# How to Break Microsoft Rights Management Services

Martin Grothe Ruhr-University Bochum Christian Mainka Ruhr-University Bochum Paul Rösler Ruhr-University Bochum

Jörg Schwenk Ruhr-University Bochum

## Abstract

Rights Management Services (RMS) are used to enforce access control in a distributed environment, and to cryptographically protect companies' assets by restricting access rights, for example, to *view-only*, *edit*, *print*, etc., on a per-document basis. One of the most prominent RMS implementations is Microsoft RMS. It can be found in Active Directory (AD) and Azure. Previous research concentrated on generic weaknesses of RMS, but did not present attacks on real world systems.

We provide a security analysis of Microsoft RMS and present two working attacks: (1.) We completely remove the RMS protection of a Word document on which we only have a *view-only* permission, without having the right to *edit* it. This shows that in contrast to claims made by Microsoft, Microsoft RMS can only be used to enforce all-or-nothing access. (2.) We extend this attack to be stealthy in the following sense: We show how to modify the content of an RMS write-protected Word document issued by our victim. The resulting document still claims to be write protected, and that the modified content was generated by the victim. We show that these attacks are not limited to local instances of Microsoft AD, and can be extended to Azure RMS and Office 365.

We responsibly disclosed our findings to Microsoft. They acknowledged our findings (MSRC Case 33210).

## 1 Introduction

Access control in distributed environments. Access control (discretionary or role-based) can be enforced in closed environments, for example, on files controlled by an operating system. Once a file leaves this closed environment, the file becomes freely accessible. This problem is well known as *data leakage*, and *data leakage prevention (DLP)* is a major security goal in each company.

One major class of DLP tools are Digital Rights Management (DRM) systems, which are often called *Enter*- *prise Rights Management (ERM)* systems when applied to company data. DRM/ERM systems protect data by encrypting and digitally signing it, together with access rights, before transferring them over an unprotected network, for example the Internet. This protects data against unauthorized read and write access.

Modern ERM systems enforce a complex access control methodology (rights to read, write to, print, extend rights to other entities, etc.) through a combination of document encryption and signing, key management and rights-enforcing applications. Previous research [11, 43, 48] has concentrated on generic weaknesses of this concept, but no attacks on industry-grade ERM systems have been published to date.

**Microsoft RMS in Active Directory and Azure.** Rights Management Services (RMS) is deployed in Active Directory Rights Management Services (AD RMS) and since Version 2008, it is a core part of Windows Server. In recent years, Microsoft has adapted RMS to their new cloud platform *Azure*, making Azure RMS available on mobile platforms (iOS, Android, Windows Phone) as well. RMS is maybe the most widely used ERM implementation and integrated into banking business [8], enterprise information management [41], and hardware security modules (HSMs) [47]. The UK Ministry of Defence is currently integrating RMS as a part of their *Defence-as-a-Platform* [50]. Its wide integration makes RMS an important target for attacks.

Microsoft RMS can be used to protect Microsoft Office documents (rights enforcement is integrated into the Microsoft Word, Excel and Powerpoint clients), and can be integrated in other company applications via a dedicated *RMS API*. Both application areas cannot be mixed: Especially, the RMS API cannot be used to manipulate ERM rights enforced on Microsoft Office documents.

**Breaking Microsoft RMS.** Building a demonstrator to show that the RMS API is essentially limited to all-or-nothing ERM protection is relatively easy. Much more

challenging is the task to reverse-engineer the use of the rights enforcing API by Microsoft Office applications and this way bypass the countermeasures implemented by Microsoft.

In the paper we describe two different attacks on Microsoft Active Directory (AD) RMS for Microsoft Office. (1.) Removing the RMS protection from a protected Word document resulting in a completely unprotected document. (2.) Stealthy content modification of an RMS protected Word document. Both attacks only require the *view* access right on the RMS protected file. This is the minimal right, which can be assigned to a group or users in the Microsoft RMS environment.

These attacks have a severe impact on real-world companies which rely on the RMS protection. While in principle it is always possible to leak data from a read-only document by making screenshots or photos, this in practice prevents large-scale data leakage. The attacks presented in this paper make large-volume data leakage of Microsoft RMS protected documents feasible.

Moreover, the stealthy version of our attack may enable new scenarios, for example, in spear phishing or industrial espionage. If an attacker, having only read access (e.g. an ordinary employee), may arbitrarily change the content of a document issued by the CEO of a company, this could be used to trick other employees on performing illegal actions.

Due to the efforts by Microsoft making the RMS system as much platform independent as possible, our attacks are also applicable on Microsoft's cloud platform Azure and Office 365 [5], leading to a real extension in the attack surface.

We have implemented our attacks as a proof-ofconcept and communicated our results with Microsoft.

Contributions. We make the following contributions:

- ► We describe in-depth Microsoft RMS for AD RMS and Azure RMS, including the PKI infrastructure and the used file format.
- We describe and implement two novel attacks on Microsoft RMS:
  - (1.) removing the protection of an arbitrary file, granting full access to the attacker.
  - (2.) stealthily breaking the integrity of an RMS protected file and allowing modifications of the content. The resulting file looks as if it has been created by another person.
- ► We implemented both attacks in a tool called *Disabeling Attacks on Rights Management Services* (*DisARMS*).<sup>1</sup> We used DisARMS to evaluate our attacks against Microsoft AD RMS with Office 2013

and Azure with Office 365. We then communicated our results with Microsoft (MSRC Case 33210).

➤ We provide possible countermeasures. Due to the current design of RMS, finding sufficient countermeasures are not trivial.

# 2 From classical DRM to modern ERM

Classical access control, with its two most popular paradigms Discretionary Access Control (DAC) and Role Based Access Control (RBAC), can directly by enforced in closed systems like operating systems or databases. DAC is an entity-based access control concept. Access to ressources (e.g., files, directories, devices, ...) is granted to single or multiple entities. In RBAC [1], access control decisions are based on roles. Each entity is assigned one or more roles. A typical example for this concept is AD from Microsoft.

**Digital Rights Management.** In the 1990s, companies like Microsoft and Real Networks introduced consumermarket DRM implementations to protect digital audio and video distribution. In 1998, the *Digital Millenium Copyright Act (DMCA)* made it illegal to circumvent DRM systems, regardless of their security. DMCA was first used in a lawsuit against dozens of systems administrators in 1999 in an attempt to limit the distribution of the DVD decyption software deCSS.

