

DATA HIDING AND DETECTION IN OFFICE OPEN XML (OOXML) DOCUMENTS

by

Muhammad Ali Raffay

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Applied Science (MAsc)

in

Electrical and Computer Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology

Oshawa, Ontario, Canada

March, 2011

Copyright ©Muhammad Ali Raffay, 2011

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Muhammad Ali Raffay

Abstract

With the rapid development and popularity of information technology, criminals and mischievous computer users are given avenues to commit crimes and malicious activities. One of the commonly used tactics, called steganography, is to hide information under a cover media so that except participants, no one else knows the existence of such information. Many techniques have been proposed for hiding data in images, videos and audios, but there is not much research devoted to data hiding in the popular MS Office documents which have recently adopted Office Open XML (OOXML) format.

In this research, we first focus on identifying several data hiding techniques for OOXML documents. Then, we design and develop a fast detection algorithm based on the unique internal structure of OOXML documents, which contains multiple XML files, by using multi-XML query technique. Experimental results show the proposed detection algorithm outperforms the traditional one in terms of detection speed and completeness, where performance is the key to success of detecting hidden data in OOXML documents due to the fact that millions of documents are generated and transferred over the internet every day.

Acknowledgements

I am heartily thankful to my supervisor, Dr. Xiaodong Lin, for his continuous support and interesting discussions from the initial to the final level enabled me to develop an understanding of the subject. I would also like to express my sincere appreciation to my co-supervisor Dr. Ali Grami for his patience and kind guidance throughout this study. I owe many thanks to both for their challenging questions and simplifying explanations. I am sure that I become stronger because of them.

I am extremely thankful to my mother for many things especially for being herself. This dissertation is dedicated to her.

Contents

Abstract...	iii
Acknowledgements.....	iv
List of Table	viii
List of Figures	ix
List of Acronyms	xi
Chapter 1 – Introduction	1
1.1 – Research Motivation	3
1.2 – Research Objectives & Contributions	5
1.3 – Outline of the thesis	7
Chapter 2 – Literature Review	8
2.1 – History of Steganography	8
2.2 – Present Steganography.....	10
Chapter 3 – Data Hiding Techniques in OOXML Documents.....	15
3.1 – Introduction.....	15
3.2 –OOXML File Format & MS Office 2007	16
3.2.1 – Container – Package	17
3.2.2 – Package Parts	19
3.2.3 – Relationships.....	20
3.3 – Data Hiding in OOXML – MS Word 2007 Documents.....	22

3.3.1 – Data Hiding Using OOXML Relationship Structure.....	22
3.3.2 – Data Hiding Using XML Format Feature.....	27
3.3.3 – Data Hiding Using XML Format Feature & OOXML Relationship Structure.....	30
3.3.4 – Data Hiding Using OOXML Flexibility for Embedded Resource Architecture	37
3.3.5 – Data Hiding Using OOXML Flexibility of Swapping Parts	43
Chapter 4 – Hidden Data Detection in OOXML Documents	48
4.1 – Introduction.....	48
4.2 – Document Inspection – Detection Feature of MS Office 2007	50
4.3 – Detection Logic of Hidden Data.....	52
4.3.1 – Detecting Hidden Data using OOXML Relationship Structure	54
4.3.2 – Detecting Hidden Data using XML Format Feature & OOXML Relationship Structure	55
4.3.3 – Detecting Hidden Data using OOXML Flexibility for Embedded Resource Architecture	56
4.3.4 – Detecting Hidden Data using OOXML File Architecture Flexibility of Swapping Parts	57
4.4 – OOMXQA	59
4.4.1 – OOMXQA Algorithm.....	60
4.4.2 – OOMXQA Steps for OOXML Document Format	61
4.5 – Performance Evaluation.....	67

4.5.1 – Data Sets	69
4.6 – Conclusion	69
 Chapter 5 – Conclusions & Future Work	 71
5.1 – Conclusions.....	71
5.2 – Summary	73
5.3 – Future Research	74
 References	 76
 Appendix ‘A’ – DTD & XML Document Tree, Path Language & XML Functional Dependency	 79
Appendix ‘B’ – Document Type Definition (DTD)	81
Appendix ‘C’ – Global Materialized View (GMV)	84
Appendix ‘D’ – OOMXQA XQuery Code	89
Appendix ‘E’ – Conventional Algorithm XQuery Code	90
Appendix ‘F’ – Global Materialized View (GMV) XQuery Code	91

List of Table

Table 4.5	OOMXQA Performance Comparison with Conventional Algorithm	68
-----------	---	----

List of Figures

Figure 2.1	5x5 Tap Code used by Armed Forces in Vietnam	10
Figure 3.1	Internal Directory Structure of an OOXML Document	18
Figure 3.2	Logical Structured Organization of Package Parts	19
Figure 3.3	Logical Representation of OOXML Document Relationships	21
Figure 3.4	MS Office 2007 – Raised an Error “Problems with the Contents”	23
Figure 3.5	MS Office 2007 – Option to Recover the Document	23
Figure 3.6	Modified [Content_Type].xml File	25
Figure 3.7	Modified Relationship File (.rels)	26
Figure 3.8	Added Image Shown in OOXML document	32
Figure 3.9	Beginning Section of Main Document File	33
Figure 3.10	End Section of Main Document File	33
Figure 3.11	Metadata of First Image Inserted Using MS Word 2007	34
Figure 3.12	Sample Metadata of Hidden Image	35
Figure 3.13	Relationship File Showing Entry for Hidden Image	36
Figure 3.14	Document Structure With Custom Xml Data and its Part Relationship Code	39
Figure 3.15	Part Relationship File Code (document.xml.rels)	40
Figure 3.17	Sample code of “hiddendata.xml” file	41
Figure 3.18	Sample Code of customXml part relationship file	42
Figure 3.19	Inspect Document Feature of MS Office 2007	43
Figure 3.20	Swapping of Image between two OOXML Documents	46
Figure 4.1	Package Layout of MS Word 2007 Document	50

Figure 4.2	Preview of Associated Parts Using Relationship Information with Main Document File	50
Figure 4.3	XML Preview of Main Document File	53
Figure 4.4	XML Preview of Part Relationship File	53
Figure 4.5	MS Word Document Output	53
Figure 4.6	Unknown Relationship & Unknown Part	54
Figure 4.7	Detection Logic for Unknown Parts & Unknown Relationships	55
Figure 4.8	Detection Logic for Ignorable Attribute Technique	56
Figure 4.9	Detection Logic for CustomXml Technique	57
Figure 4.10	Detection Logic for Image Steganography Technique	58
Figure 4.11	Office 2007 Document Layout	59
Figure 4.12	MS Office document with main folders and files along with data	59
Figure 4.13	Package level & Part level Relationship Files with Elements and Attributes	60
Figure 4.14	Main Document File Snapshot	60
Figure 4.15	DTD of Relationship Files	62
Figure 4.16	DTD of Main Document File	62
Figure 4.17	GMV's DTD for an OOXML Documet with Elements & Attributes	65
Figure 4.18	GMV's DTD for an OOXML Document in Tree Form	65
Figure 4.19	XQuery Result – Hidden files inside MS Word 2007 document dataset	67
Figure 4.20	OOMXQA Performance Evaluation with Conventional Algorithm	68

List of Acronyms

.docx	.doc extended (file extension) defaults to MS Word 2007
.rels	Relationship (file extension)
AES	Advanced Encryption Standard
Aka	Also known as
ASCII	American Standard Code for Information Interchange
ASTM	American Society for Testing and Materials
BMP	Bitmap (file name extension)
CCR	Continuity of Care Record
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
DTD	Document Type Definition
ECMA	Standardization of Information and Communication Technology (ICT) and Consumer Electronics (CE)
Emf	Enhanced Windows Metafile (filename extension)
Emu	English Measurement Unit
FD	Functional Dependency
GB	Gigabyte
GIF	Graphic Interchange Format (file extension)
Gif	Graphic Interchange Format (file extension)
GMV	Global Materialized View
HTML	HyperText Markup Language
HTTP	Hyper Text Transmission Protocol
ID	Identification
JPEG	Joint Photographic Experts Group
LSB	Least Significant Bit
LSB	Least Significant Bit
MIME	Multipurpose Internet Mail Extensions
ML	Markup Language
MP3	Moving Picture Experts Group Layer-3 Audio
MPEG	Moving Picture Experts Group
MS Office	Microsoft Office
ODF	Open Document Format
OLE	Object Linking & Embedding
OOMXQA	Office Open Multi-XML Query Algorithm
OOXML	Office Open XML
PC	Personal Computer
PDA	Personal Digital Assistant

PDF	Portable Document Format
PDF/H	Portable Document Format – Healthcare
PHR	Personal Health Record
Png	Portable Network Graphics (graphic file standard/extension)
RAM	Readable Memory
RGB	Red Green Blue
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman (Algorithm)
SMXQA	Semantic Based Multi-XML Query Algorithm
TC	Technical Committee
TV	Television
URI	Uniform Resource Identifier
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
ZIP	Compressed File (file extension)

Chapter 1

Introduction

The advent of information technology has brought us with more convenience and comfort. Millions of files and documents are transferred daily over the Internet. For example, we can easily pay bills, transfer money, and view paystubs and cheques all online. However, criminals and mischievous computer users are also given new avenues to commit crimes and malicious activities. One of the commonly used tactics is to hide information under a cover media, such as image, so that except participants, no one knows the existence of such information, also known as steganography. For example, a child pornography image can be hidden inside another image file or audio file or any other file format, which could look perfectly legitimate. Therefore, it is crucial to uncover these activities.

Steganography (pronounced STEHG-uh-NAH-gruhf-ee, from Greek *steganos*, or "covered," and *graphie*, or "writing") is the hiding of a secret message within an ordinary message and only the sender and receiver know of its existence and method of access [1]. New digital steganographic techniques in which messages are hidden into text, image and video files raise new challenges and require detection of steganographically encoded packages, is called steganalysis [2].

Steganography also known as steg or stego, poses a major challenge to law enforcement officers. One of the most common illicit uses is for the possession and storage of child pornography images. However steganography can also be used to commit fraud, terrorist activities and other illegal acts [3]. Often these hidden files are also encrypted, adding

another layer of security to impede investigators [4]. Currently there are more than 300 publicly available steganography encoding programs employing many different encryption algorithms which result in precluding any universal test for steganography [5].

Steganography made news headlines when the US Department of Justice charged 11 individuals in two separate criminal complaints with conspiring to act as unlawful agents of the Russian Federation within the United States. The defendants allegedly used steganography to embed messages in more than 100 image files posted on public websites [6].

Cryptography is another technique of secret writing and allows anyone to see the message, but nobody else can read it. This is because its letters have been rearranged, or replaced by different letters by using some scheme only known by the sender and receiver. Cryptography is being outlawed by many countries including US, European and Asian countries in important areas such as banks and financial institutions (notably brokers, credit card companies and securities firms) [7]. As a result, it is easy to attract the attention of law enforcement when any other encrypted network traffic or documents are seen online due to the fact that encrypted data looks like random data and can be easily distinguished [8]. Then, law enforcement can decrypt suspicious messages, especially because of the rapidly increasing computing power, such as quantum computing [9]. In this scenario, steganographic techniques become more attractive for mischievous users to transmit their messages and turn into serious problem for us, and needs to fight against.

Steganography has attracted a lot of attention in the recent years, and most steganography work has been performed on images, video clips, text, music and sound. Until recently,

the switch from proprietary formats to XML format by applications vendors, such as Microsoft, Sun Microsystems and other developers for document files raised new concerns [10]. For example, a newly developed file format introduced by Microsoft named Office Open XML (OOXML) is a zipped, XML based file format for representing spreadsheets, charts, presentations and word processing documents [11]. It has been adopted in Microsoft Office 2007/2010. While it offers greater benefits over its predecessor, the proprietary binary file formats seen in previous versions of Microsoft Office, its unique internal file structure also opens the door to many new steganographic methods for data hiding in Microsoft Office documents based on OOXML. Furthermore, due to the popularity of Microsoft Office documents such as word or excel files, they have become a strong preference for mischievous users to use them as cover data for hiding information due to the fact that these document files could be easily ignored [12].

There is currently no available tools for law enforcement officers to find hidden information using various steganographic techniques, especially on these newly emerging cover media formats like OOXML. Active research is being carried out to further improve their adequacy. The work to standardize OpenXML has been carried out by Ecma International via its Technical Committee 45 (TC45), which includes representatives from Apple, Barclays Capital, BP, The British Library, Essilor, Intel, Microsoft, NextPage, Novell, Statoil, Toshiba, and the United States Library of Congress [13].

1.1 Research Motivation

With the growth of Internet based technologies the XML is playing a vital role in data sharing, especially in the financial sector. Applications which use XML allow greater

flexibility in terms of data integration and cross platform applications. XML is also adapted and used by the software vendors and there are a number of XML based applications available on the market [10]. Now XML is also being used for document representation. For example, MS Office 2007 and Office Open from Sun Microsystems are few applications using XML file format known as OOXML and Open Document Format (ODF) respectively.

In 2006, Microsoft brings in OOXML file format for its MS Office 2007 product. It brought a lot of advantages including compression of document, thus leading to less disk space utilization [13, 14]. Also, Microsoft Office documents becomes defacto standard such as word or excel files, and can be used on a wide range of hardware that runs Microsoft Office products including PC's, mobile phones and PDAs. However that in turn provided many new ways to hide data in OOXML documents. The widespread of these documents and the likely chance to get unnoticed turn to be a good choice for mischievous users to use them as carrier files.

Vendors are now also becoming aware of these limitations in their applications and propose features for its removal, such as provided by MS Office 2007 and known as Document Inspection. This feature inspects MS Office 2007 documents and allows removal of personal information and hidden data present in the document. Also, a warning message may pop up when a modified OOXML document doesn't satisfy the OOXML structural requirements because of the hidden content. This makes data hiding more difficult and challenging, and recently, a lot of attentions have been paid to new steganographic methods of hiding data within OOXML documents [10, 16]. These could be used by some mischievous user to transmit information masked inside a document in

plain view for malicious purposes such as coordinating terrorist attacks or distributing inappropriate materials, and it becomes critical for law enforcement officers to monitor and detect hidden data in these documents in case that may be used by some terrorists for communication over the web.

It is worth mentioning that around 7.5 million users downloaded beta version of MS Office 2010 whereas 40 million users generate documents using MS Office various versions with estimates exceeding 40 billion documents and billions more being created each year [13, 17]. This is a very challenging task to find hidden data in thousands of documents and the performance becomes key for hidden data detection algorithm to catering real time needs. Furthermore, unlike conventional steganography, which highly depends on cover media with limited applicable data hiding schemes, steganography in OOXML document allowing combination of data hiding techniques makes detection process more difficult and advance. There is a high demand for a fast and effective detection algorithm for hidden data in OOXML documents usually found in large numbers.

Data Hiding and detection in OOXML documents is under-researched, especially considering how popular they are. In this thesis, we will investigate data hiding techniques for OOXML documents. We also study the fast and efficient detection algorithm to cater real time needs for such large extent of files.

1.2 Research Objectives and Contributions

The objective of this research is to identify steganographic techniques for MS Office 2007 document which conforms to OOXML file format and as well as to design and

develop a fast and efficient algorithm to detect hidden data using presented techniques. Since these identified techniques are distinct from previous format that is binary format and new format is complex and pose many challenges. Newly introduced built in intelligent features for removing suspicious data in turn leads to application robustness and challenges data hiding attacks.

The contribution of this thesis can be classified into two parts. First, we identify several successful data hiding techniques for OOXML files. These techniques are categorized into different ways: data hiding using OOXML relationship structure, data hiding using XML format feature, data hiding using XML format feature and OOXML relationship structure, data hiding using OOXML file embedded resource architecture and data hiding using OOXML flexibility of swapping parts.

Second as per observation of data hiding techniques, we realize that the detection of hidden data requires scanning of multiple XML files as OOXML file comprises of multiple XML files zipped together. The conventional technique is to read these files one by one and thus inefficient in terms of time as thousands of documents transferred daily over the Internet. In order to overcome this issue, we designed a fast and efficient multi-XML files querying algorithm for OOXML documents by using a multi-XML query technique. The designed algorithm is enhanced and further customized for OOXML files structure. Our developed detection algorithm is using XQuery code and can be embedded with any steganalysis and detection tools available online. The XQuery provides efficient XML querying capabilities known so far.

Finally, our developed detection XQuery code is open for code reuse and permits multiple interfaces which can be applied by other researchers.

1.3 Outline of the Thesis

The organization of the remainder of the thesis is organized as follows. Chapter 2 deals with literature review of other researcher works in this area and best known so far. Chapter 3 gives an overview of OOXML file format related in terms with steganography plus also discuss approaches employed for data concealment in OOXML files with its experiment logic. Chapter 4 introduces a brief explanation of detection algorithm as well as hypothetical rules. These rules are subsequently justified with the experiments and findings depicted and also a comparison of developed tool technique with available conventional detection technique is presented. The performance analysis graph shows the time efficiency of proposed detection technique over conventional technique. Finally Chapter 5 gives a conclusion and proposes some future research directions.

Chapter 2

Literature Review

This chapter is devoted to illustrating how outside research in the field of digital steganography has affected this research.

2.1 History of Steganography

Throughout history steganography has been used to secretly communicate information between people. Some examples of use of steganography in past times are [18]:

1. In Ancient Greece they used to select messengers and shave their head, they would then write a message on their head. Once the message had been written the hair was allowed to grow back. After the hair grew back the messenger was sent to deliver the message, the recipient would shave off the messengers hair to see the secret message.
2. Another method used in Greece was where someone would peel wax off a tablet that was covered in wax, write a message underneath the wax then re-apply the wax. The recipient of the message would simply remove the wax from the tablet to view the message.
3. During World War II invisible ink was used to write information on pieces of paper so that the paper appeared to the average person as just being blank pieces of paper. Liquids such as urine, milk, vinegar and fruit juices were used, because when each one of these substances is heated they darken and become visible to the human eye.

4. In more recent history, several steganographic methods were used during World War II. Nazis developed microdots microfilm chips are the size of periods on a standard typewriter. These dots contain pages of information, drawing etc [3]. The Nazis also employed invisible ink and null ciphers. One of the most noted null cipher messages sent by a Nazi spy follows:

“Apparently neutral’s protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils.

Using the second letter from each word, the following message appears:

“Pershing sails from NY June I”

5. Finally during the Vietnam era, there were instances where captured members of the U.S. Armed Forces would use various hand gestures during photo ops, often only to have these gestures airbrushed out by the media. Other techniques employed were using the eyelids to blink word in Morse code (such as torture).

The code was based on a five by five matrix with each letter being assigned a tap sequence based on this matrix. Spaces (pauses) between characters were twice as long as the spaces in that letters code [3].

	1	2	3	4	5
1	A . .	B . ..	C,K	D	E
2	F .. .	G	H	I	J
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Fig. 2.1 : 5x5 tap code used by Armed Forces in Vietnam

2.2 Present Steganography

With the boost in computer power, the internet and with the development of digital signal processing (DSP), steganography has gone digital which adds terms like “mp3”, “jpeg”, “mpeg”, and “document” files into our everyday vocabulary. Commonly these are the number of digital technologies that the community is concerned with, namely text files, still images, audio or video and documents.

Today, steganography is researched both for legal and illegal reasons. Steganography would provide an ultimate guarantee of authentication that no other security tool may ensure. For example a digital watermark controls copyright of material transmits over the web such as images, music, movies and TV broadcasting. Such multimedia content is compressed first to save transmission time without affecting the quality of the content. These lossy compression techniques lend themselves perfectly for hiding data in such files. Another important use of steganography is to embed data about medical images, so that there are no problems with matching patients’ records and images. It is believed that

cyber crime also benefits from this digital revolution. Hence an immediate concern was shown on the possible use of steganography by criminals.

It is beyond the scope of this thesis to go into details of steganographic methods, suffice it to say that there are two primary groups, multimedia steganography and document steganography. In multimedia contents, most of the work has been performed on images to embed data and mostly available tools support an image steganography by altering or replacing the Least Significant Bit (LSB). These tools take an advantage of compression types such as lossy and lossless to embed data inside an image. A well known JPEG format uses lossy compression technique, whereas lossless compression technique used by GIF and BMP formats [18]. These compression techniques are explained in detail in data hiding section of this thesis.

