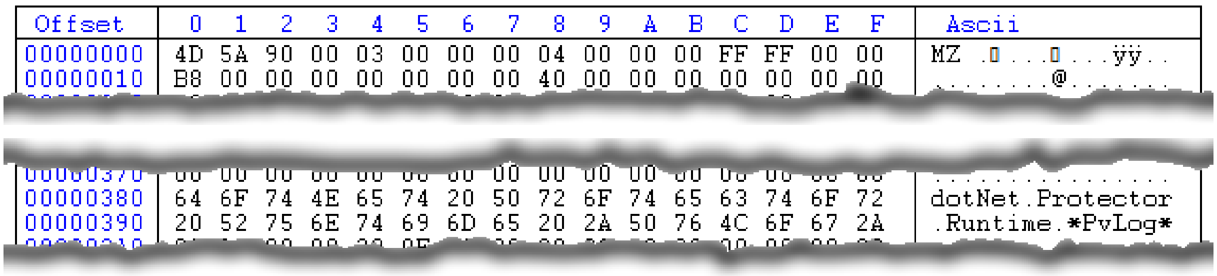
PvLog Signature Structure

1. Signature for dotNet Protector Runtime

A dotNet Protector Runtime image is either Authenticode signed or PvLog signed.
Authenticode signature is outside the scope of this document.

PvLog signed runtime images

File Offset	Type	Description
0x380	BYTE[]	Watermark (dotNet Protector Runtime *PvLog*)
0x3A0	DWORD	Offset from beginning of the file of the signed message
0x3A4	DWORD	Size of signed message



```
Offset     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ .□...□...ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......

00000370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000380  64 6F 74 4E 65 74 20 50 72 6F 74 65 63 74 6F 72   dotNet.Protector
00000390  20 52 75 6E 74 69 6D 65 20 2A 50 76 4C 6F 67 2A   .Runtime.*PvLog*
```

Process for Generating the Pvlog dotNet Protector Runtime Image Hash
Generating the hash for a dotNet Protector image is similar to computing a PE Authenticode hash, with the exception of the runtime watermark and the signature itself that should be omitted.
All data in sections of the PE image that are specified in the section table are hashed in their entirety except for the following exclusion ranges:

- **The file CheckSum field of the Windows-specific fields of the optional header**. This checksum includes the entire file (including any attribute certificates in the file). In all likelihood, the checksum will be different than the original value after inserting the Authenticode signature.
- **The Runtime watermark and message pointers (from file offset 0x380 to 0x3A7)**
- **Information related to attribute certificates**. The areas of the PE image that are related to the Authenticode signature are not included in the calculation of the Runtime image hash because Authenticode signatures can be added to or removed from an image without affecting the overall integrity of the image. PvLog Runtime Hash excludes the following information from the hash calculation:

  The Certificate Table field of the optional header data directories.

  The Certificate Table and corresponding certificates that are pointed to by the Certificate Table field listed immediately above.

  To calculate the PE image hash, PvLogSigVerif orders the sections that are specified in the section table by address range, then hashes the resulting sequence of bytes, passing over the exclusion ranges.
- **Information past of the end of the last section**. The area past the last section (defined by highest offset) is not hashed. This area commonly contains debug information. Debug information can generally be considered advisory to debuggers; it does not affect the actual integrity of the executable program. It is quite literally possible to remove debug information from an image after a product has been delivered and not affect the functionality of the program. In fact, this is sometimes done as a disk-saving measure. It is worth noting that debug information contained within the specified sections of the PE Image cannot be removed without invaliding the PvLog Runtime signature.

2. Signature for dotNet Protector protected assemblies

Like the non-authenticode runtime, protected assemblies have a watermark at file offset 0x380.

This watermark includes either a 64-bit publisher Id if licensed or 0 (64-bit) if protected by an evaluation version. It also includes a 64-bit computer hash (the computer on which the program has been protected).

In the case of an evaluation edition, altering the watermark or the signature will prevent the protected assembly from initializing, thus from running.

The protected image is hashed using the strong name hashing algorithm and this hash is signed with a key-pair whose public key is signed and time-stamped by PV Logiciels and embedded into the protected assembly.

File Offset	Type	Description
0x380	BYTE[]	Watermark (dotNet Protector)
0x390	QWORD	Publisher Hash
0x398	QWORD	Computer Hash
0x3A0	BYTE[256]	Signed Hash (2048-bit)
0x4A0	BYTE[]	Signed and time-stamped public key

Process for authenticating a protected assembly
Verify message signature and time-stamp

Message should be signed by PV Logiciels / LA TESSOUALLE / FRANCE

Signer cert should be delivered by a trusted authority (currently Symantec class 3 SHA256 Code Signing CA)

Message should include a RFC3161 time-stamp token (`1.2.840.113549.1.9.16.1.4`) countersigned by a trusted authority (currently COMODO)

Extract the public key (what has been signed)

Verify that the signed hash matches by computing a strong name hash on the file and verifying the signature