PODPORA ALGORITMŮ SHA-2 VE FORMÁTU S/MIME V PROSTŘEDÍ OS MICROSOFT WINDOWS

Zhotovil:	Němec Radek
Datum vzniku:	18.6.2010
Jméno souboru:	Podpora algoritmů SHA-2 ve formátu SMIME v prostředí OS Microsoft Windows.doc
Počet stran:	16

Aquasoft

VERZE A REVIZE DOKUMENTU

Verze	Datum	Autor	Popis
1.0	18. 6. 2010	Radek Němec	Úvodní verze

OBSAH:

1. CÍL DOKUMENTU 5
1.1. Shrnutí použití kódování SMIME pro elektronický podpis 5
2. ÚVOD 6
2.1. Co je to hashovací algoritmus 6
2.2. Algoritmy rodiny SHA-2 7
3. NUTNÉ PODMÍNKY PRO POUŽÍVÁNÍ SHA-2 ALGORITMŮ V OPERAČNÍM SYSTÉMU MICROSOFT WINDOWS 8
3.1. Nutné podmínky pro ověření důvěryhodnosti certifikátů podepsaných algoritmy SHA-2 8
3.2. Nutné podmínky pro vytváření a ověřování podpisů zpráv ve formátu S/MIME za použití algoritmů SHA-2 9
3.2.1. Podporované operační systémy 9
3.2.2. Správná hodnota Cryptographic Service Provider při generování žádosti o certifikát 10
3.3. Oprava certifikátu pro zajištění správného výběru CSP při podepisování zprávy 11
3.3.1. Oprava CSP u certifikátu pomocí prostředků Microsoft .NET Framework 11
3.3.2. Žádost o certifikát byla vygenerována na Windows XP SP3 s CSP Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype) a certifikát nelze importovat do ostatních systémů, nebo je importován se špatným CSP 12
3.3.3. Žádost o certifikát byla vygenerována na jiném systému než Windows XP, import do systému Windows XP funguje, ale nelze použít SHA-2 algoritmy pro podpis 12
3.3.4. Žádost o certifikát byla vygenerována s chybně zvoleným CSP 12
4. VLASTNÍ IMPLEMENTACE PODPISU NEBO OVĚŘENÍ S/MIME ZPRÁVY POMOCÍ MICROSOFT .NET FRAMEWORK VERZE 2.0 A VYŠŠÍ 13
4.1. Použití SHA-2 algoritmů při podepsání S/MIME zprávy 13
4.2. Ověření S/MIME zprávy podepsané algoritmem rodiny SHA-2 v operačních systémech Windows Vista, Windows Server 2008 a Windows 7 14
4.3. Ověření S/MIME zprávy podepsané algoritmem rodiny SHA-2 v operačních systémech Windows XP a Windows Server 2003 14
5. VYUŽITÍ KOMPONENT TŘETÍCH STRAN 16

1. Cíl dokumentu

V rámci problematiky zaručeného elektronického podpisu využívaného Celní správou ČR dochází v průběhu roku 2010 ke změně technických parametrů elektronického podpisu. To může mít dopady na externí subjekty komunikující v rámci celních agend s centrálním bodem - ECR bránou. Ta pro předání dat opatřených zaručeným elektronickým podpisem připouští dva způsoby kódování:

- XML signature dle doporučení W3C,
- S/MIME.

První způsob je nově zavedený v průběhu roku 2009 a jeho problematice jsou věnovány samostatné dokumenty. Při použití druhého způsobu, který je rozšířen zejména mezi dodavateli programů pro komunikaci v agendě tranzitu, dovozu a vykazování pro Intrastat, lze narazit na celou řadu překážek, protože operační systémy Microsoft Windows poskytují různou podporu kryptografických algoritmů zejména podle doby, kdy byly uvedeny na trh.