Another group is the use of steganography in documents works simply by adding white space and tabs to the ends of the lines of a document. This type of steganography is extremely effective, because the use of white space and tabs is not visible to the human eye at all, at least in most text/document editors. White space and tabs occur naturally in documents, so there isn't really any possible way using this method of steganography would cause someone to be suspicious. Almost all computer users write and exchange documents written with application using proprietary document formats such as MS Office and gains lot of attention in the past in terms of steganography. The earlier versions of MS Office documents were having compound document file format and provided limited ways of data hiding. The data hiding can be done in binary format or using white spaces, tabs, line or word shifting and semantic methods [3]. The average size of trash space available in a compound document format is 6.53% of its size and can

be used for steganography [19]. Similar data hiding techniques also applies on XML files for text steganography. From the perspective of suspects, good data hiding techniques should meet the goals of security and capacity [19]. Unfortunately, these works are limited in the same way and information hiding using these techniques is limited.

Recently, Microsoft introduced XML file format known as OOXML for its MS Office 2007 documents. This format contains several XML files and other binary files which are zipped together to form an OOXML document. The flexibility of OOXML document can be used for steganographic purposes and MS Office 2007 has a built in feature to hide text in the document. MS Office 2007 also gives feature “Document Inspection” to remove hidden information from the documents generated by the application and by the user. To the best of our knowledge, few authors have provided a formal framework for steganography in OOXML documents and proved its possibility. This proves that a very little work is being done with this new format, i.e. OOXML format.

In [15], the authors conceal data in OOXML document using unknown parts and unknown relationships. They also develop detection algorithm to detect existence of hidden data using identified technique. The detail explanation of their work is highlighted in data hiding section of this thesis. We believed that there are more ways to hide data in OOXML documents and presented few more data hiding techniques identified by us. In order to meet the goals of security and capacity, we can say that our identified data hiding techniques allow numerous options and looks genuine.

Based on the fact that large number of documents is generated every day, we strongly believed that the key factor for detection algorithm is performance time. The conventional detection algorithm fails to cater real time need as thousands of documents

transferred daily over the net and thus requires a fast and efficient detection algorithm. This motivates us to design and develop a fast and efficient detection algorithm to detect hidden data in OOXML documents in much less time to cater real time need. We compared their given detection algorithm with our detection approach and presented the comparison in terms of performance in detection of hidden data section of this thesis.

In [10], the authors discussed steganographic techniques for OOXML documents using encryption and comments feature of zip and XML files. Their work involves creation of encrypted documents containing simple texts and images. This encrypted OOXML document is stored as OLE compound file. They use popular 7-Zip utility software to read these files and the cryptanalysis reveals that OOXML document uses RSA and AES algorithms with a 128-bit key for encryption. They also presented technique of adding comments directly into zip archive using comment feature of zip file format and adding XML comments to the XML file. In both cases these comments are ignored by MS Office 2007 application and these comments are discarded when document is written back out. They further investigated that by using base64 encoding method they successfully embeds file as a comment in one of the XML files of OOXML document and MS Office 2007 ignores it silently. Their developed tool is called docx-steg.py can hides any file using this data hiding scheme. The data hiding scheme is given in data hiding section of this thesis.

Their research work also focused on insights of XML based documents regarding their forensic implications. This gives further research directions such as these formats allows interpreting an information from XML tag's to identify unauthorized tampering and

support scrutiny in court which enable law enforcement officers to use as digital evidence against suspects.

We have chosen this research because it gives a chance to study several various aspects of steganography in OOXML documents which highlights deeper format restrictions for embedding hidden data and shows techniques which makes detection difficult, and also to identify and develop a fast and effective detection algorithm for OOXML documents to cater real time need.

Chapter 3

Data Hiding Techniques in OOXML Documents

3.1 Introduction

A great deal of research has been accomplished in the area of hiding data in text, image, or audio files, but not so much on hiding data inside file structure of Office 2007 documents, which adopts OOXML file format.

As an open standard for data integration and interoperability, OOXML brings a lot of advantages. However, the open structure of OOXML, which is organized in a zip archive or package, also faces various security threats. One of the most serious security threats is hidden data issue, where information is concealed within an OOXML file.

Data hiding in Office Open XML presents a variety of challenges that arise due to the importance of conformance of the relationships within the package. In order for an OOXML Word document to be properly displayed in an OOXML file editor, for example, Microsoft Office Word, the relationships among the various files in the package have been satisfied. For example the image is stored as a separate file in a package and only its metadata information containing “Id” is stored in the main document file. When the main document is opened, its corresponding “Id” has been searched in its relationship file which contains the type and target (location) of the image. Then, the document editor fetches the image by using the relationship information and places it in the main document, making this whole process transparent from the user. Furthermore an OOXML file editor, for example MS Office 2007, has a document inspector feature. It not only removes some hidden data or private or personal information (often known as metadata)

from OOXML files, but also gives a warning if certain relationship of the files is broken. From the perspective of suspects, good data hiding techniques should meet the goals of security and capacity, in particularly unsuspecting for the existence of secret data hidden inside a cover media [19]. Obviously, successful information concealment in an OOXML document or MS Office 2007 document must satisfy the following conditions:

- The hidden data must not satisfy all the relationships in the package. Otherwise, this hidden information might be immediately visible when the document is open in an OOXML editor.
- The hidden data must be specifically inserted to avoid detection by document inspectors. Otherwise, a warning might be displayed when the document is open in an OOXML editor.
- Hidden data would not be overwritten or the possibility of data being overwritten is low and the technique can store reasonable amount of hidden data.

Despite these challenges the potential OOXML structure provides for new ways of data hiding exploitation techniques in MS Office 2007. An overview of OOXML format related in terms of steganography is presented here for better understanding.

3.2 OOXML File Format and MS Office 2007

Microsoft introduced XML into Office 2007 with full fidelity known as Office Open XML (OOXML) file format. OOXML document structure is also based on Open Packaging Conventions (OPC). The OOXML format enables that generated document will be fully compatible with other cross platform business applications. Several countries including US and few European countries have formally announced either

adoption, or the evaluation of adoption of OOXML. It means that OOXML standard is permitted to be used where national regulations do not allow proprietary formats.

OOXML file format consists of a compressed ZIP file, called package. ZIP was chosen as the package format for the Office XML formats because it is a well understood industry standard. All of the contents within the document are hold by a package. In addition to Office markup, the package can also include embedded files such as images, videos, or other documents. The ZIP compression decreases the size of the document up to 75% and is more robust to error handling [19], which allows in turn an ease of managing and repairing of individual segmented files within a package. For example, you can open MS Word 2007 document that uses OOXML format, and locates the XML part that represents the body of the Word document. By altering this part using any technology capable of editing XML and returning the XML part to the container package, creates an updated Office document. This new file format is classified into three main parts.

3.2.1 Container – Package

OOXML documents are stored in OPC package form, which is a ZIP file, containing XML and other data parts [11, 14]. The relationships specification between the parts is also stored inside the container. Relationship information is used by an application to locate individual parts within a package. The package can have different internal directory structure and names depending on the type of the document as shown in Fig.3.1.

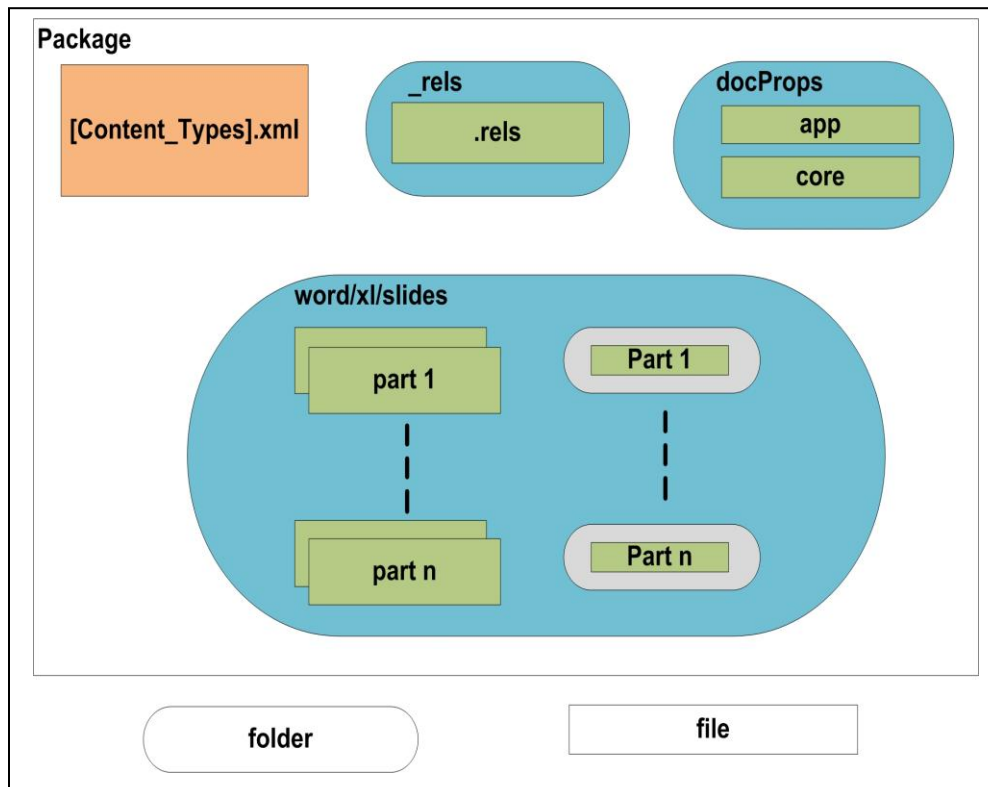


Fig. 3.1: Internal directory structure of an OOXML document, shows package, parts and relationships files a package may contain.

A basic package contains an XML file called [Content_Types].xml at the root, along with three directories: “_rels”, “docProps” and document type specific directory. For example, in MS Word 2007 document the “word” directory has been created and contains the “document.xml” file which is the starting path of the document. These folders have all the files located in the package and zipped together to form a single instance of the document. Every part in a package has a unique URI (Uniform Resource Identifier) part name along with specified content type. A part’s content type explicitly defines the type of data stored and reduces ambiguity and duplication issues inherent with file extensions. Package can also include relationships that define association between the package, parts and external resources.

3.2.2 Package Parts

Parts inside a package are of any type including text, image etc [11, 14]. The extension “.rels” is reserved for storing relationship information of package parts and stored in “/rels” subfolders. Three names are reserved by package for organizing its files i.e. “_rels” subfolder carrying relationship information with “.rels” file extension and file name “[Content_Type].xml”. Fig. 3.2 shows the logical organization of package parts.

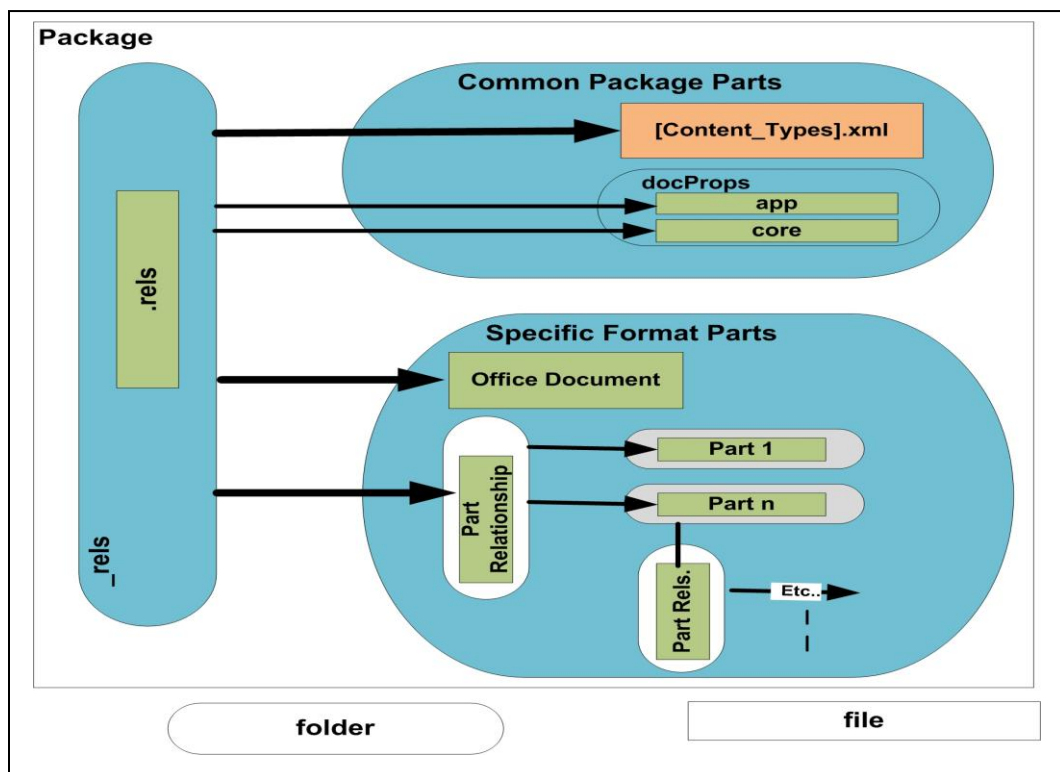


Fig. 3.2: A logical structured organization of package parts of an OOXML document. The “common package parts” exists in all OOXML document, whereas “specific format parts” depends on type of application used.

Where, “[Content_Types].xml” file provides MIME (Multipurpose Internet Mail Extensions) type information for parts used in the OOXML document. It also defines mapping based on the file extensions, along with overrides for specific parts other than default file extensions. This enable an application and third party tools to determine the

contents of any part to process accurately; “docProps/app.xml” file contains application centric properties such as application type “Microsoft Office Word” etc; “docProps/core.xml” file contains OOXML document core properties such as machine name, creation and modification dates etc; whereas “word/document.xml” is the main part of any word document.

3.2.3 Relationships

Relationship items classify that how the document parts are placed together to form a document as shown in Fig. 3.3. This is achieved by verifying connection between source part and target part. Two types of relationships are permitted in OOXML documents, internal and external [11, 14]. All relationships, including the relations associated with the root package, are represented as XML files. These XML files contain relationships information and are stored inside a package, for example, default location for relationships is “/_rels/.rels”. Relationships are composed of four elements: an identifier (Id), an optional source (package or part), relationship type (URI style expression) and a target (URI to another part). Two types of relationship files usually exist in a package. These are:

- **/_rels/.rels:** Root level “_rels” folder contains relationship file which carries information of parts for the package. For example “_rels/.rels” file defines the starting part of the document i.e. “word/document.xml”.
- **[partname].rels:** Each part may have its own relationships. The part specific relationship can be looked in “word/_rels” subfolder, a sibling of the file with original file name appended to it with “.rels” extension. For example “word/_rels/document.xml.rels”. Figure 3.3, Showing how package parts are tied

with each other using relationship information associated with each part. Highlighted nodes link associated XML files of a package to represent document.

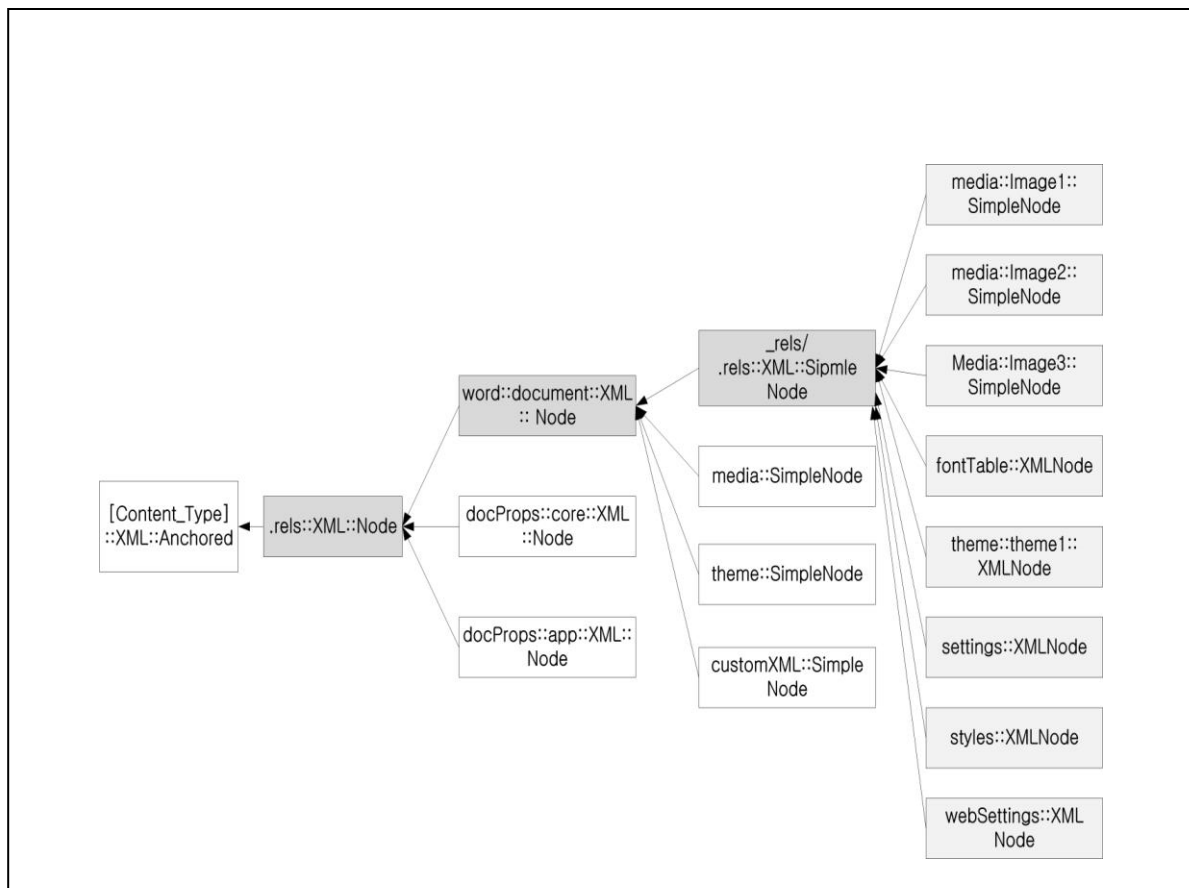


Fig. 3.3: Logical representation of OOXML document relationships.

A typical package relationships file “.rels” contains XML code and for simplicity we only present XML code for “document.xml” part as follows:

```
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId1"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument" Target="word/document.xml" />
</Relationships>
```

In above code, “Relationship Id” attribute value “rId1” is default for main document part which is the starting part of a document. Once the document is being launched the

OOXML editor looks for an OOXML parser to be used depending on document type. In this case the type specifies that MS Word ML is being used for MS Word document. Another attribute “Target” specifies path or location of beginning part i.e. document.xml.

3.3 Data Hiding in OOXML - MS Word 2007 documents

Next, we will briefly introduce several successful data hiding techniques for OOXML files. These techniques can be classified into different categories including data hiding using OOXML relationship structure, data hiding using XML format features, data hiding using XML format features and OOXML relationship structure, data hiding using OOXML flexibility for embedded resource architecture and data hiding using OOXML flexibility of swapping parts. We use MS Word 2007 document, as an example, to illustrate our concepts, but the methodology can be easily extended to any documents in MS Office 2007 or OOXML file format.

3.3.1 Data Hiding using OOXML Relationship Structure

As mentioned earlier the MS Office 2007 document is comprises of several xml and other files. These files are known as parts and compressed together using ZIP format. These parts are also organized using the relationship information found in relationship files inside an OOXML document. To satisfy relationships within a document, all parts have to be a target of valid relationship entry. Parts which are not the target of a valid relationship entry, we considered that part as an unknown part [15]. These parts are not to be ignored when reading the document by MS Office 2007 application and raised an error that the document is corrupt as shown in Fig. 3.4. MS Office 2007 also facilitate user by

giving an option to recover the document and removes the unknown parts as seen in Fig. 3.5.