Microsoft used DRM in Windows Media Player [10] to protect multimedia files, and Real Networks implemented it in Real Player. Adobe released their own DRM systems to protect e.g. PDF files. Most of these systems could be broken generically, by just playing and rerecording ("ripping") the multimedia content.

With DRMv2, Microsoft introduced the Windows Media Rights Manager 7 SDK, allowing to manage the rights of given multimedia files (e.g., the total number of allowed plays). DRMv2 separated the use licenses for costumers from the protected file, previously both were stored in one file. This concept is similar to modern RMS implementations [10].

**Enterprise Rights Management.** In June 2003 Microsoft introduced its evolved DRMv2 system named Rights Management Services as an Add-on for the Windows Server 2003 operating system. It was now named Enterprise Rights Management to highlight that it was targeted especially to the corporate market [38].

With Windows Server 2008, RMS was fully integrated as a new server role of the Active Directory server [17, 28]. In 2013 Microsoft introduced ERM for its cloud platform Azure under the name Azure RMS [42]. Sometimes Information Rights Management (IRM)

<sup>&</sup>lt;sup>1</sup>https://github.com/RUB-NDS/MS-RMS-Attacks

is used as a synonym for ERM (e.g. by EMC's in Documentum IRM [7]).

Microsoft AD RMS. Microsoft Active Directory Rights Management Services [13] are an on-premise ERM system by Microsoft and part of current Windows Server operating systems. They give employees of a company the ability to fine-grained set rights on files they create. Windows Server is a basic system with different enhancements, so called server roles. These roles can be added separately. RMS is one of these server roles which works together with another server role, the Active Directory Domain Services (AD DS). That role is an administration tool whereby users, groups, server's, and other objects of a companies' infrastructure can be controlled. The RMS then adds a special Public Key Infrastructure (PKI) to the AD DS. By this means, users of the AD which are the employees of a company can control the access to files they create.

Microsoft Azure RMS. Azure RMS is part of Azure, which is a cloud platform system for storage and infrastructure sharing in- and outside of companies. Azure combines the advantages of a cloud system with those of ERM and a DRM system. An Azure instance manages company users and their software and data requirements. For this Azure requires an own AD within the cloud. This AD is called Azure AD. Azure can be used to create and work on a Virtual Machine (VM) or store data in the cloud as well as to share those stored data and manage the usage of documents with an integrated RMS system. During our research we analyzed Azure and Azure RMS and could successfully apply our findings from AD RMS to Azure RMS.

**Other ERM Implementations.** RMS systems were also implemented by companies like Adobe or SAP. Adobe LiveCycle manages PDF documents comparable to Microsoft RMS for office documents.

## **3** Security Model

To access an ERM protected document, which contains *access conditions* (either DAC or RBAC based), an entity needs a *license*, which contains *access rights* bound to this entity. The strength of this binding (and therefore the strength of the ERM protection) depends on the enforcement mechanism, which may be either implemented partially in hardware (e.g. smart cards) or completely in software.

Since ERM protected documents are often sent over the Internet, our model provides the adversary with full access to such documents. He also is allowed to acquire a license with limited access rights (e.g. read-only).

The goal of the adversary is to get additional access

rights on the document (e.g. write, forward, print, ...), without being legally entitled to get these. These advanced access rights may be acquired by interacting with a rights management server, or without such interaction. In the present paper we show how access rights can be extended without any interaction.

Please note that we do not consider "ripping" attacks (e.g. taking screenshots of protected documents and forward these screenshots), as these attacks do not scale well. We also argue that all software-based ERM systems could principally be broken by an attacker having root access in the OS, where the enforcement mechanism is running and the attacker have the view right on the document. Instead, we give full working implementations, running without root privileges, to extend read-only to full access, both in direct and stealthy mode. Note, that even with root privileges, but without view right on the document, the RMS protection can not be bypassed.

## 4 Microsofts Rights Management Services

#### 4.1 General Overview

AD RMS is an ERM solution developed by Microsoft and available since Windows Server 2003. We analyzed the implementation in the stable version of Windows Server 2012 R2. We activated the RMS features by enabling the server role for AD RMS in the AD DS [32]. This is an optional server role for companies which already use the AD DS to manage their infrastructure on-premise. The role enables employees to create protected documents with specific rights for selected users or groups.

Microsoft further provides an RMS API. To develop applications with the API, the RMS SDK is necessary. Then applications can be created to protect generic files via the AD RMS or Azure RMS. In general most of the API functions can also be used to do legit edits on native protected Office files such as \*.*docx* or \*.*pptx*, but therefore the correct rights are necessary. Also countermeasures were implemented to prevent bypasses of the RMS protection with the API functions (cf. Section 5.1).

## 4.2 Test Setup

For our evaluation we installed one Windows Server and two Windows 7 Enterprise clients. All machines had access to a dedicated network. We then configured the two server roles: (1.) AD DS and (2.) AD RMS. In addition to the server-side component, the Office Suite (Office 2013 Professional) by Microsoft was required on the clients.

Analysis. Before we could start testing different scenarios on our clients we had to create different users for our AD. We monitored the communication between client computer and the AD RMS server with Wireshark. Then we started work flows like they would happen in a real company, such as domain join of a client, first login of a user, creation of an AD RMS protected file and opening a protected document. We used a set of users for this action to examine the difference in the communication. We disabled TLS, to eavesdrop the traffic. Note, that disabling TLS has no impact on our attacks, they work the same way with TLS enabled.

Attack. Based on the information gathered in the analysis phase, we were able to verify and extend the information extracted from Microsoft documentation and construct our attacks. Our attacks work in every network which uses Microsoft AD DS or Azure AD with enabled RMS server role (AD RMS or Azure RMS). Further in every situation Office 365 is used together with Azure RMS to protect documents. DisARMS takes a simple ERM protected document as input, with read-only rights.

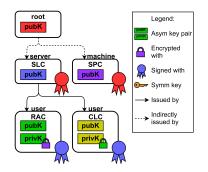


Figure 1: Certificates of the AD RMS PKI.

# 4.3 The Active Directory Rights Management Services in detail

**AD RMS PKI.** The fundamental concept of the AD RMS is a complex PKI (see Figure 1). Each certificate contains an RSA public key. This PKI consists of certificates and licenses. The certificates and licenses are stored in the Extensible Rights Markup Language (XrML) [37]. Certificates and Licenses are mainly structured in three parts: (1.) Issuer (2.) Key (3.) Signature [19, 26, 29, 31, 36]. Additional certificates are located between the root certificate and the Server Licensor Certificate (SLC) respectively the Security Processor Certificate (SPC), but they are not necessary to understand this ERM solution.