Podpoře vytváření a ověřování elektronického podpisu kódovaného pomocí SMIME na operačních systémech Microsoft Windows je věnován tento dokument. Je orientován převážně na technicky orientované pracovníky s rámcovou znalostí problematiky elektronických podpisů.

1.1. Shrnutí použití kódování SMIME pro elektronický podpis

Na základě dále uvedených skutečností lze konstatovat, že pro nově vydávané certifikáty a od roku 2011 i pro povinný algoritmus z rodiny SHA-2, lze při zachování kódování S/MIME použít pouze operační systém Microsoft Windows XP SP3 a novější.

2. Úvod

Ministerstvo vnitra na základě vyhlášky č. 378/2006 Sb. o postupech kvalifikovaných poskytovatelů certifikačních služeb, zveřejňuje kryptografické algoritmy a jejich parametry, které mohou být použity pro elektronický podpis ve smyslu zákona č. 227/2000 Sb., o elektronickém podpisu. Při stanovení zveřejňovaných algoritmů a jejich parametrů vychází jednak z dokumentů vydaných z iniciativy Evropské komise organizacemi, které k tomu EK určila (tzv. ALGO paper zveřejňovaný ETSI), a dále z aktuálního vývoje v oblasti bezpečnosti kryptografických algoritmů. V úvahu bere zároveň stanoviska významných mimoevropských institucí (např. NIST v USA) a českého Národního bezpečnostního úřadu. Na základě aktuálních poznatků v oblasti kryptografie a dokumentu ETSI TS 102 176-1 V2.0.0 (ALGO Paper) Ministerstvo vnitra stanovilo, že:

Kvalifikovaní poskytovatelé certifikačních služeb ukončí vydávání kvalifikovaných certifikátů s algoritmem SHA-1 do 31. 12. 2009. Od 1. 1. 2010 budou tito poskytovatelé vydávat kvalifikované certifikáty podporující některý z algoritmů SHA-2.

Tento dokument se zabývá jednak situací ohledně podpory ověření důvěryhodnosti těchto certifikátů různými verzemi operačního systému Microsoft Windows, jednak samotným vytvářením či ověřováním podpisů s vyžitím algoritmů SHA-2 za pomoci Microsoft .NET Framework a CryptoAPI (součást systému Microsoft Windows poskytující vývojářům software šifrovací, hashovací a další kryptografické funkce).

2.1. Co je to hashovací algoritmus

Hashovací algoritmus je takový algoritmus, který na základě vstupní zprávy vyrobí její „otisk" (hash), což je datová struktura pevně dané délky (např. 160 bitů pro SHA-1). Ideální hashovací algoritmus musí splňovat 4 základní kritéria:

1. Pro jakoukoliv vstupní zprávu musí být jednoduché zjistit její hash.
2. Nelze získat obsah původní zprávy pouze na základě znalosti její hodnoty hash.
3. Nelze změnit zprávu, aniž by se při tom změnila i hash hodnota této zprávy.
4. Každý hashovací algoritmus z principu obsahuje tzv. „kolize", tj. stav, kdy dvě různé vstupní zprávy generují tentýž hash. Je to proto, že zatímco počet různých zpráv je nekonečný, počet různých hodnot hash těchto zpráv je konečný (závisí na délce hash). Nesmí však existovat postup, který by kolize umožňoval najít „v rozumném čase". Rozlišujeme kolize 1. řádu (dvě různé náhodné zprávy mají stejný hash) a 2. řádu (ke známé první zprávě je nalezena jiná zpráva se stejným hash).

Důvodem pro použití takovéhoto algoritmu v rámci elektronického podpisu je výrazné zmenšení dat pro následně používané kódování asymetrickou šifrou (RSA, DSA). V rámci elektronického podpisu se tedy zašifruje pouze hash podepisovaného dokumentu a případně některé doprovodné informace, což výrazně urychluje celý proces vytváření a ověřování podpisu.