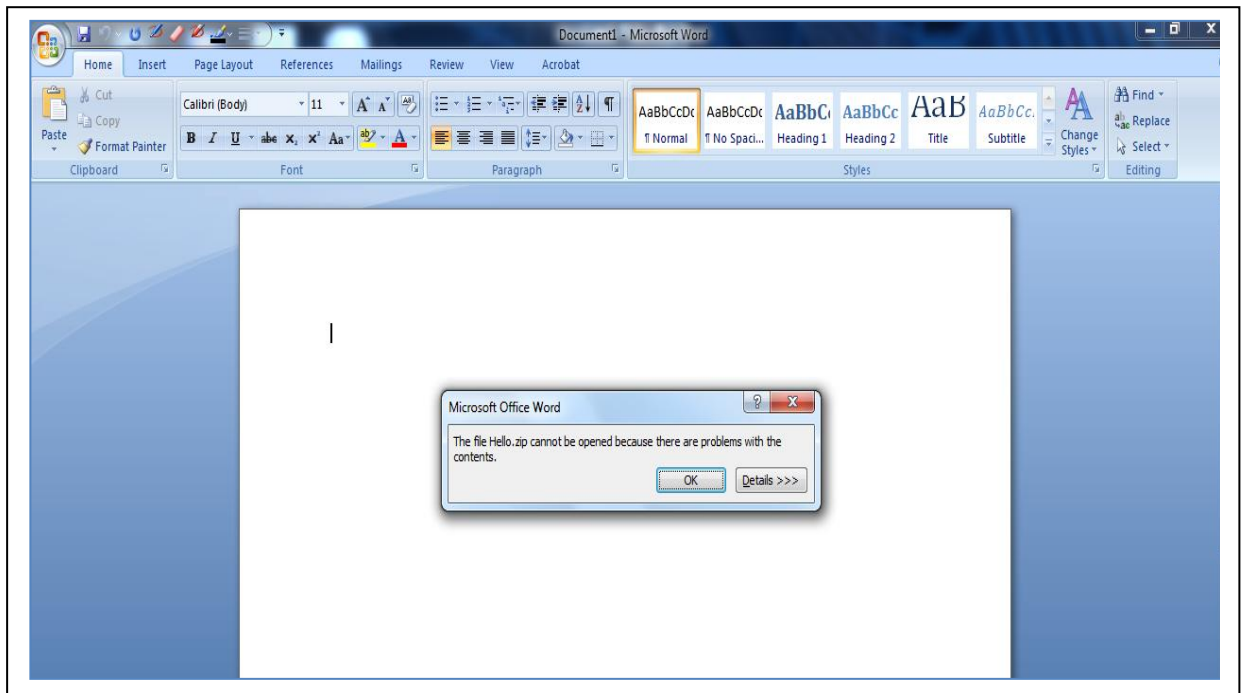


Fig. 3.4: MS Office 2007 raised an error stated that problems with the contents

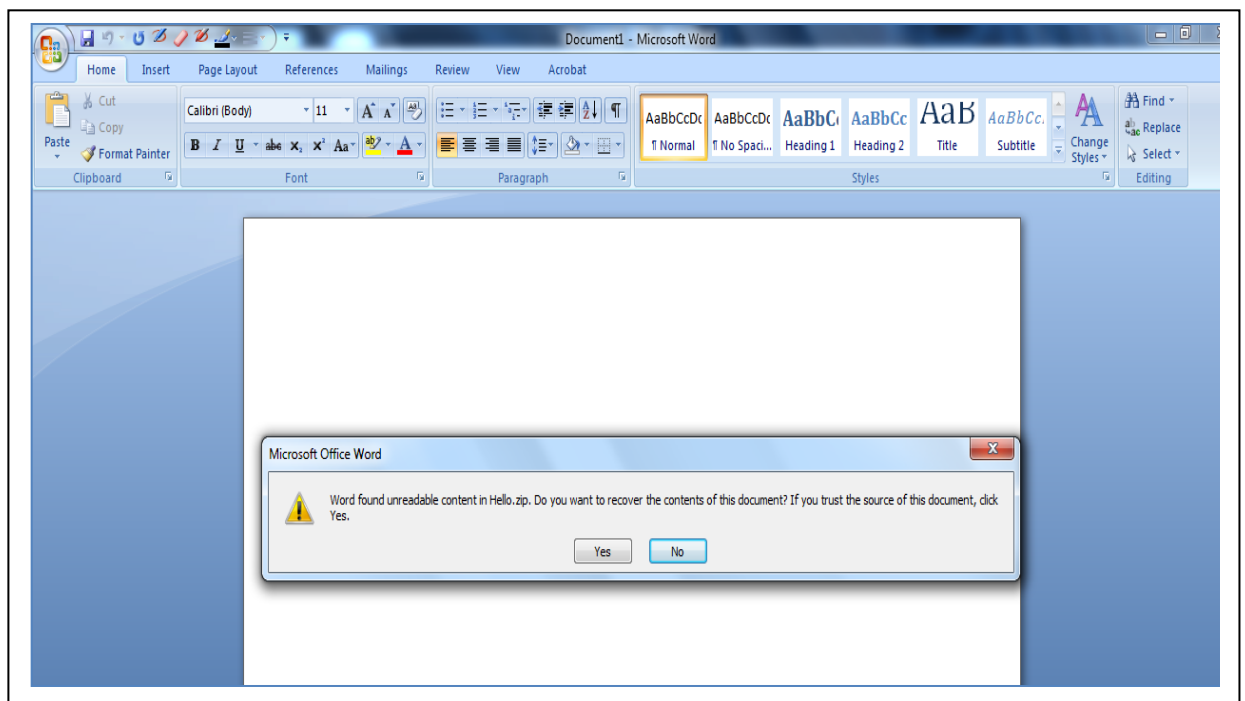


Fig. 3.5: MS Office 2007 gives an option to recover the document by removing unknown parts

In similar way the relationship entry which is not defined in ECMA-376 standard is also considered as unknown relationships. These relationship entries are accepted in OOXML document and raise no errors. Document containing unknown relationship information are opened normally and the relationship entry is also being found inside a relationship file. This is also remains present if user saves a document with new name.

Next we provide specific example of data hiding by using information of relationship structure of OOXML document. The OOXML document works as a carrier for the hidden data. In this case MS Word 2007 document is used in this data hiding process.

- The first step of data hiding initiates with unzipping the OOXML document using any zip utility software. This shows that OOXML document contains several XML files and other objects.
- In Second step, insert files wish to hide in the unzipped OOXML document archive. These files can be added to any folder or sub folder of the document.
- Third, we need to define types of added files into OOXML documents content type's file. There is no need to define file types multiple times if it's already exist in the content type file.
- The fourth step is defining relationships entry for inserted files into package relationship file i.e. "_rels/.rels" The attributes of relationship entry is "Id", "Type" and "Target". The relationship attribute "Id" must be unique as it connects the document and the target files, whereas "Type" attribute is some character which is not defined in OOXML standard such as "a", "b" etc. The "Target" attribute contains the complete path or location from the document's root folder for the inserted files.

- Finally, all the files are zipped together with an extension of “.zip” which later is renamed to “.docx”.

For example, we created a document containing some text and image and saved it as “Uparts&Rels”. Next by using WinZip utility we unzipped this document and inserted “sysinternals.zip”, “mask.jpeg” and “BYE.mp3” as our hidden files in the root folder of the document i.e. “Uparts&Rels”. After inserting these files we updated the “[Content_Types].xml” file for inserting files types. The content type file can be found inside root folder named “[Content_Types].xml”. Before opening a document, the OOXML parser validates that the files present in the package are defined in [Content_Types].xml file. The code for defining hidden file types into “[Content_Type].xml” file is highlighted with existing code as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Types
  xmlns="http://schemas.openxmlformats.org/package/2006/content-
  types">
  <Override PartName="/word/footnotes.xml" ContentType="....." />
  <Default Extension="jpeg" ContentType="image/jpeg" />
  <Default Extension="rels" ContentType="application/vnd.openxmlformats-
  package.relationships+xml" />
  <Default Extension="xml" ContentType="application/xml" />
  <Default Extension="zip" ContentType="application/zip" />
  <Default Extension="mp3" ContentType="application/mp3" />
  <Default Extension="jpg" ContentType="application/jpg" />
  <Override PartName="/word/document.xml" ContentType="....." />
  .....
</Types>
```

Fig. 3.6 – Modified [Content_Types].xml File

At last the package level relationship file “_rels/.rels” is used for creating relationships of hidden files within a package. The relationship entry attributes such as Id, Type, and

Target are defined in relationship file. Entry showing parts including unknown parts are as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">

  <Relationship Id="rId3" Type="..." Target="docProps/app.xml" />

  <Relationship Id="rId2" Type="..." Target="docProps/core.xml" />

  <Relationship Id="rId1" Type="..." Target="word/document.xml" />

  <Relationship Id="rId100"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/a"
    Target="word/media/sysinternals.zip" />

  <Relationship Id="rId101"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/b"
    Target="mask.jpg" />

  <Relationship Id="rId102"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/c"
    Target="word/BYE.mp3" />

</Relationships>
```

Fig. 3.7 – Modified relationship file (.rels)

As seen above, the relationship Id of “BYE.mp3” is “rd102” and Type is `http://schemas.openxmlformats.org/officeDocument/2006/Relationships/c`. Note as mentioned above when setting a type we use values that does not exist in the OOXML specifications such as “a, b, c, etc”. After these modifications the OOXML document is opened normally without a warning. Also if user amends the document and updates it, the hidden data remains in the document. The MS Office application considers unknown parts and unknown relationships as valid parts and relationships of a package.

As explained earlier that hidden data must not satisfy all the relationships in the package, otherwise it is visible in document. Also if only an inserted file type is defined in Content Type file and its relationship is not created in relationship file, the document opens normally. Office application does not give any warning and inserted file still exists inside the package. In this case if user updates or makes some changes in the word document, then inserted file is being eliminated by the application. So for keeping the existence of inserted file, its relationship needs to be created in relationship file.

This data hiding method is the natural result of an explicit relationship in OOXML. The key point of this hiding process is to assign a fresh Id to new target which results that the target being overlooked by the MS Office application. The new Id is not referenced in the relationship part and the main source part is not aware of the new content and the hidden data is not shown on screen and neither can be eliminated by MS Office application because these hidden data have an Id and satisfies relationship structure of OOXML document. At this point, if relationships between main MS Office document file and hidden data are defined, the hidden data becomes more difficult to discover.

This data concealment approach also sidesteps the document inspection feature “Inspect document” available in MS Office 2007 applications.

3.3.2 Data Hiding using XML Format Feature

XML comment feature is used to leave a note or to temporarily edit out a portion of XML code. Although XML is supposed to be self-describing data, you may still come across some instances where an XML comment might be necessary. XML comments follow the exact same syntax as HTML comments. XML features are fully supported by OOXML

documents as contain several XML files. The text in OOXML documents are organized into XML sections and are repetitive. OOXML documents do not generate any comments in XML files of a document whereas XML comments feature can easily be used for data hiding purpose by some mischievous user.

It is highlighted that data can be hidden inside an OOXML document using comments feature. This involves technique of adding comments directly to zip archive using comment feature of zip file format and adding XML comments to the XML file [10]. In both cases, these comments are ignored by MS Office 2007 application and these comments are discarded when document is written back out.

Further investigation proves that base64 encoding method allows successful embedding of file as a comment in one of the XML files of OOXML document and MS Office 2007 ignores it silently and document opens normally. Base64 provided a very simple yet effective way of taking information with a wide range of characters (ASCII has 127 possible characters) and converting this into a method that can be stored with a smaller range of characters (base64 output uses 64 characters, as the name implies).

This technique hides a file by taking advantage of XML format feature such as comments. Anything in comments is effectively invisible to the XML parser but for use of successful data hiding in OOXML documents, first the data is encoded to base64 format. This data concealment process requires following steps.

- First, an OOXML document is being unzipped using WinZip utility.
- Second, the data or file to be hidden is encoded to base64 format.

- Third, the data or file is added in any of the XML file present inside an OOXML document.
- Finally, all the files are zipped together with an extension of “.zip” which later is renamed to “.docx”.

For Example, we created one document contains some text and an image. Then unzip it using any zip utility and a secret message is added using comments feature in one of the XML file. The secret message is as follows:

```
<!-- This is a secret message-->
```

The secret message is encoded using base64 encoding scheme and the above message after encoding is as follows:

```
PCEtLSBUaGlzIGlzIGEgc2VjcmV0IG1lc3NhZ2UtLT4=
```

After the base64 encoding, we add this data to any of the XML file. According to XML standard the comments cannot appear at the very top of your document in XML, only the XML declaration can come first such as:

```
<?xml version="1.0"?>
```

Document inspection feature of MS Office 2007 allow removing of comments from the document. Whereas, by using this approach the document inspection feature fails to identify and remove comments embedded using base64 encoding scheme. This technique also enables to hid some file in an OOXML document using base64 encoding scheme. The quality of this data hiding technique is relatively low as XML files carries base64 encoded data and could be easily notice by others if unzips the document.

3.3.3 Data Hiding using XML Format Feature and OOXML Relationship Structure

Ignorable attribute is also an important feature of MS Office 2007 applications [14]. For any associated XML element, compatibility rules are very important. Compatibility rules are associated with an element by means of compatibility rule attributes. These controls how MS Office 2007 parser shall react to elements or attributes from unknown namespaces. The principal compatibility rule attribute is the Ignorable attribute. By default MS Office 2007 treat the presence of any unknown element or attribute as an error condition [10, 14]. However, unknown elements or attributes identified in an Ignorable attribute shall be ignored without error.

The ignorable attribute specifies which XML namespace prefixes encountered in a markup file may be ignored by an OOXML processor. Elements or attributes, where the prefix portion of the element name are identified as “**ve:Ignorable**”, will not raise an error when processed by an OOXML processor. The “**ve**” XML namespace is the recommended prefix convention to use when mapping the XAML (Extensible Application Markup Language) compatibility namespace “<http://schemas.openxmlformats.org/markup-compatibility/2006>”. The “**ve:Ignorable**” attribute supports markup compatibility both for custom namespace mapping and for XML versioning.

ECMA-376 specification describes the Ignorable attribute: “A whitespace-delimited list of namespace prefixes identifying a set of namespaces whose elements and attributes should be silently ignored by markup consumers that do not understand the namespace of the element or attribute in question”.

This data concealment technique supports any kind of data such as image, audio or video file to be hidden by taking advantage of XML format feature and OOXML relationship structure. This technique coerces XAML parser to treat that element and attributes as it does not exist and shall not generate an error. By default, ignore element is entirely ignored including its attributes and contents. This data concealment process requires following steps.

- First, an OOXML document is being unzipped using WinZip utility.
- Second, add an image which needs to be hidden using this technique in sub folder named “word/media”. This sub folder usually contains all images used inside a document.
- Third, for the hidden image to be looking legitimate, we created metadata for hidden image inside main document file “document.xml”.
- Fourth, use Ignorable attribute to define this in declaration section of main document file i.e. “document.xml” and place this tag before and after of created metadata for hiding image.
- Fifth, the part relationship file “document.xml.rels” is amended with creating relationship attributes such as “Id”, “Type” and “Target”. The “Id” must be unique and other attributes “Type” and “Target” contains information of image type and location.
- Finally, all the files are zipped together with an extension of “.zip” which later is renamed to “.docx”.

For example, we created and saved a document containing image and some text data named as “Ignore.docx”. After unzipping this document we add image which needs to be hidden inside a document under “word/media” subfolder as shown in Fig. 3.8.

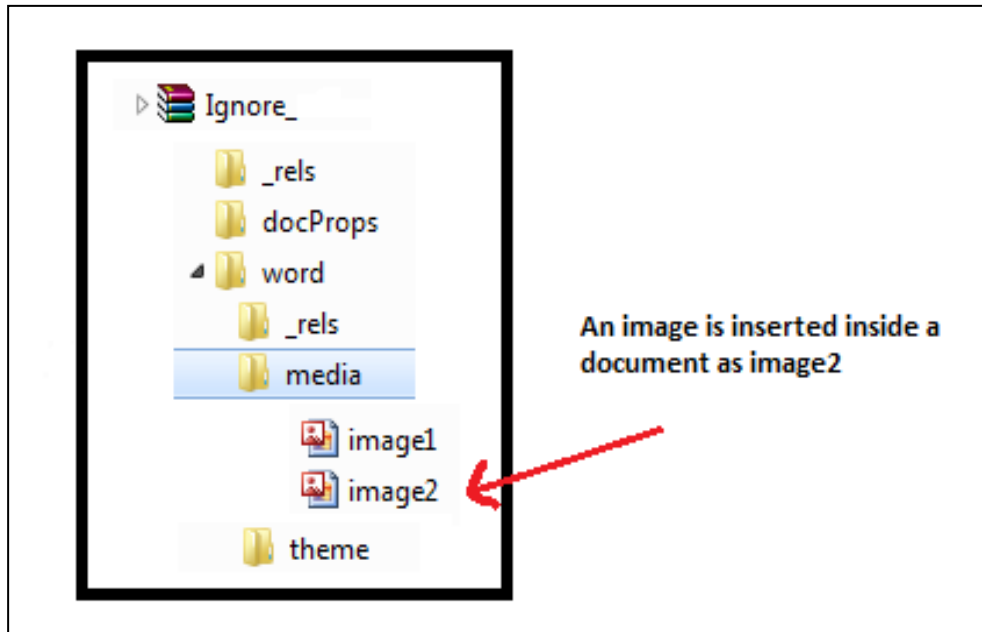


Fig. 3.8 – Added Image shown with other files in OOXML document

After inserting an image in “word/media” subfolder, the main document file is updated with the metadata code for an inserted image. For simplicity we copy the metadata code of first image inserted using MS Office application and paste as a separate block for hidden image in “document.xml” file. Next to hide this image, the Ignorable attribute is defined inside a main document declaration section as shown in Fig. 3.9. The code highlighted in red is used to define ignorable namespace which markup consumer does not understand. After defining the ignorable attribute namespace, the ignorable tag is placed before and after the metadata of second image need to be hides. The hidden image is “Garden.jpeg” and its metadata tags in document.xml file shown in Fig. 3.12.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
xmlns:w10="urn:schemas-microsoft-com:office:word"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
xmlns:p1="http://schemas.openxmlformats.org/MyExtension/p1" ve:Ignorable="p1" >

```

Fig. 3.9 – Beginning section of main document file “document.xml”

```

<w:sectPr w:rsidR="00401AD8" w:rsidRPr="0091776E" w:rsidSect="00DC51FF">

<w:pgSz w:w="12240" w:h="15840" />

<w:pgMar w:top="1440" w:right="1440" w:bottom="1440" w:left="1440" w:header="720"
w:footer="720" w:gutter="0" />

<w:cols w:space="720" />

<w:docGrid w:linePitch="360" />

</w:sectPr>

</w:body>

</w:document>

```

Fig. 3.10 – End section of main document file “document.xml”

```

<w:body>
<w:p w:rsidR="00DC51FF" w:rsidRDefault="00401AD8">
<w:r>
<w:t>Hello</w:t>
</w:r>
</w:p>
<w:p w:rsidR="0091776E" w:rsidRDefault="00401AD8">
<w:r>
<w:drawing>
<wp:inline distT="0" distB="0" distL="0" distR="0">
<wp:extent cx="5943600" cy="4457700" />
<wp:effectExtent l="19050" t="0" r="0" b="0" />
<wp:docPr id="1" name="Picture 0" descr="Dock.jpg" />
<wp:cNvGraphicFramePr>
<a:graphicFrameLocks
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
noChangeAspect="1" />
</wp:cNvGraphicFramePr>
<a:graphic
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
<a:graphicData
uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:pic
xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:nvPicPr>
<pic:cNvPr id="0" name="Dock.jpg" />
<pic:cNvPicPr />
</pic:nvPicPr>
<pic:blipFill>
<a:blip r:embed="rId4" cstate="print" />
<a:stretch>
<a:fillRect />
</a:stretch>
</pic:blipFill>
<pic:spPr>
<a:xfrm>
<a:off x="0" y="0" />
<a:ext cx="5943600" cy="4457700" />
</a:xfrm>
<a:prstGeom prst="rect">
<a:avLst />
</a:prstGeom>
</pic:spPr>
</pic:pic>
</a:graphicData>
</a:graphic>
</wp:inline>
</w:drawing>
</w:r>
</w:p>

```

Fig. 3.11 – Metadata of first image, inserted using MS Word 2007