The SPC [31] contains the identity of a computer, but it is also bound to a certain user: Its private key is encrypted with the login password of the user and the Data Protection Application Programming Interface (DPAPI) [16]. The SLC contains the identity of the rights management server, and is used to verify user identities and to grant user's access to protected files.

A user identity consists of two certificates, the Rights Account Certificate (RAC) and the Client Licensor Certificate (CLC), which are issued by the SLC. Both RAC and CLC contain a whole key pair.<sup>2</sup> The CLC is used to protect files and the RAC is used to get access to protected files [19, 29]. The private key of the CLC is encrypted with the public key of the RAC. The private key of the RAC is encrypted with the public key of the SPC.

**AD RMS Licenses.** A Publishing License (PL) contains the name of the author (alice@company.com), a list with pre-configured access rights for different users or groups (hr@company.com), and the content key [26]. Examples for preconfigured access rights are *view*, *edit* or *print* [15, 20]. Once the PL is created, it is encrypted with the public key of the SLC. The content of the PL is signed by the author's (Alice's) CLC private key.

The Use License (UL) contains all ERM-related information for one specific user, for example, a list with pre-configured rights for this specific user, the name of the author, and the content key [36]. Every data in the UL is encrypted with the public key of the RAC of the UL requesting user and signed by the servers SLC private key.

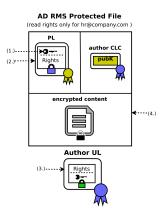


Figure 2: AD RMS protected Word-File. The encrypted content key is contained in the PL, which itself is encrypted with the SLC public key and signed with the private key of the author's CLC.

**Protecting Documents.** For the creation of a protected Office document (Figure 2) four steps are executed locally on the author's machine [27]:

(1.) Alice's client software (e.g., Word) generates a random content key and uses it to encrypt the whole

<sup>&</sup>lt;sup>2</sup>Although a private key is in these data structures, Microsoft documentations nevertheless refer to them as certificates.

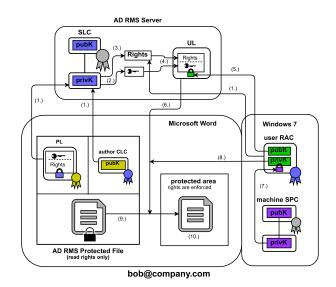


Figure 3: Accessing protected Word-File in AD.

existing document file.

- (2.) Alice generates a PL, and
- (3.) a UL for herself.
- (4.) The protected word document consists of the PL for the server, the author's public part of the CLC, and the encrypted document.

The protected files can then be distributed over a network share folder, via E-Mail or by copying the file to an external storage.

Accessing Protected Documents. Once Bob wants to access the protected document, he uses his RAC to request a UL from the server [12]. The process is depicted in Figure 3:

- (1.) The client software extracts the PL and CLC of the author (alice@company.com) from the Word document and sends it together with the public part of the locally stored RAC of the requesting user to the server.
- (2.) The server uses its SLC to decrypt the content key from the PL.
- (3.) Afterwards the server extracts the access rights from the PL and validates whether the requesting user (bob@company.com) is allowed to access the file.
- (4.) In case the validation ended successfully the server generates a new UL for the user (with content key and access rights) and sends it to the client software.
- (5.) The UL is encrypted with the public key of the user's RAC.

- (6.) The client software receives the UL.
- (7.) The private key of the user's RAC is decrypted via the private key of the SPC.
- (8.) The UL gets decrypted by the private key of the user's RAC.
- (9.) Access rights and content key get extracted and are used to
- (10.) Decrypt and open the document in a protected environment. There the rights are enforced by deactivating options like printing or copying the content.

Opening a protected document for the first time always requires a connection to the AD RMS server. After this initial connection the UL can be stored in the cache of the client. The expiration time (from 1 day to never) of caching a UL can be adjusted in the PL of the protected document or in the template used to create the document. [40].

## 4.4 The Azure RMS in detail

Azure Rights Management Services is part of Azure Active Directory and enables its users to share files, for example, Office documents, with other users and preserve the control over the content and its distribution. In comparison to the classical AD RMS, Azure RMS client software is available on all modern platforms (Android, iOS, Linux, Mac OS, Windows) [18]. Therefore, protected content can be consumed on mobile devices, laptops and computers.

The work flow behind Azure RMS is similar to AD RMS, when it comes to the creation of protected documents or processing those documents. Both use the same client software (Office 2010-2016) to create protected documents, Azure RMS also uses the same certificates (RAC, SPC, etc.) and licenses (UL and PL). Further the PKI behind Azure RMS is smaller than for AD RMS. In contrast to the AD RMS, the authentication is done against an Azure AD instance and not the AD DS. The authentication process requires a valid Azure account. In case the authentication procedure was finished successfully, the Azure AD server sends back a token, the client software can use to communicate with the Azure RMS instance. The user is then able to consume or create a protected document via the client software [21].

Accessing Protected Documents. This process is shown in Figure 4. We assume that the user is already authenticated to the Azure AD and that the RAC is already stored on the client device. (1.) The client software sends the PL that is contained in the protected file, as well as the RAC, of the user, to the Azure RMS instance. (2.) Azure RMS decrypts those elements with its private key of the SLC. (3.) A list of access rights for the requesting user is created by the Azure RMS instance, according to the policy. (4.) The content key is extracted from the decrypted policy. (5.) A UL is created from the content key and the access right list. The UL is encrypted with the RAC public key of the requesting user. (6.) Afterwards the UL is sent to the Azure RMS client. (7.) The client software decrypts the UL with the private key of the user's RAC. (8.) The protected document is decrypted with the symmetric content key from the UL. (9.) The opening software (e.g., Microsoft Word) gets the decrypted document with a list of rights from the RMS plugin. The listed rights are enforced by Word [49].

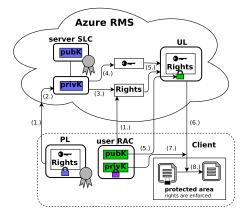


Figure 4: Accessing a protected Word file in Azure.

# 5 Attacking Microsoft RMS

As described previously attacking a DRM or an ERM system and removing the protection is pretty easy to accomplish, in case the attacker has control over the underlying operating system. This normally requires some kind of advanced user privileges. This is an unrealistic scenario for most employees in modern companies. Our goal was to show that this attack is also possible, when a user just has very limited rights on the client operating system. Further we never attack the operating system, instead DisARMS exploits design flaws in AD RMS and Azure RMS. Further we extended the previous attack and enable the user, with the same client privileges as before, to modify protected content without leaving evidence of the modification to other users or administrators.