2.2. Algoritmy rodiny SHA-2

SHA-2 je označení pro tzv. „rodinu" hashovacích algoritmů, které v současné době nahrazují již ne zcela bezpečný algoritmus SHA-1, u nějž byla zjištěna bezpečnostní rizika (byly nalezeny „dostatečně efektivní" algoritmy pro nalezení kolizí 1. řádu z bodu 4 předchozího odstavce 2.1).

SHA-2 obsahuje tyto algoritmy:

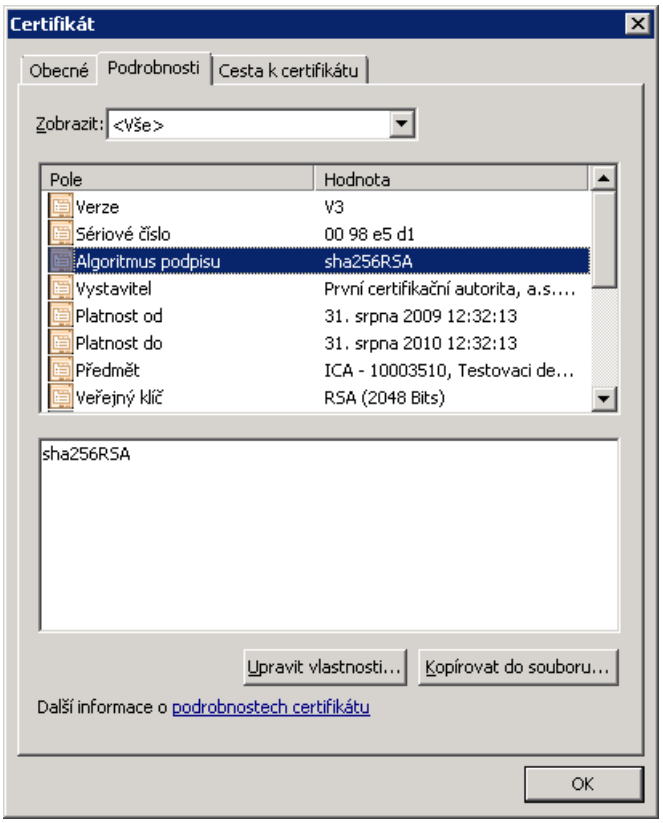
Algoritmus	Délka hash v bitech
SHA-224	224
SHA-256	256
SHA-384	384
SHA-512	512

Poznámka: Operační systém Microsoft Windows neobsahuje pro algoritmus SHA-224 přímou podporu, protože tento algoritmus má stejnou výpočetní náročnost jako SHA-256, ale je méně bezpečný (výpočet SHA-224 a SHA-256 je téměř totožný, liší se pouze počátečním nastavením hodnot *h0* – *h7* a oříznutím výsledného hashe na 224 bitů). Při vytváření zpráv s využitím algoritmů SHA-2 je tedy nutné používat pouze algoritmy SHA-256, SHA-384 a SHA-512, jinak hrozí, že příjemce zprávy nebude schopen ověřit její digitální podpis.

3. Nutné podmínky pro používání SHA-2 algoritmů v operačním systému Microsoft Windows

3.1. Nutné podmínky pro ověření důvěryhodnosti certifikátů podepsaných algoritmy SHA-2

Aby bylo možné používat certifikát pro podpis zpráv, musí být považován za důvěryhodný. Pokud má operační systém považovat certifikát za důvěryhodný, musí být vydaný důvěryhodnou certifikační autoritou (takovou, která je v úložišti certifikátů nainstalovaná v úložišti „Důvěryhodné kořenové certifikační úřady") a operační systém musí být schopen ověřit podpis tohoto certifikátu, tj. musí obsahovat podporu algoritmu specifikovaného v poli *Algoritmus podpisu* certifikátu.