```

<p1:IgnoreMe>
<w:p w:rsidR="00401AD8" w:rsidRPr="0091776E"
w:rsidRDefault="0091776E" w:rsidP="0091776E">
<w:pPr>
<w:tabs>
<w:tab w:val="left" w:pos="1155" />
</w:tabs>
</w:pPr>
<w:r>
<w:lastRenderedPageBreak />
<w:drawing>
<wp:inline distT="0" distB="0" distL="0" distR="0">
<wp:extent cx="5943600" cy="4457700" />
<wp:effectExtent l="19050" t="0" r="0" b="0" />
<wp:docPr id="2" name="Picture 1" descr="Garden.jpg" />
<wp:cNvGraphicFramePr>
<a:graphicFrameLocks
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
noChangeAspect="1" />
</wp:cNvGraphicFramePr>
<a:graphic
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
<a:graphicData
uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:pic
xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
<pic:nvPicPr>
<pic:cNvPr id="0" name="Garden.jpg" />
<pic:cNvPicPr />
</pic:nvPicPr>
<pic:blipFill>
<a:blip r:embed="rId5" cstate="print" />
<a:stretch>
<a:fillRect />
</a:stretch>
</pic:blipFill>
<pic:spPr>
<a:xfrm>
<a:off x="0" y="0" />
<a:ext cx="5943600" cy="4457700" />
</a:xfrm>
<a:prstGeom prst="rect">
<a:avLst />
</a:prstGeom>
</pic:spPr>
</pic:pic>
</a:graphicData>
</a:graphic>
</wp:inline>
</w:drawing>
</w:r>
</w:p>
</p1:IgnoreMe>

```

Fig. 3.12 – Sample metadata of hidden image

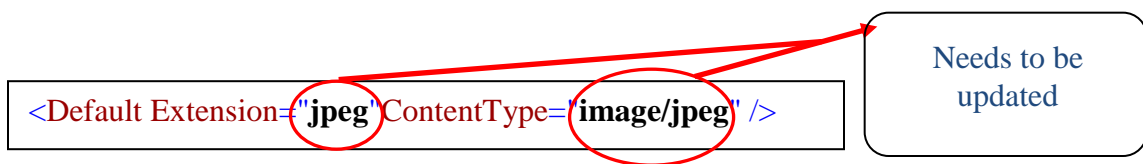
Finally, the part relationship file “document.xml.rels” is updated with a valid relationship entry of an inserted image along with its type and target information. In this case the highlighted relationship entry having Id “rId5” is added in relationship file with type “http://schemas.openxmlformats.org/officeDocument/2006/relationships/image” and target “media/image2.jpeg” values. The highlighted code for this entry is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId3"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/web
    Settings" Target="webSettings.xml" />
  <Relationship Id="rId7"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/the
    me" Target="theme/theme1.xml" />
  <Relationship Id="rId2"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/setti
    ngs" Target="settings.xml" />
  <Relationship Id="rId1"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styl
    es" Target="styles.xml" />
  <Relationship Id="rId6"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/font
    Table" Target="fontTable.xml" />
  <Relationship Id="rId5"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ima
    ge" Target="media/image2.jpeg" />
  <Relationship Id="rId4"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ima
    ge" Target="media/image1.jpeg" />
</Relationships>
```

Fig. 3.13 – Relationship file showing entry for hidden image

At this point the image is successfully hidden using Ignorable attribute. To keep this hidden image intact with the document its relationship must be created. Only using XML

feature such as ignorable attribute does not guarantee of keeping hidden image with the document and requires both the techniques for data hiding. The content type file is not required to amend with image type if same type of image is inserted. If inserted image type is different than first image, the content type file also needs to be updated with hidden image type. The sample code for updating content type is as follows: The [Content_Types].xml file is located at root level in the package.



Data hides using this technique is really hard to trace as it conform all the association to the main document file. The relationship entry is also exist in relationship file, and with the insertion of metadata code of hidden image in “document.xml” file do not raise any doubt of illegitimate data hidden inside an OOXML document.

3.3.4 Data Hiding Using OOXML Flexibility For Embedded Resource Architecture

The Custom XML feature is one of the most powerful features of OOXML documents for business scenario and document centric solutions [14]. It supports integration of documents with business process and data to get true interoperability of documents. This is a really powerful concept and entails to store custom xml file in the package, binding content controls to elements using implicit relationship, and control the display of this data used in the document as custom XML part. This allows you to embed business semantics in such a way that it is discoverable, and implementers not interested in using that feature can skip over it easily, without needing to know what application stored it

there. You can even easily extract your data using one simple generic XSLT (Extensible Stylesheet Language Transformations).

A package is permitted to contain multiple custom XML data storage parts. The ability to put custom XML file in the package means that; now have a place to store any data solution required and travels with the document. This standard curtails to have implicit or explicit relationship with other parts in the package. The Custom XML feature is fully implemented for MS Word 2007. MS Excel and MS Powerpoint 2007 do not handle custom XML the same way. The further announcements from Microsoft for translators from binary to OOXML format might be another good advantage.

The ability to embed and interweave business data into transportable and humanly readable documents is extremely useful. Take for instance the efforts to standardize the embedding of patient medical data into PDF documents, (aka PDF/H). Records For Living has been able to take advantage of Open XML's capabilities with regards to its support of custom schemas to integrate two industry standards: Ecma's Open XML and the ASTM's Continuity of Care Record (CCR). The combination is powerful: patients can use personal health record (PHR) software to exchange live reports with their doctors in a way that is both human and machine readable [20].

This feature also empowers data concealment using OOXML flexibility for embedded resources inside MS Office 2007 document. The Custom XML data is also been generated by OOXML document in some cases. The object embedding feature of OOXML document requires generation of Custom XML data for handling information of that object. This is also being in the case if integrating OOXML document data with real world data. First we take a brief overview of CustomXML data generated by an OOXML

document application and then present our data hiding method using CustomXML data scheme. The default layout of the unzip OOXML document contains Custom XML data is as follows.

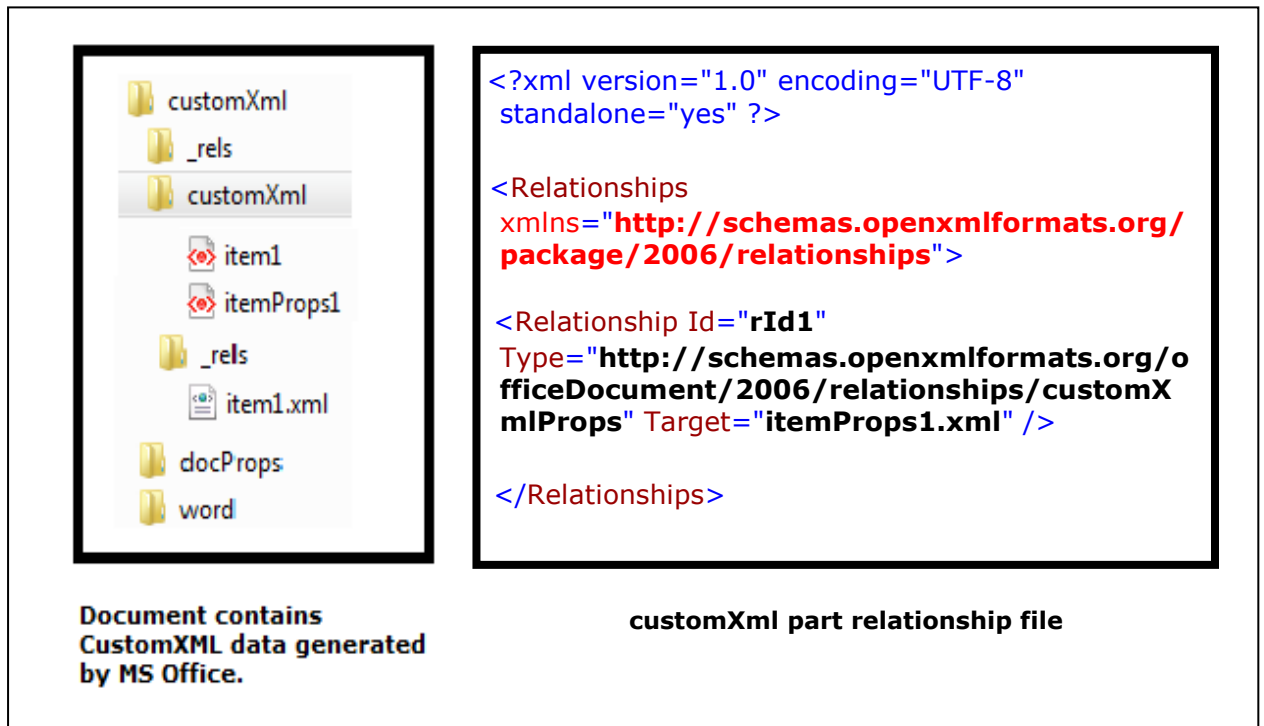


Fig. 3.14 – Document Structure with Custom XML data and its part relationship code

By default an OOXML document creates customXML folder at root to store custom XML data files. This folder contains two XML files, “item1.xml” and “itemProps1.xml”. The part relationship file of customXML data is being generated at “customXml” folder under “customXml/_rels” subfolder as “item1.xml.rels”, and contains relationship entry for “itemProps1.xml” file. Relationship entry for the custom XML files is shown as Fig. 3.14 and the part relationship file of main document “document.xml.rels” is also updated with an entry of second custom XML file i.e. “item1.xml” with Id “rId1” as shown in Fig. 3.15.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships
  xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId8"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml" />
  <Relationship Id="rId3"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml" />
  <Relationship Id="rId7"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml" />
  <Relationship Id="rId2"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml" />
  <Relationship Id="rId1"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXml" Target="../../customXml/item1.xml" />
  <Relationship Id="rId6"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/endnotes" Target="endnotes.xml" />
  <Relationship Id="rId5"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/footnotes" Target="footnotes.xml" />
  <Relationship Id="rId4"
    Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml" />
</Relationships>

```

Fig. 3.15. Part relationship file code “document.xml.rels”

Next the steps of data hiding using CustomXml feature is as follows:

- First, create and save an OOXML document named “CustomXML.docx” contains text and image.
- Second, creates a folder at root named “customXml” and insert some text file need to hides in OOXML document. The text file is in XML format to looks legitimate CustomXML data.

- Third, creates a subfolder named “customXml/_rels” and also creates one relationship file for hidden text file.
- Finally, all the files are zipped together with an extension of “.zip” and later rename it to “.docx”.

For example, we use this technique to hide some text data inside MS Office document. We created “customXml” folder at root of the package and insert one text file named “hiddendata.xml”. The code of sample “hiddendata.xml” file is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<w:document
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXml"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
<w:body>
<w:p w:rsidR="00DC51FF" w:rsidRDefault="00C2303E">
<w:r>
<w:t>Hidden Secret Message!!!!</w:t>
</w:r>
</w:p>
</w:body>
```

Fig. 3.17 – Sample code of “hiddendata.xml” file

Next we satisfy its relationship constraint by creating its relationship file inside a “customXml/_rels” subfolder and named this “hiddendata.xml.rels” shown as Fig. 3.18. We observe that if we do not skip its entry in part relationship file of main document, the Inspect Document feature can easily identify the custom XML data and allows user to discard the custom XML data.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<Relationships

xmlns="http://schemas.openxmlformats.org/package/2006/relationships">

  <Relationship Id="rId100"

Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/custom

XmlData" Target="/customXML/test.xml" />

</Relationships>
```

Fig. 3.18 – Sample code of customXml part relationship file

This technique allows insertion of any text file in xml format contains hidden data and with its internal sub-relationship. This also looks legitimate as carries Custom XML data in real business environment. The key point to identify whether this is hidden data or not is to see, that whether CustomXml type entry is present in part relationship file of main document. The hidden data under customXml folder is not linked with the main document file “document.xml”.

The MS Office document gives feature that allows you to remove custom XML data associated with the document. This feature is named as Inspect document which removes custom XML data, hidden data and other personal information form MS Office document. The snapshot of this feature is as follows:

The inspect document feature functionality is to search customXml type in part relationship file of main document (“document.xml.rels”). Once found, it deleted the associated data using target information and also discard customXml folder with its contents.

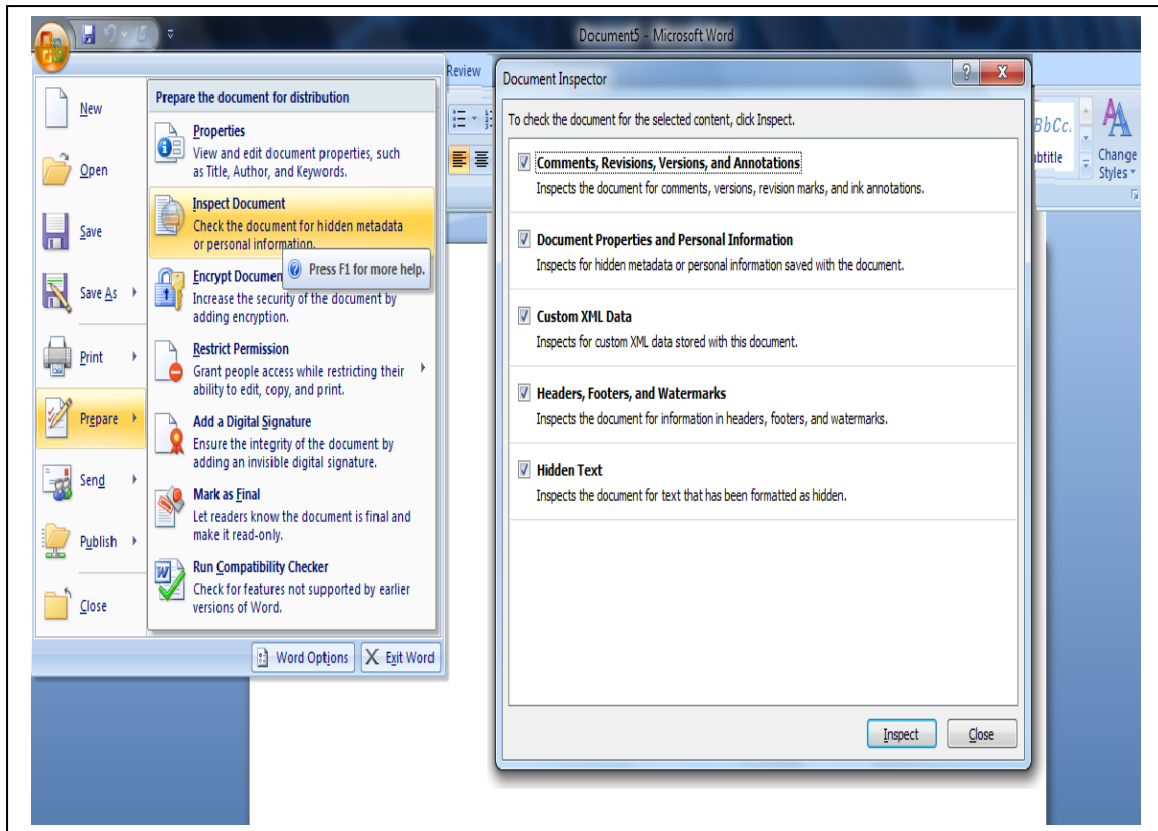


Fig. 3.19 – Inspect document feature of MS Office 2007

The document inspection feature of MS Office 2007 is unable to detect custom XML data in this scenario as it sidesteps the document inspection feature of MS Office 2007 application.

3.3.5 Data Hiding Using OOXML Flexibility of Swapping Parts

Images are the most popular cover objects used for steganography. In the domain of digital images many different image file formats exist, most of them for specific applications. The MS Office 2007 uses “png”, “jpeg”, “gif” and “emf” formats for storing images inside a document [13].

An image is a collection of numbers that constitute different light intensities in different areas of the image. This numeric representation forms a grid and the individual points are

referred to as pixels. Most images on the Internet consists of a rectangular map of the image's pixels (represented as bits) where each pixel is located with its color. These pixels are displayed horizontally row by row. Digital color images are typically stored in 24-bit files and use the RGB color model, also known as true color. All color variations for the pixels of a 24-bit image are derived from three primary colors: red, green and blue, and each primary color is represented by 8 bits. Thus in one given pixel, there can be 256 different quantities of red, green and blue, adding up to more than 16-million combinations, resulting in more than 16-million colors [21].

When working with larger images of greater bit depth, the images tend to become too large to transmit over a standard Internet connection. In order to display an image in a reasonable amount of time, techniques must be incorporated to reduce the image's file size. These techniques make use of mathematical formulas to analyze and condense image data, resulting in smaller file sizes. This process is called compression [22].

In images there are two types of compression: lossy and lossless. Both methods save storage space, but the implemented procedures are different. Lossy compression creates smaller files by discarding excess image data from the original image. It removes details that are too small for the human eye to differentiate, resulting in close approximations of the original image, although not an exact duplicate. An example of an image format that uses this compression technique is JPEG (Joint Photographic Experts Group). Lossless compression, on the other hand, never removes any information from the original image, but instead represents data in mathematical formulas. The original image's integrity is maintained and the decompressed image output is bit-by-bit identical to the original

image input. The most popular image formats that use lossless compression is GIF (Graphical Interchange Format) and 8-bit BMP (a Microsoft Windows bitmap file).

The compression technique used by MS Office 2007 for image compression requires an image is first 2D transformed and converted into emu (English Measurement Unit) rather than device pixels for implementation as recommended [11, 14].

The flexibility of data hiding using OOXML architecture of swapping parts allows swapping of images between two OOXML documents. This data hiding scheme supports two data hiding scenarios as follows.

Scenario 1

- First, the OOMXL document is unzipped using WinZip utility.
- Second, swap an image with the original image found in “word/media” subfolder. Also ensure that the swapped image follows the same name of an original image. The inserted image is being transformed according to OOXML image transformation standard before swapping.
- Third, the content type file is needed to be updated if swapped image is of another format.
- Finally, all the files are zipped together with an extension of “.zip” and later rename it to “.docx”.

Scenario 2

- First, the OOMXL document is unzipped using WinZip utility.
- Second, an original image found inside “word/media” subfolder used to embed files using any image stego software.

- Finally, all the files are zipped together with an extension of “.zip” and later rename it to “.docx”.

For example, the swapped image has to be transformed or compressed first before swapping otherwise OOXML document raise an error. The best way is to add desired image into MS Office 2007 document and save the document. This way an image is automatically transformed by the application. Later unzip this document and swap image with another image present in different document as shown in Fig. 3.20.

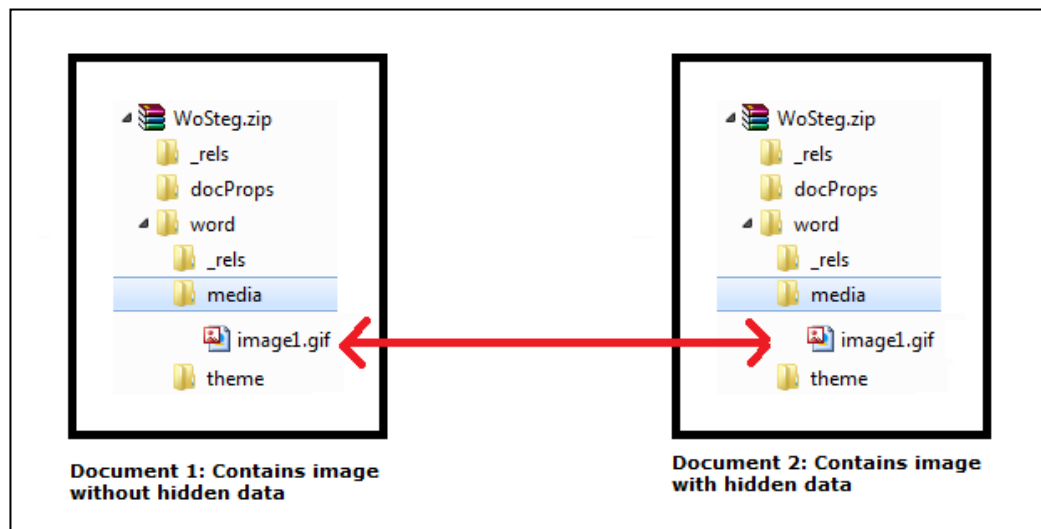


Fig. 3.20 – Swapping of image between 2 OOXML documents

This facilitates mischievous user to swap images of two documents, and by using any available image steganography tool can embeds files into an existing image. The replaced image should be of same type of swapped image otherwise the content type is updated with newly inserted image type. The OOXML document flexibility for embedding resources proves that mischievous user can easily embeds data inside an image present in the document.

The image steganography tools used least significant bit (LSB) approach for bitmap format, palette based approach for gif format and Discrete Cosine Transform (DCT) or Discrete Fourier Transform (DFT) for JPEG format images. Swapping of images cannot be detected by using any feature of MS Office 2007 application and hence seems that manual scrutiny of the images are required to ensure that it does not carries any hidden data.

This technique also supports steganography and MS Office 2007 is unable to detect the changes made into an image file and OOXML document opens normally.

Chapter 4

Hidden Data Detection in OOXML Documents

4.1 Introduction

As XML becomes the standard format for data exchange between inter-enterprise applications on the internet and enables the transmission of highly structured data using standard HTTP protocol instead of proprietary solutions, which sometimes do not allow for compatibility [23, 24]. The variety of data types is very large and also the need of bi-directional data exchange can largely benefit from the advantages offered by this new technology. That's changing as Microsoft, Sun Microsystems, and other developers migrate to new XML-based formats for document files [10].

XML is a <tag> based language incorporating a number of features directed to the hierarchical classification of the data. It is extensible, since the user can define its own labels, and it is object-oriented, when most current systems are procedural-oriented. Moreover, it allows the inclusion of metadata, i.e., the description of the structure and format of the data goes along with the data itself, and it includes mechanisms for validating the structure of the data records. This achieves all what the industry data format standards aimed for, whilst providing wide compatibility and allowing the use of the information outside the proprietary database environment.

To facilitate data exchange, industry groups define document type definition (DTDs) that specify the format of the XML data to be exchanged between their applications [25]. As seen in the previous sections, the MS Office document comprises of several XML files and parts. These files are carrying collective information to intact parts used inside a

main document. The relationship files contain the information of organization of a package which is actually an OOXML document. The main package level relationship file contains information for OOXML editor defining which application is being used to process the document, like MS Word 2007 in our case, whereas the part relationship information file contains information of associated parts and layout of a document. Therefore, the identification of hidden parts inside a package requires querying the contents of multiple xml files. Basically the query identifies concealed data in multiple XML files which are actually not generated by MS Office application.

Also discussed earlier the significance of the fulfilling relationship standard within the package is essential. A complete content to be correctly presented in an OOXML file editor, the relationships between the various parts in the package needs to be satisfied. For example, an image is saved as a separate part in the package and only its relationship Id and metadata information is stored in the main document. When the main document is launched, the parts associated with the main document and their corresponding relationship Id's have been searched in the parts relationship file, which contains the information such as type and target (location) of the parts. The OOXML parser brings the part (image) by using the relationship information and placed it in the main document and makes this whole process transparent from the user.

Fig. 4.1 shows the package layout of the MS Office document and Fig. 4.2 shows the logical order that how associated parts are referenced using relationship information with the main document.

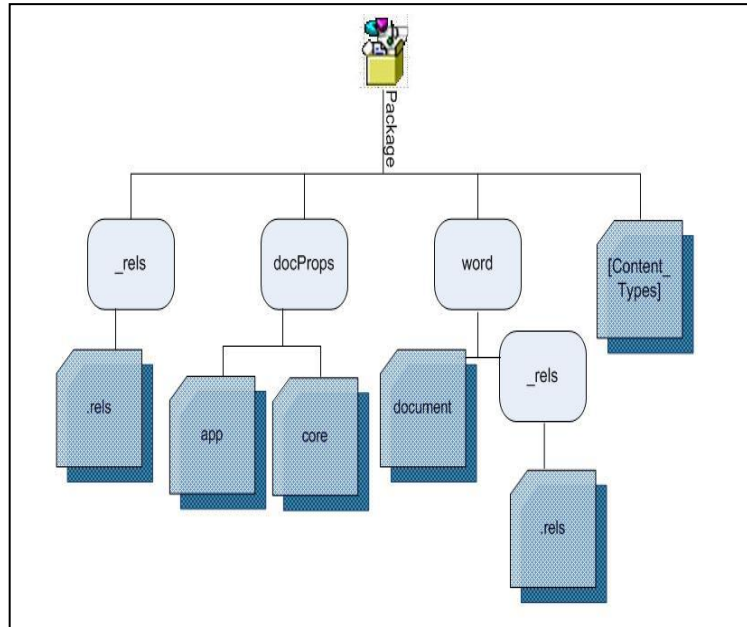


Fig. 4.1 – Package layout of MS Word 2007 document.

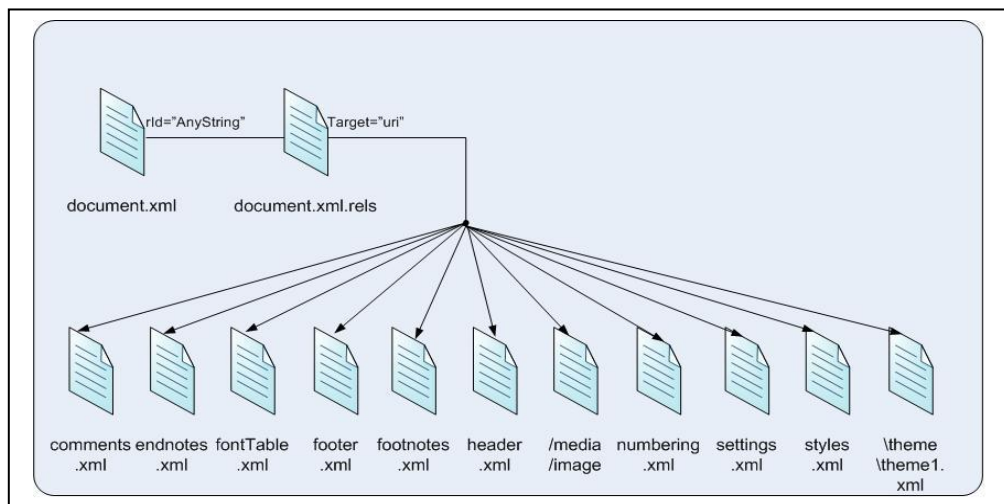


Fig. 4.2 – Preview of associated parts using relationship information with main document file.

4.2 Document Inspection – detection feature of MS Office 2007

Several types of hidden data and personal information can be saved in an MS Office 2007 document [26, 27]. This information might not be immediately visible when viewing the document in MS Office 2007 application, but it might be possible for other people to view or retrieve the information. Hidden information can include the data that MS Office

2007 application adds to a file. This enables you to collaborate on writing and editing a document with other people. It can also include information that you deliberately designate as hidden. Office documents can contain the following types of hidden data and personal information such as:

- Comments, revision marks from tracked changes, versions, and ink notations
- Document properties and personal information
- Headers, footers, and watermarks
- Hidden text
- Hidden rows, columns, and worksheets
- Invisible content
- Off-slide content
- Presentation notes
- Document server properties
- Custom XML data

The Document Inspector includes several different inspectors specific to individual Office programs. Each of these specific programs is to find and remove hidden data and personal information from the documents. The main function of detection algorithm is to find possible carrier files. Ideally, it would also provide some clues as to the steganography algorithm used to hide information in the suspect files. This enables an analyst to attempt recovery of the hidden information.

The document inspector cannot detect data, hides by using techniques listed in the data hiding section of this thesis. Therefore there is no way to detect data concealment as presented here except by using the given detection algorithm. We can take a brief overview of the query over multiple XML files using conventional approach and compare with our approach. Then we evaluate both approaches with dataset created with all listed

data hiding techniques and compare its efficiency in terms of performance. The detection techniques with logic are explained in detail in next sections.

4.3 Detection Logic of Hidden Data

The detection logic of hidden data is entirely dependent on the data hiding techniques and involves investigation of multiple XML files of OOXML document. The detection logic for each data hiding technique is different and needs to be explained first. We present our algorithm for hidden data detection and compare results with conventional algorithm and conclude the best approach. The basis of query to reveal concealed data from the document is also defined first for better understanding of the detection process.

Detail overview of each detection query is explained in detection of hidden data section. The query involves first reading document's relationship files and by using this information, it detects hidden data hides by the techniques listed. This proves that relationship information plays a vital role and becomes key information in detection process. As mentioned earlier there are two types of relationship files, package level and part level. Package level relationship file contains information such as "application used", "machine information" and "starting point of the package" for the processing editor. Whereas the part level relationship file contains information of associated parts with main document along with their location.

Furthermore, a document is combination of main document file, relationship files and their associated parts. Fig. 4.3 shows the main document file metadata code which includes associated parts and Fig. 4.4 shows part relationship file including entry of associated parts. The OOXML parser placed associated parts in the main document file

using relationship information and launched combined output of a document for user using these files as shown in Fig. 4.5.