# 5.1 Attack Challenges

The goal of our first attack is to completely remove the protection of a given protected document, in case the user

has at least the *view* right. We therefore tried to use the RMS API in order to decrypt the encrypted content of a protected office document. Before we could implement our first attack we encountered the first challenge.

**Pre-production Hierarchy.** In the How-to use guide of the RMS SDK it is stated out, that the development of an AD RMS application should be done in a pre-production development environment [33]. After we switched our client and server to this environment we were not able to open any previously protected file anymore. This was caused by the pre-production hierarchy, which replaces the original AD RMS PKI with a development PKI. Though, we switched back to the productive environment and started the development of DisARMS.

Bypassing RMS API Countermeasures. Microsoft AD RMS implementation provides several API functions and data structures [25], which can be used by developers to implement their own RMS application. First it looked straightforward to implement an application which decrypts or encrypts an office file using the available RMS API functions. Examples for these functions are IpcfDecryptFile or IpcfEncryptFileStream. As a countermeasure for this method Microsoft checks whether the user. who tries to process the protected document, has the access right to export a document to another, potentially unprotected file format. If the user does not have the export right, the Ipc-functions simply return NULL instead of the decrypted/encrypted document. To bypass this restriction, we had to avoid those IpcfDecrypt-File/IpcfEncryptFile functions. We therefore emulate these functions by implementing our own, custom encrypt/decrypt functions (Appendix Listing 1 line 42 - 78) that rely on low level crypto library functions offered by the RMS API.<sup>3</sup>

**Reproduce the Office Work-Flow.** To successfully execute our custom decrypt/encrypt function we had to reproduce the work-flow from Figure 4 via RMS API function calls. In total the RMS API offers 48 functions [30]. So we had to find the right functions to reproduce the work-flow (as described in Section 5.2 and Section 5.3)

**Reverse Engineering the Correct License Structure.** The IpcEncrypt/IpcDecrypt function requires a valid key handle to get executed. This is accomplished via the function *IpcGetKey* of the RMS API. Instead of the expected key handle, the value NULL was returned by the function. After eliminating all other causes of error, we analyzed the binary structure of some licenses generated by the RMS Sample application [23]. The encrypted data created by the Sample application, when content was protected, revealed that this program stores 3 static

<sup>&</sup>lt;sup>3</sup>E.g., we could only decrypt one block and had to implement the cipher mode manually (AES-ECB)

bytes in front of the actual license data (Appendix Listing 1 line 4). This was never observed by the original Office client software. This requirement is further not documented by Microsoft nor described online. After we prepended these bytes to our license structure we received the valid key-handle to the content key of a our native protected document.

**Code Signing in Production Environments.** Microsoft states out in their How-to use guide, that productive applications need to be signed in order to get executed on client machines [34]. Therefore the application must be signed via a Production License Agreement requested via Microsoft, but surprisingly this was not necessary and we could execute DisARMS without any signing.

Finding the Correct Padding Scheme. For the second attack we need to re-encrypt our modified data. Microsoft has two symmetric algorithms (AES-ECB and AES-CBC-4k) for content encryption [24] and offers two different cryptography modes to secure the licenses and certificates [14]. They distinguish between the protection offered by their own implementations (native) and third party applications (generic) [35]. This results in a different selection of the content encryption algorithm. All analyzed Microsoft RMS products use the AES-ECB mode, instead of the proprietary CBC-4k mode of operation, which leads to a weaker level of security. If the CBC-4k mode of operation is used the padding is done by the API function *IpcDecrypt* and *IpcEncrypt* automatically. For the ECB mode we needed to reverse engineer the padding scheme. As described in Figure 2 every protected file has an encrypted content part. The first 8 bytes of this part contain the length of the content after it got decrypted. This way the padded bytes can be random data, which are ignored by the client software after it decrypts the encrypted content part.

# 5.2 DisARMS Attack #1: How to Remove RMS Protection

The goal of this attack is to completely remove the protection of a given protected document, in case the user has at least the *view* right. DisARMS requires the RMS Clients 2.1 software, as well as the C++ Redistributable 2015 to be installed on the client computer where the *view* right is valid. The attack proceeds as follows (cf. Figure 5):

- (1.) Charlie (who has view rights on the document) retrieves the protected document.
- (2.) DisARMS splits the protected document into the following parts: (a) the encrypted content, (b) the Publishing License (PL), (c) and the author's CLC.

- (3.) The PL and the CLC are read and parsed, and this information is used for requesting a UL via Microsoft's RMS library, which either requests it from the AD RMS server or uses a previously requested UL from the machine's local cache.
  - (a) If a request to the AD RMS server is used, it contains the public part of Charlie's locally stored RAC and the PL. The server then decrypts the PL in order to determine the access rights. In this scenario, Charlie has only the *view* right and the AD RMS server successfully validates that Charlie is allowed to access the protected document.
  - (b) Since the previous validation step is successful, the AD RMS server uses its SLC to decrypt the **content key** from the PL.
  - (c) The server generates a new UL containing the content key encrypted with the public key of Charlie's RAC and the previously extracted access rights. The UL is then sent back to Charlie's client.
  - (d) If the UL is cached, steps (a-c) are omitted.
- (4.) To retrieve the private key of his RAC, Charlie has to decrypt it with the private key of the machine SPC.<sup>4</sup>
- (5.) Charlie decrypts the UL by using the private key of his RAC and extracts the **content key** plus the **access rights** (i.e., *view*).
- (6.) Charlie then decrypts the document by using this content key and saves the plaintext of the *unprotected* Word document into a new file (*decrypted.docx*), ignoring the *view* right.

In the end, Charlie gets the unprotected version of the original file, containing all content, pictures, tables, and formatting. For a third person, there is no possibility to see, whether this file was originally protected or not.

Please note that this attack breaks the general concept of RMS protection: an attacker can get access to the whole file by only using the minimal access right. Although this attack seems to be straight forward (the attacker has access to the content key), there were lot of difficulties to obtain this attack, see Section 5.1.

# 5.3 DisARMS Attack #2: How to Modify RMS Protected Content

The previously described attack completely removes the RMS protection of a given file if the *view* right is granted

 $<sup>^{4}\</sup>mbox{This}$  step cryptographically binds Charlie's RAC to a specific machine.