Certifikáty podepsané algoritmy SHA-2 mají v současné době v poli *Algoritmus podpisu* hodnotu sha256RSA, v budoucnu se ale mohou objevit i hodnoty sha384RSA nebo sha512RSA (v závislosti na použitém algoritmu). Podporu ověření důvěryhodnosti SHA-2 certifikátů operačním systémem shrnuje následující tabulka:

Operační systém	Podpora ověření certifikátu podepsaného algoritmem SHA-2
Windows 9x Windows NT, Windows ME, Windows 2000	Ne
Windows XP bez SP3	Ne
Windows XP SP3	Ano
Windows Server 2003	Ano, ale musí být nainstalován hotfix KB938397
Windows Vista	Ano
Windows Server 2008	Ano
Windows 7	Ano

3.2. Nutné podmínky pro vytváření a ověřování podpisů zpráv ve formátu S/MIME za použití algoritmů SHA-2

3.2.1. Podporované operační systémy

Pro použití algoritmů SHA-2 při vytváření nebo ověřování zpráv ve formátu S/MIME s využitím prostředků operačního systému (tj. pomocí CryptoAPI) je třeba mít nainstalován podporovaný operační systém z bodu 3.1 a dále případně splnit podmínku dle 2. sloupce následující tabulky:

<table>
<tr><th>Operační systém</th><th>Podpora vytváření a ověřování podpisů zpráv ve formátu S/MIME</th><th>Cryptographic Service Provider (CSP), který zajišťuje podporu SHA-2 algoritmů</th></tr>
<tr><td>Windows XP SP3</td><td>Ano, ale musí být nainstalován hotfix KB968730</td><td>Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)</td></tr>
<tr><td>Windows Server 2003</td><td>Ano, ale musí být nainstalován hotfix KB968730</td><td rowspan="4">Microsoft Enhanced RSA and AES Cryptographic Provider</td></tr>
<tr><td>Windows Vista</td><td rowspan="3">Ano</td></tr>
<tr><td>Windows Server 2008</td></tr>
<tr><td>Windows 7</td></tr>
</table>

Cryptographic Service Provider je (dále jen CSP) je součást CryptoAPI, která obsahuje množinu šifrovacích a hashovacích funkcí. CryptoAPI těchto modulů obsahuje několik, některé zprostředkovávají základní kryptografické funkce, jiné mohou obsahovat podporu pro práci s čipovými kartami (a certifikáty, které jsou na nich uložené), další přináší podporu nových algoritmů (v našem případě např. podporu SHA-2), atd. Pokud CryptoAPI potřebuje provést nějakou operaci se zprávou, např. vypočítat její hash, vybere příslušný CSP modul a zavolá jeho funkci pro výpočet hash hodnoty.

3.2.2. Správná hodnota Cryptographic Service Provider při generování žádosti o certifikát

Pro bezchybnou činnost aplikací pracujících s digitálními podpisy je tedy nutné, aby operační systém při práci s certifikáty a digitálně podepsanými zprávami vždy zvolil správný CSP modul. Vzhledem k tomu, že v operačních systémech Microsoft Windows se výběr CSP pro podpis zpráv řídí na základě informací uložených v privátní části certifikátu, kam se tyto informace ukládají již při generování žádosti o certifikát, je nutné při podávání žádosti o certifikát zvolit správnou hodnotu CSP. Viz následující tabulka:

<table>
<tr><th>Operační systém</th><th>Online generování žádosti u I.CA, položka Typ úložiště ve formuláři žádosti o certifikát</th><th>Online generování žádosti u PostSignum, položka Umístění soukromého klíče ve formuláři žádosti o certifikát</th></tr>
<tr><td>Windows XP SP3</td><td>Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)</td><td>Operační systém Windows (Win XP SP3)</td></tr>
<tr><td>Windows Server 2003</td><td rowspan="4">Microsoft Enhanced RSA and AES Cryptographic Provider</td><td rowspan="4">Operační systém Windows</td></tr>
<tr><td>Windows Vista</td></tr>
<tr><td>Windows Server 2008</td></tr>
<tr><td>Windows 7</td></tr>
</table>