```
<w:document>
<w:body>
  <w:p w:rsidR="00DC51FF" w:rsidRDefault="00401AD8">
    <w:r>
      <w:t>Hello</w:t>
    </w:r>
  </w:p>
  <w:p w:rsidR="0091776E" w:rsidRDefault="00401AD8">
    <w:r>
      <w:drawing>
        <wp:docPr id="1" name="Picture 0" descr="Dock.jpg" />
        <pic:nvPicPr>
          <pic:cNvPr id="0" name="Dock.jpg" />
          <pic:nvPicPr>
            <pic:blipFill>
              <a:blip r:embed="rId4" cstate="print" />
            </pic:blipFill>
          </wp:inline>
        </w:drawing>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Fig. 4.3 – XML preview of Main document file (document.xml)

```
<Relationships
xmlns="http://schemas.openxmlformats.org/pa
ckage/2006/relationships">

  <Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/offic
eDocument/2006/relationships/styles"
Target="styles.xml"/>

  <Relationship Id="rId2"
Type="http://schemas.openxmlformats.org/offic
eDocument/2006/relationships/settings"
Target="settings.xml"/>

  <Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/offic
eDocument/2006/relationships/webSettings"
Target="webSettings.xml"/>

  <Relationship Id="rId4"
Type="http://schemas.openxmlformats.org/offic
eDocument/2006/relationships/image"
Target="media/image1.jpeg"/>
</Relationships>
```

Fig. 4.4 – XML preview of Part-Relationship file (document.xml.rels)

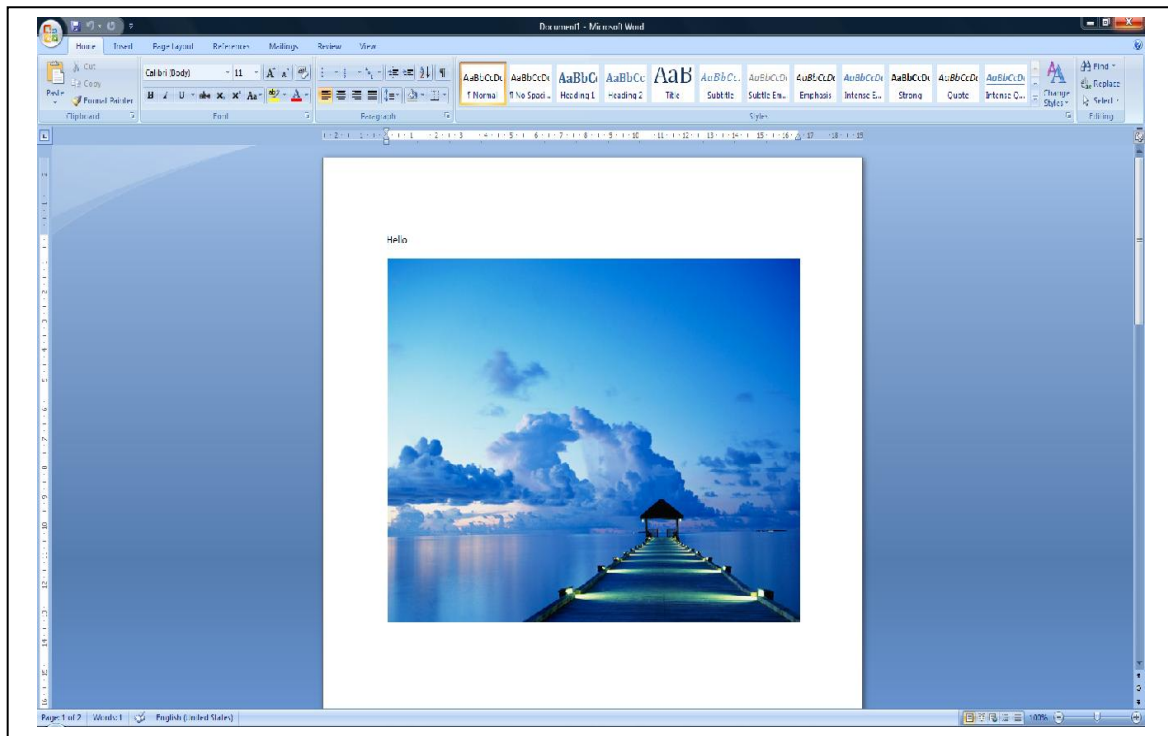


Fig. 4.5 – MS Word 2007 document output by merging main document file with associated parts. The inserted image is an image part of word document shown with text above.

4.3.1 Detecting Hidden Data using OOXML Relationship Structure

This technique requires detection logic to ensure that all package parts and relationships are verifiable against the OOXML standard. This also confirms that all parts must associate with their relevant counter parts or the main document file. Using this approach, the unverifiable and irrelevant parts are being separated by our algorithm and highlighted as hidden data for further investigation.

We start our analysis with MS Word document and extract it for further analysis. Next, the extracted files are analyzed by detection algorithm and perform first check for unknown parts and unknown relationships. This requires detection query to scan both the relationship files, package level “.rels” and part level “document.xml.rels” and identifies relationship type which is not defined in the ECMA-376 standard. This undefined relationship “Type” is being separated along with its attributes such as “Id” and “Target”. The “Target” attribute identify the unknown part and its location as shown in Fig.4.6.

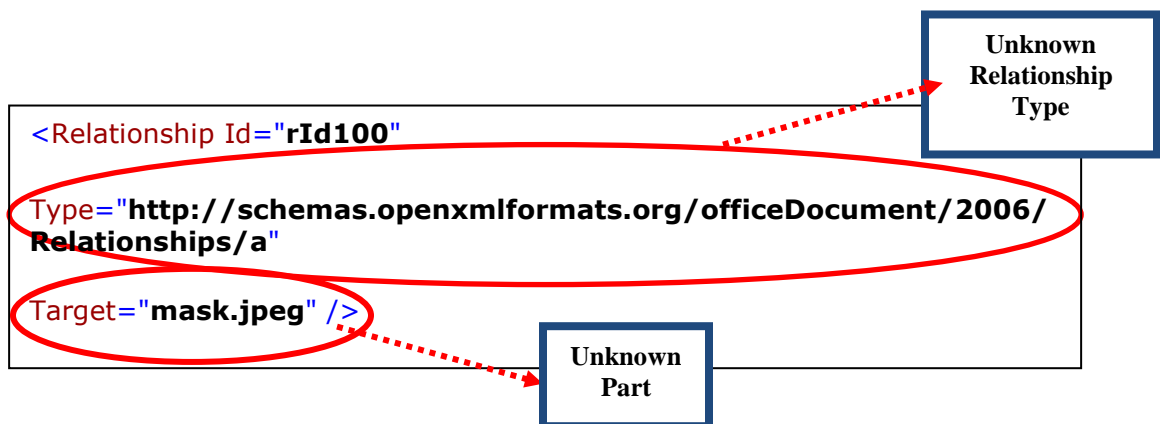


Fig. 4.6 – Unknown Relationship & Unknown Part

By performing this check the query detect hidden parts, hides using unknown parts and unknown relationship technique. The detection logic of unknown parts and unknown relationships is explicitly shown as Fig. 4.7.

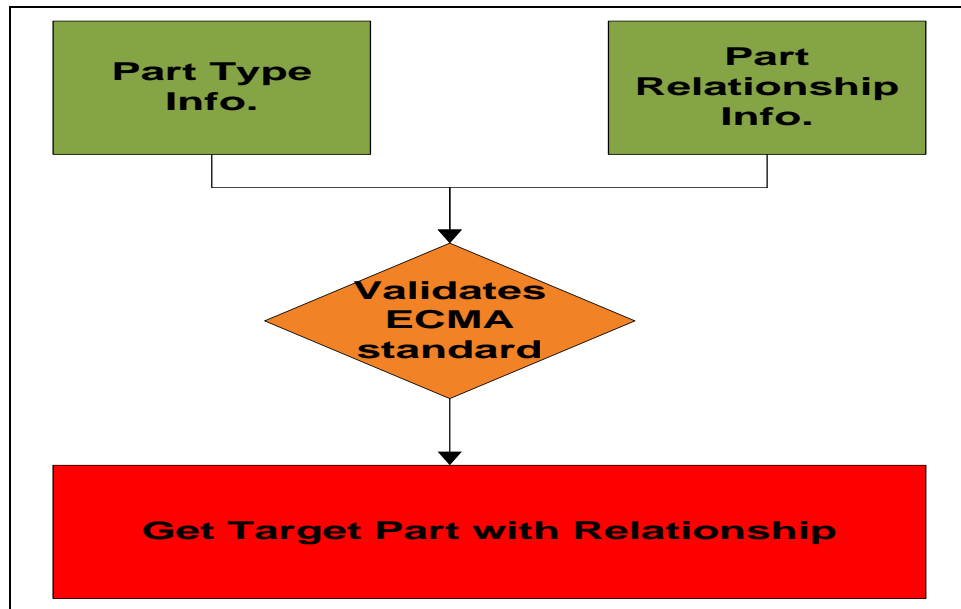


Figure 4.7 – Detection Logic for Unknown Parts & Unknown Relationships Technique

4.3.2 Detecting Hidden Data using XML Format Feature & OOXML Relationship Structure

This logic discovers the data hidden by using ignorable attribute. In the previous section we hid an image inside MS Office document using this technique. In order to detect the hidden image, the query first scans main document file “document.xml” for ignorable attribute and if found, separates all the metadata code inside ignorable attribute tag. We observed that in main document file “document.xml” contains “r:embed” attribute which carries same value of part relationship file “document.xml.rels” attribute “Id”. Next in this metadata code, the query looks for “r:embed” attribute value and matches it with part relationship file “Id” attribute value. The matched relationship “Id” attribute “Target” contains name and location of hidden image using ignorable attribute. Also

explained earlier, the OOXML parser ignores processing of metadata tags within ignorable attribute. This is worth mentioning that hidden image inside the document seems to be normal as its corresponding metadata is also being present in main document file along with its relationship information in part relationship file.

This technique considered hard to detect because it satisfies all the relationship requirements and unlikely to arouse anyone attention. The algorithm classifies ignorable attribute and its metadata tags shown as Fig. 4.8.

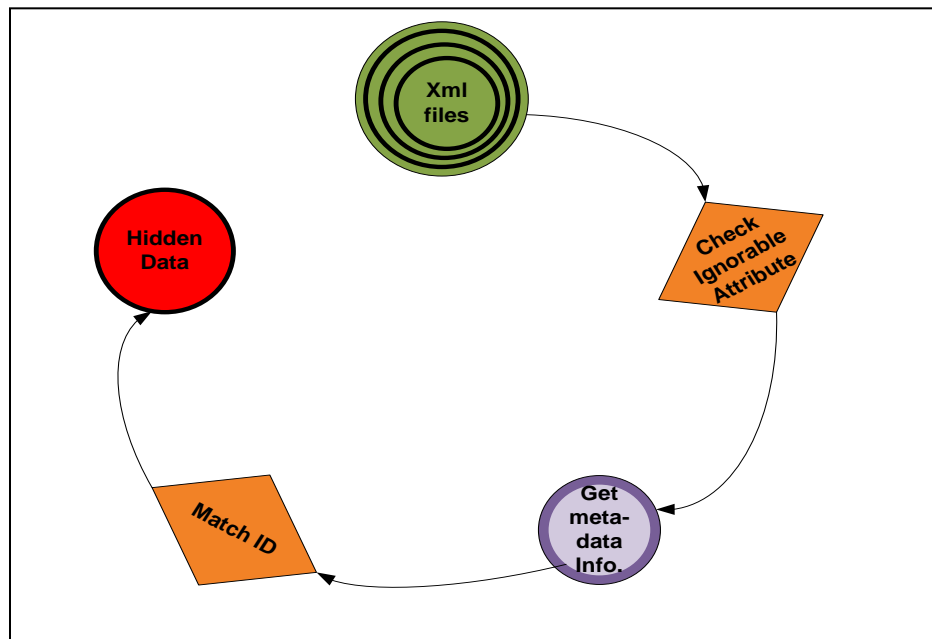


Figure 4.8 – Detection Logic for Ignorable Attribute Technique

4.3.3 Detecting Hidden Data Using OOXML Flexibility For Embedded Resource Architecture

To detect data hidden by using customXML data technique we begins with performing check to validate all the files defined in .rels files in MS Office 2007 application. The package relationship file “.rels” and part relationship file “document.xml.rels” are used to validate and ensures that all the parts present inside the package are associated with main document file. The unassociated parts with main document are considered as

hidden data and are being detected by performing this check. The query double checks the parts to see if its relationship with the main document is exists and custom XML data generated by the word document is not being detected as hidden data. Detection logic of this technique is openly shown as Fig. 4.9.