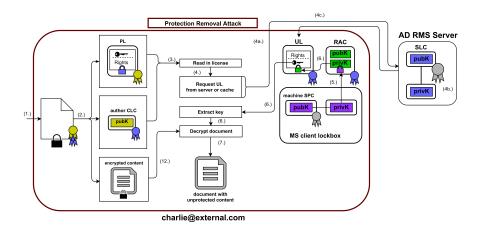


Figure 5: DisARMS Attack #1 – Decrypting a protected Word-File

to the attacker. In this section, we describe an attack that goes one step further: The attacker (Charlie) again has *view* access on a given file (created by Alice), but this time, he modifies its content. In the end, the file looks as if it has been created by the original author (Alice), but Charlie can arbitrarily modify the protected contend leaving the original access rights unchanged. For example, he can add or remove text content of a protected Word file, although Charlie has only the *view* right.

The attack is depicted in Figure 6 and described in the following:

- (1.) Charlie gets the RMS protected file, for example, a Microsoft Word document and can remove the RMS protection as described in the previous attack.
- (2.) He can then arbitrarily modify the unprotected content. He can therefore use a common Word processor (e.g. Microsoft Word) or edit the content directly by extracting the *content.xml* contained in the \*.docx file. If Charlie decides to use a full-fledged

Word processor, he has to take care of some automatically generated metadata indicating his edit, for example, the time of the last edit, and his name. To prevent this, he could change his name in the Word processor settings to *Alice*.

- (3.) He then uses again the RMS library to encrypt the modified document using the same content key as used for the decryption. (Steps 3.1 - 3.4 are equal to steps 4 - 6 of the Protection Removal Attack)
- (4.) Finally, he exchanges the encrypted content of the original file with the encrypted modified content.

If Charlie then opens the tampered file with a common word processor, he cannot modify it – he has only the right to *view* it. By inspecting the last modification information, Alice's name appears, but the content of Charlie's choice is shown.

We were quite surprised that this attack works. Since Microsoft RMS heavily uses asymmetric cryptography for signing all licenses and certificates (UL, PL, RAC,

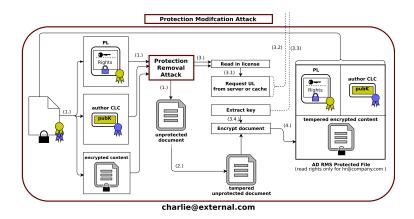


Figure 6: DisARMS Attack #2 - Modifying a protected Word-File

CLC, etc.), we expected to be hindered modifying the content by a signature. This was not the case. Even more surprisingly, Microsoft's RMS library allowed us to use the content key to encrypt data, although we only had the *view* right. The encryption method of the RMS library can only be called with a key identifier, we never get access to the raw key bytes so that we implement our own encryption method. We therefore expected the RMS library to verify our access rights, but this is not the case.

## 6 Attacking Azure RMS

On a technical level, both, the process of opening an AD RMS protected document and the process of opening an Azure RMS protected document, look very similar. An end-user will probably never see a difference on the client side. We therefore thought it would be very likely that our attacks also work with Azure RMS. Our test setup was comparable to the previous attacks. We set up two client machines. With the first user (Alice), we create the protected document and the second user (Charlie) consumes it. We rented an Azure AD instance and activated the Azure RMS role. Both users had an account at the Azure AD. Analogous to our attacks on AD RMS, we gave Charlie the view access right. We then executed the Protection Removal Attack and it worked out of the box, thanks to Microsoft efforts abstracting the RMS API functions, from the underlying workflow. The Protection Manipulation Attack also worked without any problems.

All attacks, regardless whether they targeted on the AD RMS or Azure RMS, used Microsoft Office 2013, as a client software. With the recent version *Office 365*, it is even possible to use the Azure RMS feature without administrating an Azure AD and we therefore investigated also this implementation.

## 7 Attacking Office 365

Office 365 is the latest version of Microsoft's office software. Most of the available features are implemented in Azure instances. One of these features is the Azure RMS module and available in the Enterprise version of Office 365. The workflow of accessing a protected office document is as follows: (1.) Both users are logged in with their Azure account, for example alice@rmstest.onmicrosoft.com. (2.) Alice creates a document and wants to share it with bob@rmstest. onmicrosoft.com. (3.) An email is automatically generated by Microsoft Outlook and the document is attached to it. (4.) Once Bob opens the attachment, Office 365 is used to display the document. It decrypts it and enforces the *view* only right. We found out that the Office 365 RMS integration works comparable to Office 2013 with Azure RMS. The main difference is that instead of sharing the documents via network drives or cloud storage, the documents are attached to an email and sent to the recipient. Therefore, we were able to launch both attacks again successfully on view only right protected documents.

## 8 Countermeasure

In this section, we discuss countermeasures to prevent our attacks, beginning with the modification attacks for didactic reasons.

Preventing the Modification Attack. We propose to sign the encrypted content of the RMS protected file. Although an infrastructure for asymmetric cryptographic operations is already deployed in the current RMS system, it is not that easy to apply. In RMS, the author signs the PL with his CLC private key (cf. Figure 2). He can thus use this key to sign the encrypted content. The problem appears, if another permitted editor modifies the content: since only the author's CLC is contained in the protected file, we have to add the CLC public part of the last editor of the document to the file as well. Then, this CLC can be used to verify the signature of the encrypted content by other users. For efficiency reasons, only the signature and public CLC part of the last editor should be stored in the file. As soon as a user opens a signed protected document with his client software (e.g. Word), it needs to verify that the last editor was allowed to modify the document. Therefore it sends a validation request to the RMS server. The request contains the last editors CLC, the users own RAC (public part) and the PL of the document. This way, the request is similar to the workflow used to acquire a UL as shown in Figure 3 (Messages 1 and 3). We need the approach of also sending the users own RAC to prevent information leakage, for instance, who has access to a protected document. This is reasoned by access rights in AD RMS and Azure RMS being assigned to groups. Otherwise, an unauthorized user could enumerate existing users in a group that are allowed to modify the document. By adding the RAC the server can verify, whether the user is allowed to request the access rights.

In summary, our proposed countermeasure needs to (1.) add the last editors CLC and the (2.) signature of the encrypted content to the protected file. (3.) Additionally, the client software (e.g., Word) has to verify the signature once the file is opened. (4.) The client software verifies whether the modification by the last editor for this document was legit by sending the editors CLC, his own RAC, and the PL to the RMS server. Finally, the client evaluates the response.