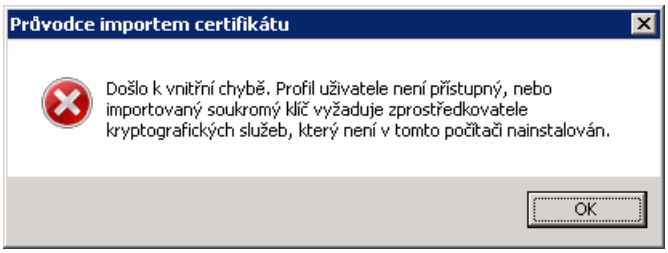
Pokud je certifikát vydaný s jinou hodnotou než je pro daný operační systém (kde se žádost generuje) uvedená v tabulce, **nemusí fungovat podepisování** algoritmy SHA-2, protože operační systém zvolí pro práci s privátním klíčem jiný CSP modul, který nepodporuje algoritmy SHA-2. **Pozor!** Na výběr CSP pro **ověření** výše uvedené nastavení nemá vliv! Proč tomu tak je – viz odstavec 4.2

Pokud je to možné, doporučujeme žádost o certifikát generovat přímo na počítači, kde bude tento certifikát používán - vyhnete se tak případným problémům s CSP. Zároveň není doporučeno přenášet (pomocí exportu a importu certifikátů z/do úložiště certifikátů) certifikáty mezi systémy Windows XP a ostatními. To proto, že název CSP v systému Windows XP (*Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)*) je jiný, než ve všech ostatních systémech (*Microsoft Enhanced RSA and AES Cryptographic*

Provider). Pokud je přeci jen nutné certifikát přenést, je nutné opravit CSP přidružený k privátnímu klíči PFX souboru, viz bod 3.3.

3.3. Oprava certifikátu pro zajištění správného výběru CSP při podepisování zprávy

Pokud vznikne podezření, že podpis zprávy nelze provést z důvodu špatně zvoleného CSP při generování žádosti o certifikát, nebo je certifikát přenášen mezi 2 počítači, z nichž jeden z nich má nainstalován operační systém Windows XP a druhý nikoli, popřípadě nejde certifikát importovat do úložiště s chybou:



je potřeba opravit CSP přidružený k privátnímu klíči vydaného certifikátu. Pro opravu CSP v certifikátu je možno použít následující kód uvedený v bodu 3.3.1. Dále lze použít nástroj, který umí manipulovat s formátem PFX a který se používá pro export certifikátů včetně soukromého klíče. Například open-source nástroj OpenSSL, jehož binární distribuci pro Windows lze stáhnout z odkazu:

http://www.openssl.org/related/binaries.html

Postačí nainstalovat Lite verzi. Pro zjednodušení práce s OpenSSL jsou k tomuto dokumentu přiloženy 2 CMD soubory, které za pomoci OpenSSL opravu provádí. Ve všech následujících bodech, kde se pracuje s OpenSSL (3.3.2 - 3.3.4), se předpokládá certifikát vyexportovaný do PFX souboru *certifikat.pfx* (včetně všech rozšířených vlastností). Před jakoukoliv manipulací s certifikátem, která je popsaná v následujících bodech 3.3.1 - 3.3.4, je potřeba provést zálohu původního certifikátu!

3.3.1. Oprava CSP u certifikátu pomocí prostředků Microsoft .NET Framework

Následující kód používá třídu RSAHelper ze souboru *RSAHelper.cs*, který je přiložen k tomuto dokumentu. V příkladu se opravený certifikát ukládá do úložiště certifikátů aktuálně přihlášeného uživatele, ale lze ho samozřejmě uložit pomocí metody X509Certificate2.Export do pole bajtů a to pak uložit do souboru nebo databáze. Parametr `NewCSP` v uvedeném příkladu bude obsahovat správnou hodnotu CSP dle bodu 3.2.1.

```
public void ChangeCertificateCSP(string NewCSP, X509Certificate2 certificate)
{
  X509Store store = new X509Store(StoreName.My, StoreLocation.CurrentUser);
  store.Open(OpenFlags.ReadWrite);
  RSAHelper RSAHelp = new RSAHelper();
  certificate.PrivateKey =
    RSAHelp.GetRSACryptoServiceProviderWithContext(certificate, NewCSP);
  store.Add(certificate);
  store.Close();
}
```