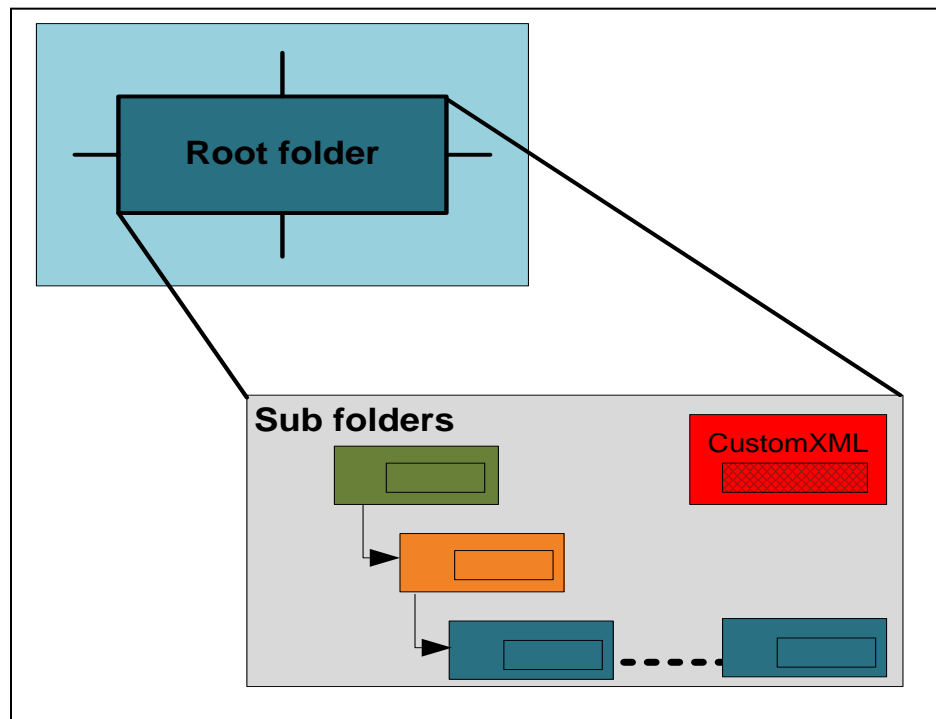


Figure 4.9 – Detection Logic for CustomXml Technique

4.3.4 Detecting Hidden Data Using OOXML File Architecture Flexibility of Swapping Parts

Fourth identified technique is using images of an OOXML document and proves unfeasible to detect using only time stamp information. In case document is unzipped and the image file is being replaced by the mischievous user, the time stamp information is being lost and new time stamp is owed by the document when zipped again. For further analysis when document is unzips again the time stamp information of all the files are set to be current system date and time. Another file “/docprops/core.xml” presents inside a package contains machine name and time information but this information is also

insufficient to guess whether the changes has been made in the image file. This information is only limited to tracking of the documents as mentioned earlier.

Based on this analysis the only way to ensure that image does not contains hidden data, currently available image stegonagraphic tools can be used for steganalysis. Detection Logic of this technique is merely shown as Fig. 4.10.

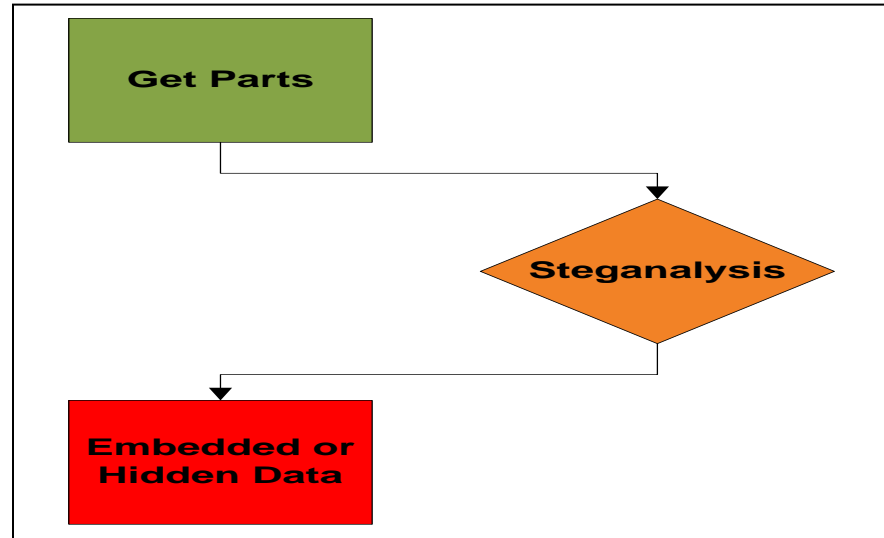


Figure 4.10 – Detection Logic for Image Steganography Technique

In next sub-section we will introduce our proposed hidden data detection algorithm. We discuss its efficient implementation and customization for MS Office 2007 documents. We choose XQuery to implement our proposed algorithm logic as it provides efficient implementation for XML based files. We compare result of approaches, a conventional detection algorithm and the proposed Office Open Multi-XML Query Algorithm (OOMXQA). After the discussion we present the results of our implemented algorithm with empirical evidence analysis which accurately predicts the performance measured and concludes with a future enhancement of our algorithm. Symbol Definitions used for the algorithm is given in Appendix A for the rest of the section.

4.4 OOMXQA

The execution of XML queries over multiple XML files is implemented inefficiently by query over one by one and this problem puzzled us for ages. We adopted a Multi-XML querying algorithm named SMXQA (Semantic Based Multi-XML Query Algorithm) to solve the problem efficiently. The adopted SMXQA is generic in nature [28]. The basic idea is surprisingly simple, but incredibly powerful. We designed and customised this algorithm for OOXML document and named as Office Open Multi-XML Query Algorithm (OOMXQA). A basic OOXML document or MS Office 2007 document without an image contains at least 12 XML files. Fig. 4.11 & Fig. 4.12 show the files of OOXML document.

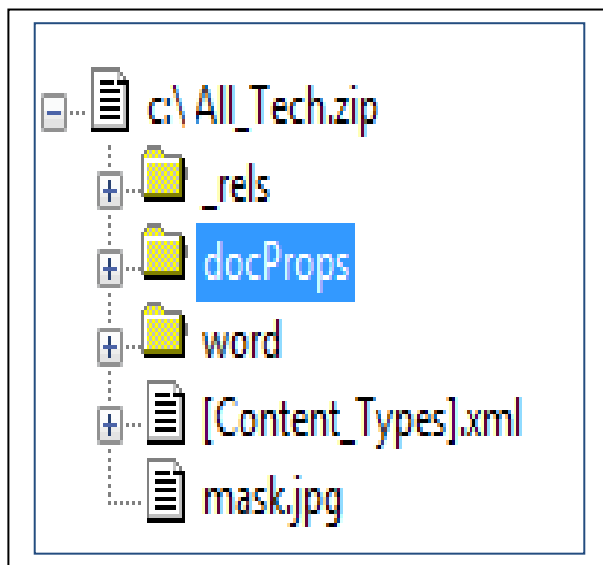


Fig. 4.11 – Office 2007 document – A document is referred as a package by Microsoft and objects inside a package are referred as parts. A package is a zip archive which comprises of multiple XML files and objects such as image or any other type format.

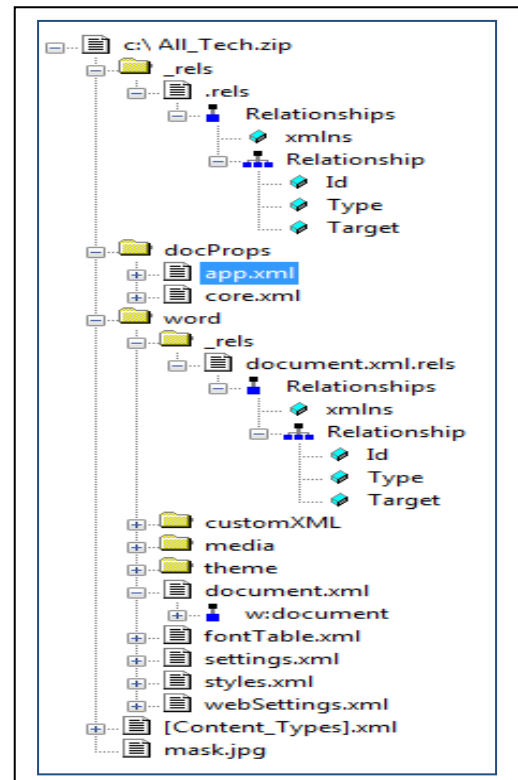


Fig. 4.12 – MS Office document with main folders and files along with the data

For integrity we only discussed the files used to reveal hidden data inside a MS Office 2007 document shown as Fig. 4.13 & Fig. 4.14. In this scenario only 3 out of 12 files

from MS Office 2007 document are being used by our algorithm. These files contain the relationship files which include the package “_rels/.rels” and part level “word/_rels/document.xml.rels” relationship files and the main document file “document.xml”.

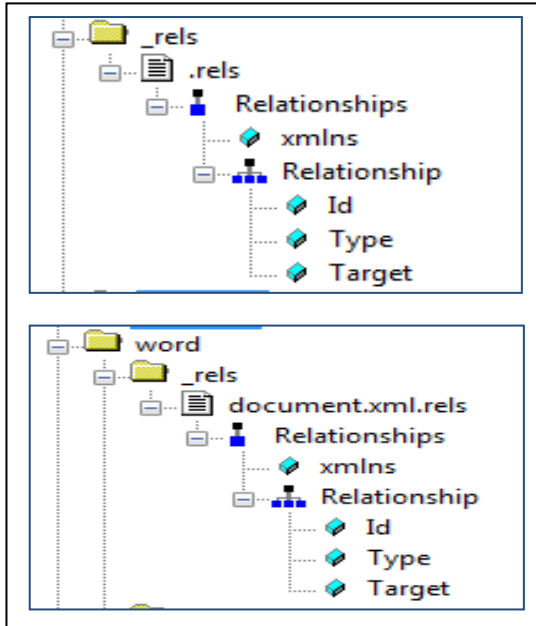


Fig. 4.13 – Package level (/_rels/.rels) and part level (/word/_rels/document.xml.rels) relationship files are listed with their elements and attributes.

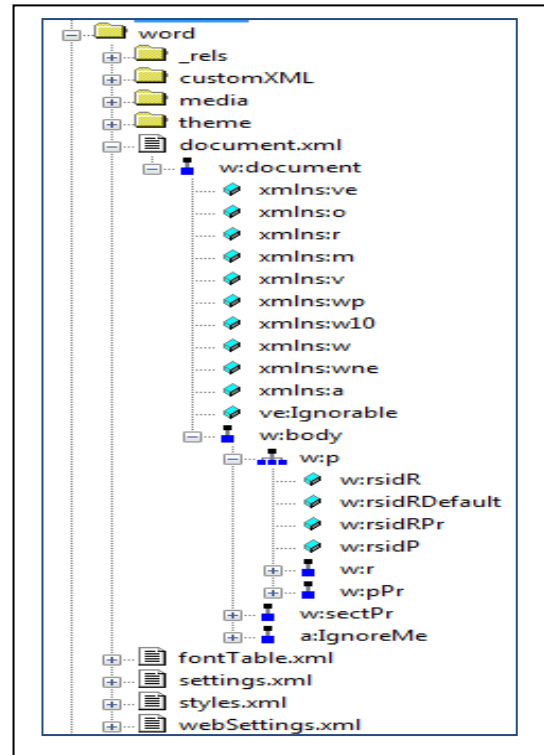


Fig 4.14 – Main document xml file snapshot

4.4.1 OOMXQA Algorithm

- 1: Create Document Type Definition (DTD) for both relationship files and main document file of MS Office 2007 document.
- 2: Identify Functional Dependency (FD) set over the DTD elements of main document file and both relationship files.
- 3: Create DTD for Global Materialized View (GMV) by specifying permissible relationship type entry from OOXML standard.
- 4: Create Global Materialized View (GMV) among these files by using the FD

set.

5: Run OOMXQA on GMV to identify conceal data in MS Office 2007

document.

4.4.2 OOMXQA Steps for OOXML Document Format

Step 1

The initiation of our algorithm begins with scanning of relationship files and main document file of MS Office 2007 document.

Step 2

Define DTD for relationship files and main document file which identifies the document structure with a list of legal elements and attributes for these files as seen in Fig. 4.15 & Fig. 4.16. For simplicity, we consider DTD as schema definition language because of its simpler design over the XML schema (XSD) [25].

Step 3

Based on the elements and attributes list the XML Functional Dependencies sets are identified over these DTD's [29, 30, 31].

We observe that both the relationship files are same in structure and hence their DTD elements and attributes are exactly the same, shown as Fig. 4.13. The functional dependency set for package and part relationship files are:

FD1: package relationships ($Id \rightarrow Type, Target$)

FD2: part relationships ($Id \rightarrow Type, Target$)

As mentioned above, Fig. 4.13 shows package level and part relationship files structure.

These files have “*Relationship*” element and “*Id*”, “*Type*” and “*Target*” as its attributes.

We call /Relationship the *scope*, Relationship/Id the *determinant*, and Relationship/Type & Target the *dependent*. We check the satisfaction, relationship by relationship (the scope).

```
<!--ELEMENT Relationships (
Relationship+ ) >
<!--ELEMENT Relationship EMPTY >
<!--ATTLIST Relationship Id
NMTOKEN #REQUIRED >
<!--ATTLIST Relationship Type
CDATA #REQUIRED >
<!--ATTLIST Relationship Target
..
```

Fig. 4.15 – DTD's of relationships files (.rels/document.xml.rels)

```
<!--ELEMENT a:avLst EMPTY >
<!--ELEMENT a:blip EMPTY >
<!--ATTLIST a:blip cstate NMTOKEN #REQUIRED >
<!--ATTLIST a:blip r:embed NMTOKEN #REQUIRED >
<!--ELEMENT a:ext EMPTY >
<!--ATTLIST a:ext cx NMTOKEN #REQUIRED >
<!--ATTLIST a:ext cy NMTOKEN #REQUIRED >
<!--ELEMENT a:fillRect EMPTY >
<!--ELEMENT a:graphic ( a:graphicData ) >
<!--ELEMENT a:graphicData ( pic:pic ) >
<!--ATTLIST a:graphicData uri CDATA #REQUIRED >
<!--ELEMENT a:graphicFrameLocks EMPTY >
<!--ATTLIST a:graphicFrameLocks noChangeAspect NMTOKEN
#REQUIRED >
<!--ELEMENT a:off EMPTY >
<!--ATTLIST a:off x NMTOKEN #REQUIRED >
<!--ATTLIST a:off y NMTOKEN #REQUIRED >
<!--ELEMENT a:prstGeom ( a:avLst ) >
<!--ATTLIST a:prstGeom prst NMTOKEN #REQUIRED >
<!--ELEMENT a:stretch ( a:fillRect ) >
<!--ELEMENT a:xfrm ( a:off, a:ext ) >
<!--ELEMENT pic:blipFill ( a:blip, a:stretch ) >
<!--ELEMENT pic:cNvPicPr EMPTY >
<!--ELEMENT pic:cNvPr EMPTY >
<!--ATTLIST pic:cNvPr id NMTOKEN #REQUIRED >
<!--ATTLIST pic:cNvPr name CDATA #REQUIRED >
<!--ELEMENT pic:nvPicPr ( pic:cNvPr, pic:cNvPicPr ) >
<!--ELEMENT pic:pic ( pic:nvPicPr, pic:blipFill, pic:spPr ) >
<!--ELEMENT pic:spPr ( a:xfrm, a:prstGeom ) >
<!--ELEMENT w:body ( w:p+, w:sectPr ) >
<!--ELEMENT w:cols EMPTY >
<!--ATTLIST w:cols w:space NMTOKEN #REQUIRED >
<!--ELEMENT w:docGrid EMPTY >
<!--ATTLIST w:docGrid w:linePitch NMTOKEN #REQUIRED >
<!--ELEMENT w:document ( w:body ) >
<!--ELEMENT w:drawing ( wp:inline ) >
<!--ELEMENT w:lang EMPTY >
<!--ATTLIST w:lang w:eastAsia NMTOKEN #REQUIRED >
<!--ELEMENT w:noProof EMPTY >
<!--ELEMENT w:p ( w:r ) >
<!--ATTLIST w:p w:rsidR NMTOKEN #REQUIRED >
<!--ATTLIST w:p w:rsidRDefault NMTOKEN #REQUIRED >
<!--ELEMENT w:pgMar EMPTY >
<!--ATTLIST w:pgMar w:bottom NMTOKEN #REQUIRED >
<!--ATTLIST w:pgMar w:footer NMTOKEN #REQUIRED >
<!--ATTLIST w:pgMar w:gutter NMTOKEN #REQUIRED >
<!--ATTLIST w:pgMar w:header NMTOKEN #REQUIRED >
<!--ATTLIST w:pgMar w:left NMTOKEN #REQUIRED >
<!--ATTLIST w:pgMar w:right NMTOKEN #REQUIRED >
<!--ATTLIST w:pgMar w:top NMTOKEN #REQUIRED >
<!--ELEMENT w:pgSz EMPTY >
<!--ATTLIST w:pgSz w:h NMTOKEN #REQUIRED >
<!--ATTLIST w:pgSz w:w NMTOKEN #REQUIRED >
<!--ELEMENT w:r ( w:t, w:rPr?, w:drawing? ) >
<!--ELEMENT w:rPr ( w:noProof, w:lang ) >
<!--ELEMENT w:sectPr ( w:pgSz, w:pgMar, w:cols, w:docGrid ) >
<!--ATTLIST w:sectPr w:rsidR NMTOKEN #REQUIRED >
<!--ATTLIST w:sectPr w:rsidSect NMTOKEN #REQUIRED >
<!--ELEMENT w:t ( #PCDATA ) >
<!--ELEMENT wp:cNvGraphicFramePr ( a:graphicFrameLocks ) >
<!--ELEMENT wp:docPr EMPTY >
<!--ATTLIST wp:docPr descr CDATA #REQUIRED >
<!--ATTLIST wp:docPr id NMTOKEN #REQUIRED >
<!--ATTLIST wp:docPr name CDATA #REQUIRED >
<!--ELEMENT wp:effectExtent EMPTY >
<!--ATTLIST wp:effectExtent b NMTOKEN #REQUIRED >
<!--ATTLIST wp:effectExtent l NMTOKEN #REQUIRED >
<!--ATTLIST wp:effectExtent r NMTOKEN #REQUIRED >
<!--ATTLIST wp:effectExtent t NMTOKEN #REQUIRED >
<!--ELEMENT wp:extent EMPTY >
<!--ATTLIST wp:extent cx NMTOKEN #REQUIRED >
<!--ATTLIST wp:extent cy NMTOKEN #REQUIRED >
<!--ELEMENT wp:inline ( wp:extent, wp:effectExtent, wp:docPr,
wp:cNvGraphicFramePr, a:graphic ) >
<!--ATTLIST wp:inline distB NMTOKEN #REQUIRED >
<!--ATTLIST wp:inline distL NMTOKEN #REQUIRED >
<!--ATTLIST wp:inline distR NMTOKEN #REQUIRED >
<!--ATTLIST wp:inline distT NMTOKEN #REQUIRED >
```

Fig. 4.16 – DTD's of main document.xml file

In relationship there are several pairs of (Id, Type, Target) values. We note that we do not consider values (rId4, <http://....././image>, image1.jpeg) and (rId5, <http://....././image>, image2.jpeg) as a pair, as the two “Id” values in the pair are different. In the same way we also do not consider relationship attributes “Id”, “Type” and “Target” pairs from different locations as they are from two locations or files. To keep this relationship entry unique in Global Materialized View (GMV) as relationship Id values are repetitive among relationship files, we add “rels ” and “dxrels” tags to classify package level and part level relationship files using relationship file names so associating this tag with “Id” attribute the same “Id” values can be distinguished in a GMV.