**Preventing the Protection Removal Attack.** The prevention of our first attack is much more complex. Since a person who is allowed to *view* the file needs somehow access to the decrypted content, the attack is always possible. Thus, we only describe methods to reduce the attack surface and make the attack much more difficult to apply in practice.

- Microsoft stated out in their How-to use guide of the RMS SDK that an application that should be used in a production environment must be signed via a Production License Agreement [34]. In reality this was not necessary and never checked by the AD RMS nor Azure RMS. So the first step should be that Microsoft verifies that the request is made by a valid (signed) client application.
- The previous verification method could also be implemented on the client side but would require some sort of trusted computing base, which ensures that only signed applications can access the RMS API. For example, on mobile platforms such as Android this is available via ARMs TrustZone [3], nevertheless it has its limitations [6].
- Microsoft states out in their RMS API documentation, that a user with rights for a document never can call a function to get direct access to the decrypted content key. Instead a key handle is returned when the function IpcGetKey is called [22]. The key handle also includes the access rights for the use, which calls the function. A simple countermeasure would be, to prevent the access to the IpcEncrypt function, if a user just has the view right. This would eliminate our second attack. [22].

**Microsoft Response (MSRC Case 33210).** We responsibly disclosed our results to different people at Microsoft and its official security response center. They responded to our findings as follows:

From: Microsoft Security Response Center <secure@microsoft.com>

"... The type of attack you present falls in the category of policy enforcement limitations. Policy enforcement capabilities, such as the ability to prevent printing or modifying content to which the user has legitimate access, are not guaranteed by cryptography or other hard technical means ..."

While this is correct for the Protection Removal Attack, the Modification Attack can be prevented by using digital signatures on the content as described in this section.

# 9 Related Work

Prior to our work different analyses related to the topic of this paper were conducted. They can be categorized as follows: (1.) theoretic and practical research on AD and ERM, (2.) risk assessment of cloud computing in general and specifically of Azure and (3.) analyses of secure cloud storage implementations.

Yu and Chiueh proposed a Display-Only File Server [51]. They described requirements to ERM systems and summarized three specific ERM implementations: (1.) Microsoft RMS, (2.) Liquid Machines which was afterwards bought by Check Point [44] and (3.) Authentica's PageRecall which was afterwards bought by EMC [45]. They argued that the effectiveness of an ERM system is based on the security of the client software enforcing the permissions. They assumed that once a protected document is opened by a permitted attacker the content can be extracted since the protection is processed on the client. They finally proposed an ERM system shifting all processes to the trusted server. The trusted server processes document modification requests and responses with the display image of the information. Thus plaintext data cannot be revealed at the client.

Schrittwieser et. al. [46] describe techniques and guidelines how to analyze RMS and Adobe LiveCycle Rights Management from the viewpoint of a digital forensic scientist. They state, that the protection of RMS on a client computer is vulnerable, in the case an attacker has access to the computer's memory.

Tajadod, Batten and Govinda compared confidentiality, integrity and availability of Microsoft Azure and Amazon Web Services (AWS) [9]. They did not analyze ERM functionalities of the cloud services but evaluated the mechanisms which enable the mentioned security goals on storage level like encryption. They concluded that both Microsoft Azure and AWS fulfill their predefined expectations.

Borgmann et al. examined the security mechanisms of several cloud storage services [39]. They gave an overview on requirements of users, legal regulations regarding security, and state-of-the-art technical security measurements.

The first implementation of Microsoft's DRMv1 was broken some short time after it's release in 1999 by an anonymous person. Another attack on DRMv2 was published by a person under the pseudonym Beale Screamer on the usenet group sci.crypt [4].

## 10 Conclusion

In this paper, we analyzed Microsoft RMS and identified two novel attacks. Our attacks can remove the whole protection of a file or modify the content without breaking its integrity, thus creating arbitrary files in the name of a different user. Microsoft RMS is widely used in enterprise systems such as banks and HSMs. Since Microsoft also includes it in its Azure platform, our attacks are even more severe. We only applied the attacks on office documents, but the concept works on arbitrary files. The only difference is the structure of the protected file, for example, the place where the licences are stored.

Our analysis of the recent RMS implementation also revealed that the security system behind RMS (with some minor updates) is used under different names since 2001 [4, 38, 2, 10]. These systems are most likely still deployed and potentially vulnerable as well. Apart from Microsoft RMS, it is an open question if our attacks can be ported to other ERM systems as well.

The attacks presented in this paper show that protecting a company's assets remains an open problem. Since an employee who is allowed to *view* a file must be able to decrypt it, additional security layers must be added to prevent leaking the decrypted content. If layers such as code signing or trusted hardware are enough remains an open question.

# Acknowledgements

The research was supported by the *German Ministry of research and Education (BMBF)* as part of the VERTRAG research project.

## References

- et al., A.A.E.K.: Organization based access control. In: Policies for Distributed Systems and Networks. LAAS CRNS (2003)
- [2] Amol Bhandarkar: Microsoft Rights Management Solutions for the Enterprise: Persistent Policy Expression and Enforcement for Digital Information (Online: 02 05 2016 2009), https://blogs.technet.microsoft.com/ amolrb/2009/05/18/how-rms-works/
- [3] ARM: TrustZone (Online: 12.05.2016), http: //www.arm.com/products/processors/ technologies/trustzone/index.php
- [4] Beale Screamer: Microsoft's Digital Rights Management Scheme Technical Details (Online: 02 05 2016 2001), http://cryptome.org/ms-drm. htm
- [5] Dan Plastina: Office 365 Information Protection using Azure Rights Management (Online: 02 05 2016 2013), https://blogs.msdn.microsoft.com/rms/2013/11/19/office-