3.3.2. Žádost o certifikát byla vygenerována na Windows XP SP3 s CSP Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype) a certifikát nelze importovat do ostatních systémů, nebo je importován se špatným CSP

Tato chyba nastává proto, že v certifikátu vygenerovaném na Windows XP SP3 je uložen CSP *Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)*, který cílový systém nezná. Certifikát opravíme příkazem:

convert_certificate *certifikat.pfx opraveny.pfx*

Po spuštění příkazu se zobrazí hlášení *„Enter Import Password:Loading 'screen' into random state – done"*, zadáme heslo pro *certifikat.pfx*, které jsme použili pro export certifikátu do souboru *certifikat.pfx*, a stiskneme Enter. Pokud bylo správně zadáno heslo, zobrazí se „MAC verified OK Enter Export Password") a zadáme 2x za sebou heslo pro nový soubor *opraveny.pfx* (minimálně 4 znaky). Původní certifikát poté smažeme z úložiště certifikátů a nainstalujeme soubor *opraveny.pfx* (při importu zaškrtneme, že chceme importovat všechny rozšířené vlastnosti). Poté by měl již certifikát fungovat správně.

3.3.3. Žádost o certifikát byla vygenerována na jiném systému než Windows XP, import do systému Windows XP funguje, ale nelze použít SHA-2 algoritmy pro podpis

Windows XP sice importují (bez jakékoliv varování, či chyby) certifikát vydaný pro CSP *Microsoft Enhanced RSA and AES Cryptographic Provider*, ale pro práci s tímto certifikátem použijí CSP, který neumí pracovat s algoritmy SHA-2. Certifikát opravíme příkazem:

convert_certificate_for_xp *certifikat.pfx opraveny.pfx*

Zbytek postupu je shodný s bodem 3.3.2.

3.3.4. Žádost o certifikát byla vygenerována s chybně zvoleným CSP

V tomto případě nezáleží na tom, jaký operační systém je použit, certifikát obsahuje v privátním klíči informaci o CSP, který nepodporuje podpis algoritmy SHA-2. Certifikát opravíme dle bodu 3.3.2 nebo 3.3.3, v závislosti na operačním systému, na který chceme certifikát importovat.

4. Vlastní implementace podpisu nebo ověření S/MIME zprávy pomocí Microsoft .NET Framework verze 2.0 a vyšší

Jak již bylo řečeno v bodě 2.2, Microsoft Windows neobsahuje podporu pro algoritmus SHA-224. Proto i Microsoft .NET Framework podporuje podpis a ověření SHA-2 podpisů vygenerovaný pouze algoritmy SHA-256, SHA-384 a SHA-512.

4.1. Použití SHA-2 algoritmů při podepsání S/MIME zprávy

Microsoft .NET Framework obsahuje od verze 2.0 třídu SignedCMS, která implementuje podpis a ověření zprávy ve formátu S/MIME. Její využití pro podpis zprávy může vypadat například takto:

```
public enum SMIMEDigestAlgorithm { Default, sha1, sha256, sha384, sha512 };

public bool Sign(Stream inData, Stream outData, X509Certificate2 certificate)
{
  try
  {
    byte[] buffer = new byte[Math.Max(0, inData.Length-inData.Position)];
    inData.Read(buffer, 0, buffer.Length);
    SignedCms signedCMS = new SignedCms(new ContentInfo(buffer), false);
    CmsSigner cms = new CmsSigner(certificate);
    cms.IncludeOption = X509IncludeOption.EndCertOnly;

    if (digestAlgorithm != SMIMEDigestAlgorithm.Default)
      cms.DigestAlgorithm = new Oid(CryptoConfig.MapNameToOID(digestAlgorithm.ToString()));

    signedCMS.ComputeSignature(cms);
    buffer = signedCMS.Encode();
    outData.Write(buffer, 0, buffer.Length);
    return true;
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    return false;
  }
}
```