Like tabular approach, data inside the MS Office document files are stored in different XML sections. The parts are also stored separately inside a document and the relationship files used to carry information of main document parts. For main document file two types of functional dependency sets are needs to be defined:

FD3: (w:rsidR+w:rsidRDefault+r:embed \rightarrow w:p) for a document which contains an image.

FD4: (w:rsidR+w:rsidRDefault+w:t \rightarrow w:p) for a document with only text.

There is no homologous semantic that exists among functional dependency sets of main document DTD elements and relationship DTD elements. We observed that an element which carries identical value is “Id” attribute of part-relationship file and “r:embed” attribute of main document file for an image part. In another case where image part does not exist, no identical value is being noted among these files.

This indicates that in each w:p, triplet (w:rsidR,w:rsidRDefault,r:embed) pair wise determines w:p. The main node is “w:document” and has “w:body” as the child node. This child node further contains pairs of w:p nodes which organizes the main document in sections. According to the specification, the “rsidR” values of w:p nodes should be unique within a document: instances with the same value within a single document indicate that modifications occurred during the same editing session [10]. We notice that these values are same for pairs of “w:p” elements.

As mentioned earlier the FD attribute set identifies “w:p” element or a section of the main document file. The only linked attribute exists if a document contains a legitimate image that is “r:embed” and its value can be used to link main document with its relationship file. To identify hidden data of a document we need to scan this file completely with identical values from relationship files. If a document wouldn’t contain any legitimate image(s) then “rsidR” values can be repetitive and hard to identify “w:p” element.

In FD1, the (Id) exclusively determines its type and target. In FD3 or FD4, (w:rsidR + w:rsidRDefault + r:embed / w:t) determines its “w:p” and associated attributes. From the DTDs information we created the GMV of these multiple xml files inside a document. The DTD of GMV is shown below as Fig. 4.17.

Step 4

We define DTD for Global Materialized View (GMV) over MS Office 2007 document relationship and main document files by specifying permissible relationship types information

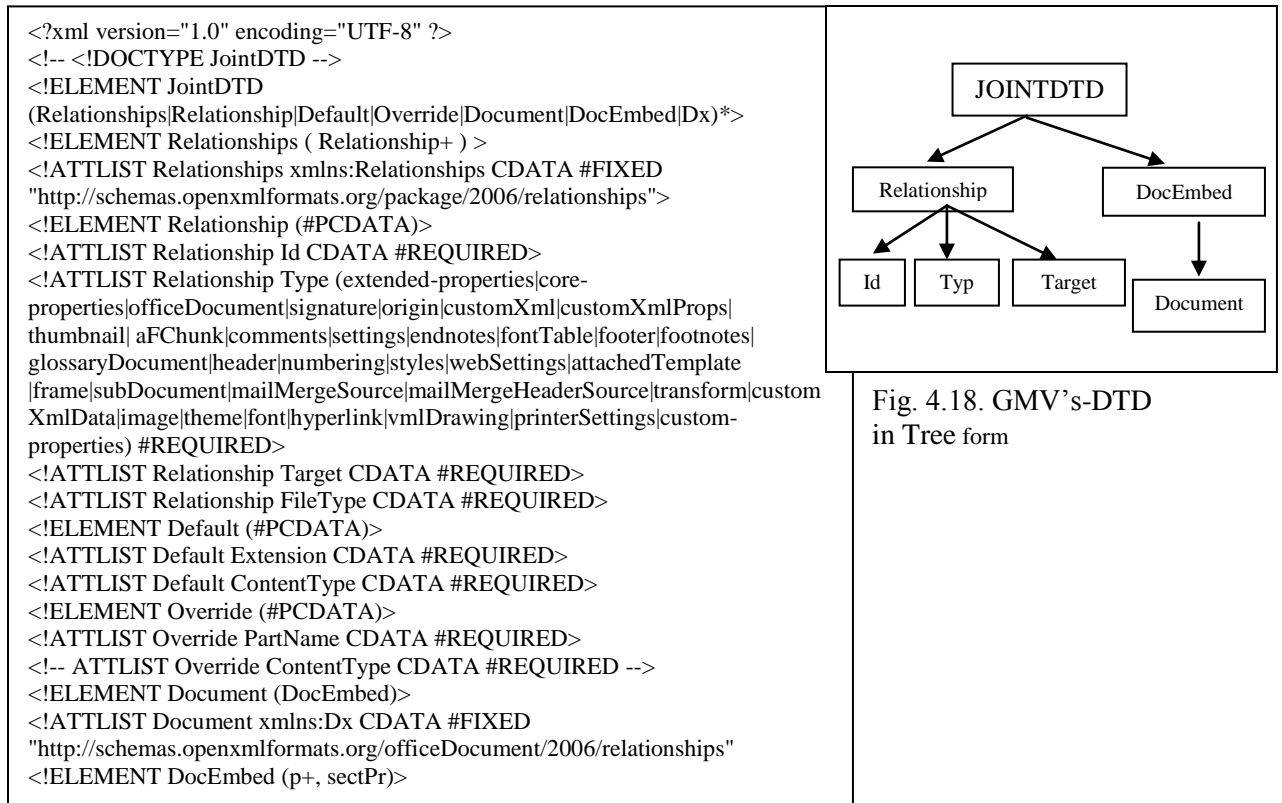


Fig. 4.18. GMV's-DTD in Tree form

Fig. 4.17. GMV's- DTD for an OOXML document with elements and attributes

We use Global Materialized View DTD's information to create GMV among files of Office 2007 document and validate their relationship types with Microsoft permissive relationship types as shown in GMV-DTD. The undefined relationship types which are not listed in OOXML standard are separated along with their target parts considered as unknown relationships and unknown parts. The rest of the hidden data using other data concealment approaches are also revealed using relationship information with main document file of MS Office 2007 document.

Step 5

Finally, a Global Materialized View (GMV) is being created using FD's information among relationship and main document files.

This information needs to detect hidden data and to create view among document and relationship files. As to make it simple for algorithm, the metadata of main document file is entirely merged with relationship files to create a Global Materialized View attached as Appendix C.

The code for both the algorithm logics to detect hidden data is being presented next. The conventional detection algorithm reads document files one by one. The second approach works more efficiently with a GMV and can read multiple files as one.

The designed query runs on XML files which are typically twig or small tree patterns. We use DTD information of XML files to tailored nodes and produce Minimum Connecting Trees (MCTs) for the query. This reduces decomposition-matching-merging process and enables query to fit targeted documents and is more efficient than conventional methods in query processing [32, 33, 34]. The XQuery code for OOMXQA technique is attached as Appendix D whereas, a conventional algorithm technique which read files one by one is attached as Appendix E. Moreover, code for creating GMV is also attached as Appendix F. We apply the proposed algorithm on created MS Word 2007 document dataset contains hidden data of all the data hiding techniques presented in data hiding section of this thesis.

The OOMXQA can be easily extended to other document formats such as Open Document Format (ODF) by customizing it according to relationship structure information of ODF standard. Report showing OOMXQA result which contains hidden data found inside MS Word 2007 document dataset is explained and presented below for better understanding.

```

<Pkg-Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships"
Id="rId100"
Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/a"
Target="mask.jpeg">
<SubPkg-Relationship Id="rId8"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image"
Target="media/ignore.jpeg"/>
<SubPkg-Relationship Id="rId300"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ignore"
Target="media/image3.jpeg"/>
<SubPkg-Relationship Id="rId200"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXmlData"
Target="/customXML/test.xml"/>
</Pkg-Relationships>

```

Fig. 4.19 – XQuery Result – Hidden files inside MS Word 2007 document dataset

In above results, the relationship Id's "rId100" and "rId300" contains relationship types ends with "a" and "ignore" are unknown relationship types and the parts associated using target attribute are unknown parts. The "mask.jpeg" and "image3.jpeg" are hidden parts identified inside MS Word 2007 document dataset using unknown relationship type information. Another relationship Id "rId8" image part "ignore.jpeg" is concealed using ignorable attribute technique in the main document file. The hidden image is searched under main document metadata including ignorable metadata created for this image which seems to be a valid image for a document. The relationship Id "rId200" is the last hidden file in this MS Word 2007 document dataset, whereas its associated target file "test.xml" is hides using CustomXML technique. This CustomXML file is not associated with main document file whereas looks legitimate as created by MS Office 2007 document.

4.5 Performance Evaluation

We performed the experiment on a dataset created with all presented data hiding techniques in this thesis. For both approaches, we executed queries for ten times as a

standard practice and their average result time has been presented. The comparison summary of both the approaches for hidden data detection functionality against each technique and the average elapsed time of the queries are as follows.

Data Hiding Technique	Conventional Algorithm [15]	Proposed Algorithm
Unknown Parts & Unknown Relationships	YES	YES
Ignorable Attribute	NO	YES
Custom XML Data	NO	YES
Image Steganography	NO	NO

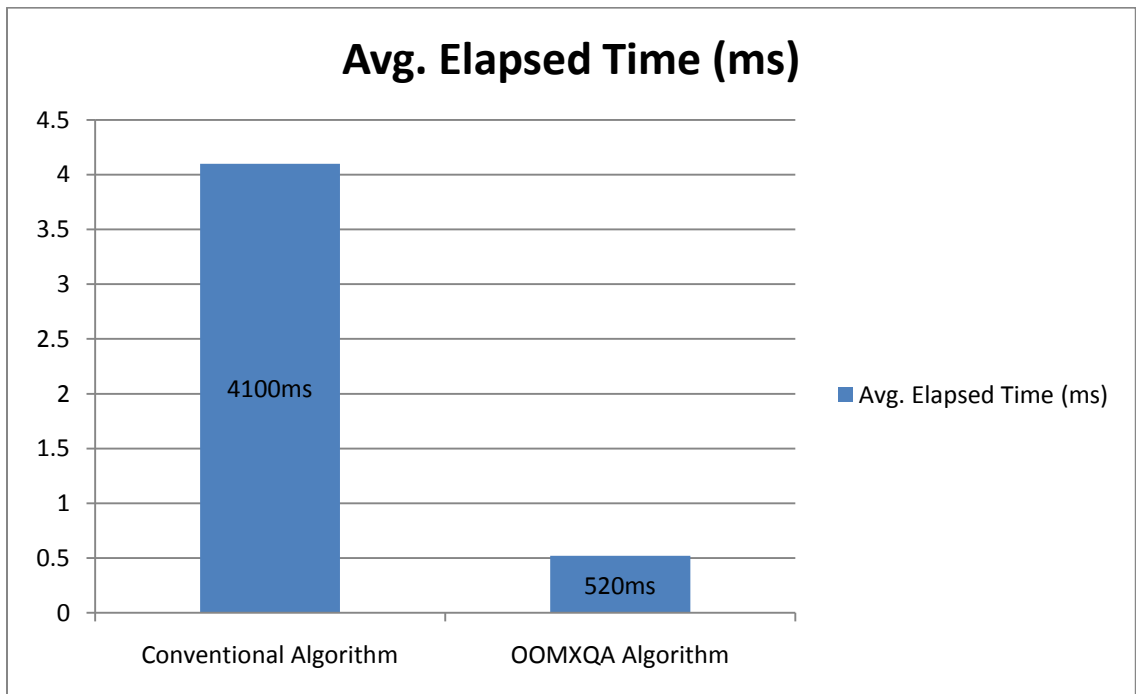


Fig. 4.20 – OOMXQA Performance Evaluation with Conventional Algorithm

In summary, the above figure clearly shows that the approach of OOMXQA is 87% less time taken than the conventional one, which is one by one XML file querying approach. This is also very robust in terms of workloads because of the created GMV. This caters real time needs as thousands of MS Office documents transferred daily over the Internet. Therefore, our algorithm achieves the expectant effect.

4.5.1 DataSets

We create five OOXML documents with hidden data, four of them with each data hiding technique and one with all data hiding techniques listed in this thesis. We compare both detection approaches with a document contains hidden data using all the data hiding techniques. This OOXML document is of 6.6MB in size and contains 12 XML and 4 image files in which 3 images and 1 XML files are hides inside a document. The conventional algorithm reads XML files one by one and takes an average of 4100ms to detect hidden data whereas proposed Office Open MultiXML Querying Algorithm (OOMXQA) only takes 520ms including query time for creating GMV. This proved that our approach is nearly 87% faster than the conventional approach as shown in Fig. 4.19. The machine used for performance test has 4GB of Ram and Intel Core 2 Duo P8400 series processor using Windows Vista. The time results for queries are performed using stylus studio 2010. The dataset is freely available by emailing authors.

4.6 Conclusion

We implemented proposed OOMXQA using XQuery language because Office 2007 documents follows OOXML standard. The designed XQuery runs on a created GMV and

reveal hidden data from the MS Office 2007 document. The GMV elements and attributes information is also presented as Fig's. 4.17 and 4.18.

We used only XQuery native code so the code and logic can be easily implemented using any programming language or platform and open for further enhancement to other document formats plus for researchers in this area. This is a first step towards development of a single tool for law enforcement officers and facilitates forensic analysis. This also caters real world needs as thousands of office documents are transferred daily over the internet among different organizations and users.

The Office 2007 application is considered to be the most widely used application. As mentioned earlier, the proposed Office Open Multi-XML Querying algorithm requires only three files information of each MS Office 2007 document to detect all the hidden files inside a document using all data hiding techniques listed in this thesis. This is highly optimized and efficient detection algorithm and the result proves that its time efficiency is more than 87% superior than conventional single file to file reading algorithm. In order to validate the experiment results, we provide XQuery code for both the detection algorithm logics to detect hidden data with all listed techniques. The conventional detection algorithm reads document files one by one. The second approach works more efficiently with a GMV of multiple XML files and can be read as one. The comparison chart is also included in terms of time efficiency performance analysis in Fig. 4.19.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

Steganography is a fascinating and effective method of hiding data that has been used throughout history. There are methods that can be employed to uncover such devious tactics, but the first step is an awareness that such methods do exist. There are many good reasons to use steganography in OOXML documents, regardless of the fact that the technology is easy to use and difficult to detect.

Microsoft Office suite is the most widely used for creating documents and proved to be the most at risk and so they would serve as a great target for this work. Recently Microsoft announced OOXML format for its MS Office 2007 suite. This thesis proposes the possibility of hiding data in OOXML documents by using the techniques listed in data hiding section, and one can see that there exists a large selection of approaches of hiding data with different strong and weak points respectively.

For steganalysis, the use of forensic tools is very important in the digital investigation process. This requires a tool which must adapt technology of cover media being used for hiding data. For example, if a suspect hides something in the OOXML document, an investigator should have the appropriate tools embedding OOXML file format relationships technology to detect hidden data.

In general, analysis of hidden data in OOXML document is divided into two phases. The first phase is to identify whether there is hidden data by searching for anomaly. For example, the analysis of hidden data in OOXML document zip archive, it is unlikely that

MS Office 2007 document inspector to detect. This is suspicious and requires further analysis. The second phase is to recover the hidden data files. Since hidden data is usually stored inside OOXML document – zip archive, with other parts or metadata, it is hard to recover them. Recovery is particularly challenging if suspects data satisfies the relationships order of an OOXML document.

There is no specific forensic analysis tool that checks for hidden data in OOXML documents except tool based on given OOMXQA detection algorithm. While the analysis techniques discussed in this thesis are able to detect and recover the hidden data, it is time consuming without automated tools.

The data hiding techniques that have been discussed in this thesis are just a fraction of possible ways to hide data. There are always new techniques to hide data and the art of data hiding highly depends on suspect's creativity. One of the difficulties of analyzing OOXML document format is that it is flexible and can thus support many options. As a result, there are many possible ways to hide data. In addition, without published specifications, it is hard to guess which value combinations are valid and which are unknown.

It is certain that the use of steganography will continue to increase and thus will be a growing hurdle for law enforcement and counterterrorism activities. Ignoring the significance of steganography because of the lack of statistics is "security through denial" and is not a good strategy. Thus for an agent to decide on which steganographic algorithm to use, he would have to decide on the type of application he want to use the algorithm for and willing to compromise on some features to ensure the security of others.

5.2 Summary

This thesis has investigated a novel approach to document steganography which provided enhancements to the current available steganographic techniques. The focus is on the shift from proprietary formats to XML based formats which has been adopted by MS Office 2007 known as OOXML for document representation. Earlier versions of MS Office documents used binary formats which is a compound document format.

A comprehensive review of steganographic work in document steganography especially for OOXML documents was discussed and classified into five main categories. These includes data hiding using OOXML relationship structure, data hiding using XML format features, data hiding using XML format features and OOXML Relationship Structure, data hiding using OOXML flexibility for embedded resource architecture and data hiding using OOXML flexibility of swapping parts.

It was observed that the current steganalysis algorithm rely heavily on the conventional searching techniques, whereas OOXML document comprises of several XML files and binary objects. The performance of conventional detection algorithm is relatively low as it reads these files one by one and fails to identify hidden data for all the techniques listed in this thesis. The proposed OOMXQA detection algorithm reads these files as one by creating global materialized view based on functional dependency sets identified and proves that it is 87% faster than conventional algorithm and also works for all the techniques listed. The results were promising and outperformed relevant methods. This caters real time needs as thousands of documents are transferred daily over the net.

The OOMXQA algorithm is design and developed using XQuery code and enables to use with steganalysis tools used in the industry or by other researchers. This is also a first

step towards development of a single tool for law enforcement officers and further facilitation of forensic analysis.

5.3 Future Research

The ease of manipulation and transmission, communication means are shifting from paper documents to e-documents globally. Millions of documents are transferred globally over the net and until now, there is no system yet which can monitor the manipulation of documents and detect steganography in documents. Based on the work done in this thesis, the following recommendations are made for further work in this area:

- The flexibility of OOXML document structure can motivate further possible techniques of data hiding in OOXML documents.
- Analyzing OOXML document metadata, which could be useful for querying purposes such as: unique reference, date and time stamp, owner and machine name etc. This information facilitated unauthorized tampering performed by mischievous user without leaving any perceptible traces can stand up in a court as reliable evidence [10].
- Applying the identified data hiding techniques to similar word processing applications using XML format such as ODF documents and observe their behavior and propose modified set of rules based on their relationship structure to be used for data hiding and detection process.
- The detection algorithm can be further enhances by adding the functionality of cryptanalysis with possible encoding/encryption methods supported by OOXML documents. Moreover, adding together of parallel searching feature for multiple

documents such as batch processing caters real world needs and permits steganalysis for hundreds of documents in no time.

- To develop a single document steganalysis framework for forensic analysis by law enforcement officers.