365-information-protection-usingazure-rights-management/

- [6] Di Shen: Exploiting Trustzone on Android (Online: 12.05.2016), https://www. blackhat.com/docs/us-15/materials/us-15-Shen-Attacking-Your-Trusted-Core-Exploiting-Trustzone-On-Android-wp.pdf
- [7] EMC Corporation: Documentum Information Rights Management (Online: 12.05.2016), http://www.emc.com/enterprisecontent-management/information-rightsmanagement.htm
- [8] Firstpost: Bank Of India Deploys Microsoft Active Directory Rights Mgmt Services (Online: 04.05.2016), http://www.firstpost. com/business/biztech/bank-of-indiadeploys-microsoft-active-directoryrights-mgmt-services-1889001.html
- [9] Golnoosh Tajadod, Lynn Batten, K.: Microsoft and amazon: A comparison of approaches to cloud security. In: 4th International Conference on Cloud Computing Technology and Science. pp. 539–544. IEEE (2012)
- [10] Hauser, T., Wenz, C.: Drm under attack: weaknesses in existing systems. In: Digital Rights Management, pp. 206–223. Springer (2003)
- [11] Liu, Q., Safavi-Naini, R., Sheppard, N.P.: Digital rights management for content distribution. In: Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21. pp. 49–58. ACSW Frontiers '03, Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2003), http://dl. acm.org/citation.cfm?id=827987.827994
- [12] Microsoft: Accessing Protected Information Example (Online: 12.05.2016), https://msdn.microsoft.com/enus/library/cc243192.aspx
- [13] Microsoft: Active Directory Rights Management Services Overview (Online: 14. 01. 2016), https://technet.microsoft.com/enus/library/hh831364.aspx
- [14] Microsoft: Ad rms cryptographic modes (Online: 02. 05. 2016), https://technet.microsoft. com/en-us/library/hh867439(v=ws.10) .aspx

- [15] Microsoft: AD RMS Rights Translation: Policy Template Rights vs Office Rights vs Share-Point Permissions (Online: 13. 02. 2016), http://social.technet.microsoft.com/ wiki/contents/articles/13163.ad-rmsrights-translation-policy-templaterights-vs-office-rights-vs-sharepointpermissions.aspx
- [16] Microsoft: AD RMS under the hood: Client bootstrapping (Online: 20. 01. 2016), http: //blogs.technet.com/b/rms/archive/ 2012/08/17/ad-rms-under-the-hoodclient-bootstrapping-part-1-of-2.aspx
- [17] Microsoft: Appendix B: Product Behavior (Online: 12.05.2016), https://msdn.microsoft. com/en-us/library/cc243421.aspx
- [18] Microsoft: Client devices that support Azure RMS (Online: 05. 03. 2016), https: //technet.microsoft.com/en-us/library/ dn655136.aspx#BKMK\_SupportedDevices
- [19] Microsoft: Client Licensor Certificate Example (Online: 12.05.2016), https: //msdn.microsoft.com/en-us/library/ ff842490.aspx
- [20] Microsoft: Encrypted Rights Data Example (Online: 12.05.2016), https://msdn.microsoft. com/en-us/library/ff958983.aspx
- [21] Microsoft: How does Azure RMS work? (Online: 05. 03. 2016), https://technet.microsoft. com/en-us/library/jj585026.aspx#BKMK\_ HowRMSworks
- [22] Microsoft: IpcGetKey function (Online: 12.05.2016), https://msdn.microsoft. com/en-us/library/windows/desktop/ hh535263(v=vs.85).aspx
- [23] Microsoft: Ipcnotepad sample (Online: 02. 05. 2016), https://code.msdn.microsoft.com/ IPCNotepad-Sample-f67dae80
- [24] Microsoft: Ms rms encryption algorithms (Online: 02. 05. 2016), https://blogs.msdn. microsoft.com/rms/2015/03/09/aes-256symmetric-key-encryption/
- [25] Microsoft: Ms rms sdk 2.1 api reference (Online: 02. 05. 2016), https://msdn.microsoft.com/ en-us/library/hh535292(v=vs.85).aspx
- [26] Microsoft: Publishing License Example (Online: 12.05.2016), https://msdn.microsoft. com/en-us/library/ff842510.aspx

- [27] Microsoft: Publishing Usage Policy Example (Online: 12.05.2016), https://msdn.microsoft. com/en-us/library/cc243349.aspx
- [28] Microsoft: Rights Management Services (RMS): Client-to-Server Protocol (Online: 12.05.2016), https://msdn.microsoft.com/enus/library/cc243191.aspx
- [29] Microsoft: RMS Account Certificate Example (Online: 12.05.2016), https://msdn.microsoft. com/en-us/library/ff842475.aspx
- [30] Microsoft: RMS API Function Reference (Online: 12.05.2016), https: //docs.microsoft.com/en-us/rightsmanagement/sdk/2.1/api/win/functions
- [31] Microsoft: Security Processor Certificate Example (Online: 12.05.2016), https://msdn.microsoft.com/enus/library/ff842500.aspx
- [32] Microsoft: Server Roles and Technologies in Windows Server 2012 R2 and Windows Server 2012 (Online: 12.05.2016), https://technet.microsoft.com/enus/library/hh831669.aspx
- [33] Microsoft: Setting up the pre-production development environment (Online: 12.05.2016), https: //docs.microsoft.com/en-us/rightsmanagement/develop/how-to-set-up-thepre-production-development-environment
- [34] Microsoft: Switching to the production environment (Online: 12.05.2016), https: //docs.microsoft.com/en-us/rightsmanagement/develop/switching-to-theproduction-environment
- [35] Microsoft: Technical overview for the microsoft rights management sharing application (Online: 02. 05. 2016), https://docs.microsoft.com/enus/rights-management/rms-client/ sharing-app-admin-guide-technical
- [36] Microsoft: Use License Example (Online: 12.05.2016), https://msdn.microsoft.com/ en-us/library/ff842520.aspx
- [37] Microsoft: XrML Elements (Online: 14. 01. 2016), https://msdn.microsoft.com/en-us/ library/cc542560(v=vs.85).aspx
- [38] microsoft: Microsoft Rights Management Solutions for the Enterprise: Persistent Policy Expression and Enforcement for

Digital Information (Online: 02 05 2016 2003). https://web.archive.org/web/ 20030618093832/http://www.microsoft. com/windowsserver2003/docs/RMS.doc

- [39] Moritz Borgmann, Tobias Hahn, M.H.T.K.M.R.U.V.S.V.: On the security of cloud storage services. Fraunhofer Institute for Secure Information Technology SIT (2012)
- [40] Official Microsoft RMS Team Blog: Accessing protected content offline and conpre-licensing (Online: 12.05.2016), tent https://blogs.technet.microsoft.com/ rms/2012/04/16/accessing-protectedcontent-offline-and-content-prelicensing/
- [41] OpenText: Open Text Announces Rights Management Services for the Open Text ECM Suite (Online: 04.05.2016), https://www.opentext. com/who-we-are/press-releases?id=2358
- [42] Redmond Magazine: Microsoft Rights Management Service Now Available (Online: 12.05.2016), https://redmondmag.com/articles/2013/ 11/18/rms-now-available.aspx
- [43] Reid, J.F., Caelli, W.J.: Drm, trusted computing and operating system architecture. In: Proceedings of the 2005 Australasian Workshop on Grid Computing and e-Research - Volume 44. pp. 127-136. ACSW Frontiers '05, Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2005), http://dl.acm.org/citation. cfm?id=1082290.1082308
- [44] Check Point Software Technologies Ltd.: Check point capsule docs document control, https://supportcenter.checkpoint. com/supportcenter/portal?eventSubmit\_ doGoviewsolutiondetails=&solutionid= sk106568
- [45] EMC Corporation: Emc documentum information rights management services, http://www.emc.com/enterprisecontent-management/information-rightsmanagement.htm
- [46] Schrittwieser, S., Kieseberg, P., Weippl, E.: Digital forensics for enterprise rights management systems. In: Proceedings of the 14th International Conference on Information Integration and Webbased Applications & Services. pp. 111-120. ACM (2012)