Aby tento příklad fungoval, musí být splněny podmínky z předchozích bodů a v příkladu uvedený parametr `certificate` musí obsahovat v

```
(certificate.PrivateKey as RSACryptoServiceProvider).CspKeyContainerInfo.ProviderName
```

správnou hodnotu CSP dle operačního systému (viz bod 3.2.1) a zároveň hodnota

```
(certificate.PrivateKey as RSACryptoServiceProvider).CspKeyContainerInfo.ProviderType
```

musí být 24. Pokud `ProviderName` obsahuje hodnotu *Microsoft Base Cryptographic Provider v1.0* nebo *Microsoft Enhanced Cryptographic Provider v1.0* a `ProviderType` obsahuje hodnotu 1, je nutné opravit CSP certifikátu dle bodu 3.3.

4.2. Ověření S/MIME zprávy podepsané algoritmem rodiny SHA-2 v operačních systémech Windows Vista, Windows Server 2008 a Windows 7

Pokud je aplikace, která ověřuje S/MIME zprávu s SHA-2 podpisem, spuštěna v operačním systému Windows Vista, Windows Server 2008, nebo Windows 7, je možné opět použít třídu SignedCMS pro ověření zprávy:

```
public bool VerifySign(Stream inData, Stream outData, out X509Certificate2 certificate)
{
  try
  {
    SignedCms cms = new SignedCms();
    byte[] buffer = new byte[Math.Max(0, inData.Length-inData.Position)];
    byte[] content = null;
    certificate = null;
    inData.Read(buffer, 0, buffer.Length);
    cms.Decode(buffer);
    certificate = cms.SignerInfos.Count == 0 ? null : cms.SignerInfos[0].Certificate;
    content = cms.ContentInfo.Content;
    try
    {
      cms.CheckSignature(false);
    }
    catch (CryptographicException ex)
    {
      MessageBox.Show(ex.Message);
      return false;
    }
    return true;
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    return false;
  }
}
```

Tato metoda bohužel nefunguje na systémech Windows XP a na Windows Server 2003. Je to ze 2 důvodů:

1. Veřejná část certifikátu, která je obsažená v S/MIME zprávě, neobsahuje údaje o tom, jaký CSP se má při ověřování podpisu použít, proto se vždy na všech verzích systému Microsoft Windows používá pro ověření výchozí CSP.
2. Výchozím CSP v operačních systémech Windows XP SP3 a Windows Server 2003 je *Microsoft Base Cryptographic Provider v1.0*, který podporu algoritmů SHA-2 neobsahuje. Naproti tomu na Windows Vista, Windows Server 2008 a Windows 7 je výchozím CSP *Microsoft Enhanced RSA and AES Cryptographic Provider*.