- [47] Sentinelcloud: SafeNet HSMs for Microsoft Rights Management Services (Online: 04.05.2016). http://sentinelcloud.com/WorkArea/ DownloadAsset.aspx?id=8589938599
- [48] Stamp, M.: Digital rights management: The technology behind the hype. J. Electron. Commerce Res. 4(3), 102–112 (2003)
- [49] TechNet: What is Azure Rights Management? (Online: 15. 02. 2016). https://technet.microsoft.com/enus/library/jj585026.aspx
- [50] v3.co.uk: Ministry of Defence CIO introduces Defence-as-a-Platform with the help of Microsoft (Online: 04.05.2016), http://www.v3.co. uk/v3-uk/news/2434336/ministry-ofdefence-cio-introduces-defence-as-aplatform-with-the-help-of-microsoft
- [51] Yang Yu, T.c.C.: Enterprise digital rights management: Solutions against information theft by insiders. In: Information Management & Computer Security (2007)

#### **Code Examples** Α

7

8

11

13

14

16

17

18

19

21

22

24

25

26

27

29

31

```
1
2
    System::Void readLicense(Stream ^encryptedStream)
3
4
            byte staticLicensePrefix[3] = {0xEF, 0xBB, 0xBF};
5
            [...]
6
            // transform and copy license structure to RMS
                  API format
            bytesLength = gcnew array<Byte>(sizeof(DWORD));
            bytesRead = encryptedStream->Read(bytesLength, 0,
                  bytesLength->Length);
9
            Marshal::Copy(bytesLength, 0, (IntPtr)(&
                  m_pLicense->cbBuffer), sizeof(DWORD));
10
            bytesLicense = gcnew array<Byte>(m_pLicense->
                 cbBuffer);
12
            bytesRead = encryptedStream->Read(bytesLicense,
                  0, bytesLicense->Length);
            m_pLicense->pvBuffer = (LPVOID)new unsigned char[
                  bytesLicense->Length];
15
            Marshal::Copy(bytesLicense, 0, (IntPtr)(
                  m_pLicense->pvBuffer), m_pLicense->cbBuffer)
    7
    MemoryStream<sup>^</sup> initDecryption(Stream <sup>^</sup>encryptedStream)
            [...]
20
            readLicense(encryptedStream);
23
            // get content key from read in license
            pKey = &m_key;
            hr = IpcGetKey(m_pLicense,
                   0.
                   nullptr,
28
                   nullptr,
                   pKey);
30
```

32 pin\_ptr<PIPC\_TEMPLATE\_INFO> ppTemplateInfo; 33 ppTemplateInfo = &m\_pTemplateInfo; hr = IpcGetSerializedLicenseProperty(m\_pLicense, 34 35 IPC\_LI\_DESCRIPTOR, 36 m\_key, 37 0. 38 (LPVOID \*)ppTemplateInfo); result = decryptEncryptedContent(encryptedStream) 39 40 [...] 41 } 42 43 MemoryStream<sup>^</sup> decryptEncryptedContent(Stream <sup>^</sup> encryptedStream) 44 { 45 46 [...] cbReadRemaining = ... // // get size of encrypted 47 content while (cbReadRemaining > \*pcbBlockSize) 48 // pcbBlockSize = AES block size 49 50 { cbRead = 0; 51 encryptedStream->Read(readBuffer, 0, \* 52 pcbBlockSize); 53 hr = IpcDecrypt(m\_key, 54 55 cBlock, false, 56 pbReadBuffer, 57 \*pcbBlockSize, 58 pbWriteBuffer, 59 cbOutputBuffer, 60 &cbRead); 61 cBlock++; 62 // write the decrypted content to new document file 63 decryptedStream->Write(writeBuffer, 0, cbRead); 64 cbReadRemaining -= cbRead; 65 } // decrypt the last encrypted content block. 66 67 cbRead = 0; 68 encryptedStream->Read(readBuffer, 0, cbReadRemaining); 69 hr = IpcDecrypt(m\_key, 70 cBlock, 71 true, 72 pbReadBuffer, 73 74 cbReadRemaining, pbWriteBuffer, 75 cbOutputBuffer, 76 &cbRead); 77 decryptedStream->Write(writeBuffer, 0, cbRead); 78 [...] 79 3

Listing 1: Source code of Attack 2/2.

```
MemoryStream encryptModifiedContent(Stream fieldStream)
1
2
    ſ
3
4
            [...]
5
            cbReadRemaining = ... // get size of modified
                  document content
6
            // pcbBlockSize = AES block size
7
            while (cbReadRemaining > *pcbBlockSize)
8
            ſ
                    cbRead = 0;
9
                    modifiedStream->Read(readBuffer, 0, *
10
                         pcbBlockSize);
11
                    hr = IpcEncrypt(m_key,
12
                           cBlock,
13
                           false,
14
                           pbReadBuffer,
                            *pcbBlockSize,
15
16
                           pbWriteBuffer,
17
                           cbOutputBuffer,
18
                           &cbRead);
```

```
cBlock++;
               encryptedStream->Write(writeBuffer, 0,
                    cbRead);
               cbReadRemaining -= cbRead;
       }
       // encryption of the last modified content block.
       cbRead = 0;
       modifiedStream->Read(readBuffer, 0,
             cbReadRemaining):
       hr = IpcEncrypt(m_key,
              cBlock,
              true,
               pbReadBuffer.
               cbReadRemaining,
               pbWriteBuffer,
               cbOutputBuffer,
              &cbRead);
       [...]
7
```

19

20

21

22

23

24

25

26

27

28

29

30

31 32

33

34

35

36

37

Listing 2: Source code of Attack 2/2.