4.3. Ověření S/MIME zprávy podepsané algoritmem rodiny SHA-2 v operačních systémech Windows XP a Windows Server 2003

V systémech Windows XP a Windows je tedy třeba použít přímého volání funkcí CryptoAPI funkcí z crypt32.dll a advapi32.dll následujícím způsobem (funkce používá soubor *CryptoAPI.cs*, přiložený k tomuto dokumentu, který zapouzdřuje volání funkcí CryptoAPI):

```
public bool VerifySignUsingCryptoAPI(byte[] signedData, out byte[] pbDecodedMessageBlob, out
X509Certificate2 certificate)
{
  CryptoAPI.CRYPT_VERIFY_MESSAGE_PARA VerifyParams;
  Boolean res = false;
  Int32 cbSignedMessageBlob = signedData.Length;
  Int32 cbDecodedMessageBlob = 0;

  VerifyParams = new CryptoAPI.CRYPT_VERIFY_MESSAGE_PARA();
  VerifyParams.cbSize = Marshal.SizeOf(VerifyParams);
  VerifyParams.dwMsgAndCertEncodingType = CryptoAPI.X509_ASN_ENCODING |
                                          CryptoAPI.PKCS_7_ASN_ENCODING;
  CryptoAPI.CryptAcquireContext(ref VerifyParams.hCryptProv, null,
                                CryptoAPI.MS_ENH_RSA_AES_PROV,
                                CryptoAPI.MS_ENH_RSA_AES_PROV_TYPE,
                                CryptoAPI.CRYPT_VERIFYCONTEXT);
  GCHandle pCertContext = GCHandle.Alloc(IntPtr.Zero, GCHandleType.Pinned);
  VerifyParams.pvGetArg = IntPtr.Zero;

  // With two calls to CryptVerifyMessageSignature, verify and decode
  // the signed message. First, call CryptVerifyMessageSignature to get the length of the
  // buffer needed to hold the decoded message.
  try
  {
    res = CryptoAPI.CryptVerifyMessageSignature(
      ref VerifyParams,       // Verify parameters.
      0,                // Signer index.
      signedData,     // Pointer to signed BLOB.
      cbSignedMessageBlob,     // Size of signed BLOB.
      null,             // Buffer for decoded message.
      ref cbDecodedMessageBlob,  // Size of buffer.
      //IntPtr.Zero           // Pointer to signer certificate.
      pCertContext.AddrOfPinnedObject()); // Pointer to signer certificate.

    if (!res)
      throw new Win32Exception(Marshal.GetLastWin32Error());

    //   Allocate memory for the buffer.
    pbDecodedMessageBlob = new Byte[cbDecodedMessageBlob];

    //  Call CryptVerifyMessageSignature again to copy the message into the buffer

    res = CryptoAPI.CryptVerifyMessageSignature(
      ref VerifyParams,       // Verify parameters.
      0,                // Signer index.
      signedData,     // Pointer to signed BLOB.
      cbSignedMessageBlob,     // Size of signed BLOB.
      pbDecodedMessageBlob,     // Buffer for decoded message.
      ref cbDecodedMessageBlob,  // Size of buffer.
      pCertContext.AddrOfPinnedObject());     // Pointer to signer certificate.

      if (!res)
        throw new Win32Exception(Marshal.GetLastWin32Error());
      else
      {
        try
        {
          certificate = new X509Certificate2((IntPtr)pCertContext.Target);
        }
        finally
        {
          CryptoAPI.CertFreeCertificateContext((IntPtr)pCertContext.Target);
        }
      }
    }
    finally
    {
      pCertContext.Free();
      CryptoAPI.CryptReleaseContext(VerifyParams.hCryptProv, 0);
    }
    return true;
}
```

5. Využití komponent třetích stran

Pro vytvoření či ověření S/MIME zpráv s podpisy vytvořenými algoritmy SHA-2 je kromě již zmíněného Microsoft .NET Framework samozřejmě možné použít software třetích stran, například těchto:

Autor	Produkt	Stránky produktu
Bouncy Castle	Bouncy Castle	http://www.bouncycastle.org/
EldoS	SecureBlackbox	http://www.eldos.com/sbb/
Chilkat Software	Chilkat S/MIME	http://www.chilkatsoft.com/mime-dotnet.asp
OpenSSL	OpenSSL	http://www.openssl.org

Pozor: Uvedené produkty mohou a nemusí záviset na prostředcích operačního systému. Není tedy zaručeno, že při jejich použití je možné používat algoritmy SHA-2 i v operačních systémech, které nejsou podporovány (Windows 2000 a starší, Windows XP bez SP3).