References:

1. R. Neijts and M. Semilof, "Steganography," online at http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci213717,00.html, last accessed April 14, 2011.
2. W. Bender, et al. "Applications for data hiding," *IBM Systems Journal*, vol. 39, no. 3/4, pp. 547-68, 2000.
3. J.C. Judge, "Steganography: Past, Present, Future," SANS white paper, November 30, 2001.
4. A. Cheddad, J. Condell, K. Curran and P. Mc Kevitt, "A secure and improved self embedding algorithm to combat digital document forgery," *Signal Processing*, vol. 89, no. 12, pp. 2324-32, December 2009.
5. C. Hosmer, "Discovering hidden evidence," *Journal of Digital Forensic Practice*, vol. 1, no. 1, pp. 47-56, January 2006.
6. US Department of Justice, Office of Justice Programs, National Institute of Justice online at <http://www.ojp.usdoj.gov/nij/topics/forensics/evidence/digital/analysis/steganography.htm#steganography>, last accessed April 14, 2011.
7. A. Cheddad, J. Condell, K. Curran and P. Mc Kevitt, "Securing information content using new encryption method and steganography," in *Proceedings of the 3rd IEEE International Conference on Digital Information Management*. London, UK, pp. 563-568, November 2008.
8. P. Hayati, V. Potdar and E. Chang, "A survey of steganographic and steganalytic tools for the digital forensic investigator," in *Proceedings of the Workshop of Information Hiding and Digital Watermarking in conjunction with IFIPTM*, New Brunswick, Canada, July 2007.
9. A. Cheddad, J. Condell, K. Curran and P. Mc Kevitt, "Review: Digital image steganography: Survey and analysis of current methods," *Signal Process.* vol. 90, no. 3, pp. 727-752, March 2010.
10. S.L. Garfinkel and J.J. Migletz, "New XML-Based Files Implications for Forensics," *Security & Privacy, IEEE*, vol. 7, no. 2, pp. 38-44, March-April 2009.
11. F. Rice, "Open XML file formats," *Microsoft Corporation*, online at <http://msdn.microsoft.com/en-us/library/aa338205.aspx>, last accessed April 14, 2011.
12. N. Provos and P. Honeyman, "Hide and seek: an introduction to steganography," *Security & Privacy, IEEE*, vol.1, no. 3, pp. 32- 44, May-June 2003.
13. T. Ngo, "Office Open XML Overview," *ECMA TC45* white paper, online at <http://www.ecma->

- international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf, last accessed April 14, 2011.
14. "ECMA OOXML documentation," online at <http://www.ecma-international.org/publications/standards/Ecma-376.htm>, last accessed April 14, 2011.
 15. B. Park, J. Park and S. Lee, "Data Concealment & Detection in Microsoft Office 2007 files," *Digital Investigation*, vol. 5, no. 3/4, pp. 104-114, March 2009.
 16. B.D. Zeve, "Readying Your Firm for Office 2010 An Insider's Report," *ILTA White Paper April 2010, Microsoft Corporation*, online at <http://docs.google.com/viewer?a=v&q=cache:-H7ugyZ3DyUJ:download.microsoft.com/download/E/4/F/E4F6A507-3CDD-4DB3-9395-612F474F874B/Readying%2520Your%2520Firm%2520for%2520Office%25202010.pdf+Readying+Your+Firm+for+Office+2010+An+Insider%E2%80%99s+Report&hl=en&gl=ca&pid=bl&srcid=ADGEESjenYaKK4OFGxYzGsfz0se4gQAWhGWWO4tRv2ukhtTJkp9XMNUgUH9jGajKChZF6FZIdXO1v0tPKn30dWP6PLuvxm3DPJGjozuUI0jRPo5QcxhQQafUktdPyXCqGCn2ffVzDjIf&sig=AHIEtbT0gvXR8eByuyhKpyeZCp-wYNrtAQ>, last accessed April 19, 2011.
 17. www.wikipedia.org/steganography, last accessed April 14, 2011.
 18. N. Provos and P. Honeyman, "Detecting steganographic content on the Internet," *Technical report, University of Michigan. August 31, 2001*.
 19. A. Castiglione, A. De Santis and C. Soriente, "Taking advantages of a disadvantage: Digital forensics and steganography using document metadata," *Journal of Systems and Software*, vol. 80, no. 5, pp. 750-764, May 2007.
 20. S. Pringle, "CustomXML? This Custom XML!!!," *Code Counsel Blogs – Wouter*, online at <http://blogs.code-counsel.net/Wouter/Lists/Posts/Post.aspx?ID=43>, last accessed at April 14, 2011.
 21. N.F. Johnson and S. Jajodia, "Exploring steganography: Seeing the unseen," *Computer*, vol. 31, no. 2, pp. 26-34, February 1998.
 22. R.H. Wiggins II, I, MD, H.C. Davidson, MD, H.R. Harnsberger, MD, J.R. Lauman, BS and P.A. Goede, BS. "Image File Formats: Past, present, & Future," *The journal of continuing medical education in radiology*, vol. 21, pp. 789-798, February 2001.
 23. M.F. Fernandez, W.C. Tan and D. Suciu, "SilkRoute: trading between relations and XML," *Computer Networks*, vol. 33 no. 1-6, pp. 723-745, June 2000.
 24. X. Zhao, J. Xin and E. Zhang, "XML Functional Dependency and Schema Normalization," in *Proceedings of the Ninth International Conference on Hybrid Intelligent Systems (HIS '09)*, vol. 3, pp. 307-312, IEEE Computer Society, Washington, DC, USA, 2009.
 25. M.S. Shahriar and J. Liu, "On Defining Functional Dependency for XML," in *Proceedings of the IEEE International Conference on Semantic Computing (ICSC*

- '09), California USA, pp. 595-600, IEEE Computer Society, Washington, DC, USA, September 2009.
26. Microsoft, "Remove personal or hidden information," online at <http://office.microsoft.com/en-us/word/HP051901021033.aspx>, last accessed April 14, 2011.
 27. Microsoft, "The remove hidden data tool for Office 2003 and Office XP," online at <http://support.microsoft.com/kb/834427>, last accessed April 14, 2011.
 28. B. Zhang and F. Ye, "Semantics Based Multi-XML Query Algorithm," *International Conference on Computer Science and Software Engineering, 2008*, vol. 4, pp. 708-712, IEEE Computer Society, Washington, DC, USA, December 2008.
 29. T. Lv and P. Yan, "A Survey Study on XML Functional Dependencies," *The First International Symposium on Data, Privacy, and E-Commerce, 2007. (ISDPE 2007)*, Chengdu China, pp.143-145, November 2007.
 30. J. Song, W. Zhao, X. Zhang and G. Liu, "Multivalued Dependencies for XML Documents with DTDs," *International Conference on Web Information Systems and Mining, 2009 (WISM 2009)*, Shanghai China, pp. 284-288, November 2009.
 31. J. Liu, M. Vincent and C. Liu, "Local XML functional dependencies," in *Proceedings of the 5th ACM international workshop on Web information and data management (WIDM '03)*. ACM, New York, USA, pp. 23-28, New Orleans, Louisiana, USA, 2003.
 32. V. Hristidis, N. Koudas, Y. Papakonstantinou and D. Srivastava, "Keyword proximity search in XML trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 4, pp. 525- 539, April 2006.
 33. J.T. Yao and M. Zhang, "A Fast Tree Pattern Matching Algorithm for XML Query," in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence 2004 (WI 2004)*, Beijing China, pp. 235- 241, September 2004.
 34. K.M. Sook and K.Y. Hae, "Ontology-DTD Matching Algorithm for Efficient XML Query," *Fuzzy Systems and Knowledge Discovery 2005*, vol. 3614, pp. 484, 2005.

Appendix A – Functional Dependency in XML Documents

DTD & XML Document Tree:

DTD $D = (E_c, E_s, A, M, N, r)$, where

- E_c is the finite set of complex elements
- E_s is the finite set of simple elements
- A is the finite set of attribute names
- M is the map from E_c to element type: $\forall e \in E_c, M(e)$ is a regular expression
 $M(e) = \mathcal{E}|e'|M(e)|M(e)|M(e), M(e)|M(e)^*$
- Where \mathcal{E} denotes null string, $e' \in E_c \cup E_s$, and $"|", ", ", " * "$ denote union, connection and Kleene closure;
- N denotes the map from E_c to the attribute set, for $e \in E_c$, any $M(e)$ not \mathcal{E} or $N(e)$ not \emptyset
- $r \in E_c$ is another different element name, $\forall e \in E_c$, r is the only flag in E_c which is not in the alphabet of $M(e)$.

XML Tree $T = (V, M, ele, att, num, str, root)$, where

- V is the node set of XML tree
- M is the map from V to $E_c \cup E_s \cup A$, it can get the node type in V , if $M(v) \in E_c$, then a node $v \in V$ is called a complex element node else then it is called a simple element node. If $M(v) \in E_s$, then it is called attribute node.
- ele and att are the functions defined on the set of complex elements: for every $v \in V$, if $M(v) \in E_c$ then $ele(v)$ denotes a list of element nodes and $att(v)$ denotes a set of different attribute names
- num denotes the function of node number from every attribute or element
- str denotes the function of string value from every attribute or element, if $M(v) \in E_c$, $str(v)$ is null, and if $M(v) \in E_c \cup A$, $str(v)$ is the document contained root denotes the only root node.

Path Language:

Path Expression on DTD $D = (E_c, E_s, A, M, N, r)$,

a simple path is defined as $l_1/ l_2/... l_n$, where $l_i \in E_c$ ($i=1, ..., n-1$) and $l_n \in E_c \cup E_s \cup A$ is called as a sequence of path. $l_1 = M(r)$, $l_i = M(l_{i-1})$ for $i [2, n-1]$ and $l_n = M(l_{n-1}) \cap N(l_{i-1})$.

Whereas a complex path is in the form of $l_1/ l_2/... l_n$, where $l_i \in E_c \cup \{ /\}$ ($i=1, ..., n-1$) and $l_n \in E_c \cup E_s \cup A$. Symbol $" / "$ represent the Kleene closure of the wildcard and can match any label l .

XML Functional Dependency:

XFD Constraint language on DTD D , functional dependency for XML (XFD) is ϕ over D has the form $P: Q: X_1, \dots, X_n \rightarrow Y_1, \dots, Y_m$ where

- $P \in \text{paths}(D)$ is downward context path starting from the root, which identifies the scope of ϕ over D . If $P \neq r$ and $P \neq \epsilon$ (where ϵ means empty path), then ϕ is called a local XFD, which means that the scope of ϕ is the sub-tree rooted at P and last $(P) \in E_1$. Otherwise ϕ is called a global XFD, which means the scope of ϕ is the whole D or at the root and simplified as $Q: X_1, \dots, X_n \rightarrow Y_1, \dots, Y_m$.
- Q is called downward target path, where $Q \in \text{paths}(D)$ and $P \subseteq Q$.
- X_1, \dots, X_n is the left path of ϕ (Left Hand Side, LHS) and Y_1, \dots, Y_m is the right path of (Right Hand Side, RHS), which is a non-empty subsets of paths (D) rooted at $[[Q]]$.

Appendix B – Document Type Definition (DTD)

DTD of .rels & document.xml.rels

```
<!ELEMENT Relationship EMPTY >  
<!ATTLIST Relationship Id NMTOKEN #REQUIRED >  
<!ATTLIST Relationship Target CDATA #REQUIRED >  
<!ATTLIST Relationship Type CDATA #REQUIRED >
```

```
<!ELEMENT Relationships ( Relationship+ ) >
```

NOTE: Both relationship files are same in structure hence their DTD elements and attributes are same. We listed DTD of package level relationship file as sub-package level relationship file also be the same.

DTD of document.xml

```
<!ELEMENT a:avLst EMPTY >
```

```
<!ELEMENT a:blip EMPTY >  
<!ATTLIST a:blip cstate NMTOKEN #REQUIRED >  
<!ATTLIST a:blip r:embed NMTOKEN #REQUIRED >
```

```
<!ELEMENT a:ext EMPTY >  
<!ATTLIST a:ext cx NMTOKEN #REQUIRED >  
<!ATTLIST a:ext cy NMTOKEN #REQUIRED >
```

```
<!ELEMENT a:fillRect EMPTY >
```

```
<!ELEMENT a:graphic ( a:graphicData ) >
```

```
<!ELEMENT a:graphicData ( pic:pic ) >  
<!ATTLIST a:graphicData uri CDATA #REQUIRED >
```

```
<!ELEMENT a:graphicFrameLocks EMPTY >  
<!ATTLIST a:graphicFrameLocks noChangeAspect NMTOKEN #REQUIRED >
```

```
<!ELEMENT a:off EMPTY >  
<!ATTLIST a:off x NMTOKEN #REQUIRED >  
<!ATTLIST a:off y NMTOKEN #REQUIRED >
```

```
<!ELEMENT a:prstGeom ( a:avLst ) >  
<!ATTLIST a:prstGeom prst NMTOKEN #REQUIRED >
```

```

<!ELEMENT a:stretch ( a:fillRect ) >

<!ELEMENT a:xfrm ( a:off, a:ext ) >
<!ELEMENT pic:blipFill ( a:blip, a:stretch ) >

<!ELEMENT pic:cNvPicPr EMPTY >

<!ELEMENT pic:cNvPr EMPTY >
<!ATTLIST pic:cNvPr id NMTOKEN #REQUIRED >
<!ATTLIST pic:cNvPr name CDATA #REQUIRED >

<!ELEMENT pic:nvPicPr ( pic:cNvPr, pic:cNvPicPr ) >

<!ELEMENT pic:pic ( pic:nvPicPr, pic:blipFill, pic:spPr ) >

<!ELEMENT pic:spPr ( a:xfrm, a:prstGeom ) >

<!ELEMENT w:body ( w:p+, w:sectPr ) >

<!ELEMENT w:cols EMPTY >
<!ATTLIST w:cols w:space NMTOKEN #REQUIRED >

<!ELEMENT w:docGrid EMPTY >
<!ATTLIST w:docGrid w:linePitch NMTOKEN #REQUIRED >

<!ELEMENT w:document ( w:body ) >

<!ELEMENT w:drawing ( wp:inline ) >

<!ELEMENT w:lang EMPTY >
<!ATTLIST w:lang w:eastAsia NMTOKEN #REQUIRED >

<!ELEMENT w:noProof EMPTY >

<!ELEMENT w:p ( w:r ) >
<!ATTLIST w:p w:rsidR NMTOKEN #REQUIRED >
<!ATTLIST w:p w:rsidRDefault NMTOKEN #REQUIRED >
<!ELEMENT w:pgMar EMPTY >
<!ATTLIST w:pgMar w:bottom NMTOKEN #REQUIRED >
<!ATTLIST w:pgMar w:footer NMTOKEN #REQUIRED >
<!ATTLIST w:pgMar w:gutter NMTOKEN #REQUIRED >
<!ATTLIST w:pgMar w:header NMTOKEN #REQUIRED >
<!ATTLIST w:pgMar w:left NMTOKEN #REQUIRED >
<!ATTLIST w:pgMar w:right NMTOKEN #REQUIRED >
<!ATTLIST w:pgMar w:top NMTOKEN #REQUIRED >

```

```

<!ELEMENT w:pgSz EMPTY >
<!ATTLIST w:pgSz w:h NMTOKEN #REQUIRED >
<!ATTLIST w:pgSz w:w NMTOKEN #REQUIRED >
<!ELEMENT w:r ( w:t, w:rPr?, w:drawing? ) >

<!ELEMENT w:rPr ( w:noProof, w:lang ) >

<!ELEMENT w:sectPr ( w:pgSz, w:pgMar, w:cols, w:docGrid ) >
<!ATTLIST w:sectPr w:rsidR NMTOKEN #REQUIRED >
<!ATTLIST w:sectPr w:rsidSect NMTOKEN #REQUIRED >
<!ELEMENT w:t ( #PCDATA ) >

<!ELEMENT wp:cNvGraphicFramePr ( a:graphicFrameLocks ) >

<!ELEMENT wp:docPr EMPTY >
<!ATTLIST wp:docPr descr CDATA #REQUIRED >
<!ATTLIST wp:docPr id NMTOKEN #REQUIRED >
<!ATTLIST wp:docPr name CDATA #REQUIRED >

<!ELEMENT wp:effectExtent EMPTY >
<!ATTLIST wp:effectExtent b NMTOKEN #REQUIRED >
<!ATTLIST wp:effectExtent l NMTOKEN #REQUIRED >
<!ATTLIST wp:effectExtent r NMTOKEN #REQUIRED >
<!ATTLIST wp:effectExtent t NMTOKEN #REQUIRED >

<!ELEMENT wp:extent EMPTY >
<!ATTLIST wp:extent cx NMTOKEN #REQUIRED >
<!ATTLIST wp:extent cy NMTOKEN #REQUIRED >

<!ELEMENT wp:inline ( wp:extent, wp:effectExtent, wp:docPr,
wp:cNvGraphicFramePr, a:graphic ) >
<!ATTLIST wp:inline distB NMTOKEN #REQUIRED >
<!ATTLIST wp:inline distL NMTOKEN #REQUIRED >
<!ATTLIST wp:inline distR NMTOKEN #REQUIRED >
<!ATTLIST wp:inline distT NMTOKEN #REQUIRED >

```

Appendix C – Global Materialized View (GMV)

```
<?xml version="1.0" ?>
<GlobalView xmlns="http://schemas.openxmlformats.org/package/2006/relationships">

  <rels>
    <Relationship Id="rId3"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties" Target="docProps/app.xml" />

    <Relationship Id="rId2"
      Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties" Target="docProps/core.xml" />

    <Relationship Id="rId1"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument" Target="word/document.xml" />

    <Relationship Id="rId100"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/Relationships/a" Target="mask.jpeg" />
  </rels>

  <dxrels>
    <Relationship Id="rId3"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml" />

    <Relationship Id="rId7"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml" />

    <Relationship Id="rId2"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml" />

    <Relationship Id="rId1"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml" />

    <Relationship Id="rId6"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml" />

    <Relationship Id="rId5"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image2.jpeg" />

    <Relationship Id="rId4"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image1.jpeg" />

    <Relationship Id="rId8"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/raffay.jpeg" />
  </dxrels>
</GlobalView>
```



```

    <Relationship Id="rId300"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/raf
      fay" Target="media/image3.jpeg" />
    <Relationship Id="200"
      Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/cu
      stomXmlData" Target="/customXML/test.xml" />
  </dxrels>

  <docxml>

    <w:body xmlns:a="http://schemas.openxmlformats.org/MyExtension/p1"
      xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
      xmlns:o="urn:schemas-microsoft-com:office:office"
      xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
      xmlns:v="urn:schemas-microsoft-com:vml"
      xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
      xmlns:w10="urn:schemas-microsoft-com:office:word"
      xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
      xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingD
      rawing">
      - <w:p w:rsidR="00DC51FF" w:rsidRDefault="00401AD8">
      - <w:r>
        <w:t>Hello</w:t>
        </w:r>
      </w:p>
      - <w:p w:rsidR="0091776E" w:rsidRDefault="00401AD8">
      - <w:r>
      - <w:rPr>
        <w:noProof />
      </w:rPr>
      - <w:drawing>
      - <wp:inline distT="0" distB="0" distL="0" distR="0">
        <wp:extent cx="5943600" cy="4457700" />
        <wp:effectExtent l="19050" t="0" r="0" b="0" />
        <wp:docPr id="1" name="Picture 0" descr="Dock.jpg" />
      - <wp:cNvGraphicFramePr>
        <a:graphicFrameLocks
          xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
          noChangeAspect="1" />
        </wp:cNvGraphicFramePr>
      - <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
      - <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
      - <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
      - <pic:nvPicPr>
        <pic:cNvPr id="0" name="Dock.jpg" />
        <pic:cNvPicPr />
      </pic:nvPicPr>
      - <pic:blipFill>
        <a:blip r:embed="rId4" cstate="print" />
      - <a:stretch>
        <a:fillRect />
      </a:stretch>
      </pic:blipFill>
      - <pic:spPr>
      - <a:xfrm>

```

```

<a:off x="0" y="0" />
<a:ext cx="5943600" cy="4457700" />
</a:xfrm>
- <a:prstGeom prst="rect">
  <a:avLst />
  </a:prstGeom>
  </pic:spPr>
  </pic:pic>
  </a:graphicData>
  </a:graphic>
  </wp:inline>
  </w:drawing>
  </w:r>
  </w:p>
- <w:p w:rsidR="00401AD8" w:rsidRPr="0091776E" w:rsidRDefault="0091776E"
  w:rsidP="0091776E">
- <w:pPr>
- <w:tabs>
  <w:tab w:val="left" w:pos="1155" />
  </w:tabs>
  </w:pPr>
- <w:r>
- <w:rPr>
  <w:noProof />
  </w:rPr>
  <w:lastRenderedPageBreak />
- <w:drawing>
- <wp:inline distT="0" distB="0" distL="0" distR="0">
  <wp:extent cx="5943600" cy="4457700" />
  <wp:effectExtent l="19050" t="0" r="0" b="0" />
  <wp:docPr id="2" name="Picture 1" descr="Garden.jpg" />
- <wp:cNvGraphicFramePr>
  <a:graphicFrameLocks
    xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
    noChangeAspect="1" />
  </wp:cNvGraphicFramePr>
- <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
- <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
- <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
- <pic:nvPicPr>
  <pic:cNvPr id="0" name="Garden.jpg" />
  <pic:cNvPicPr />
  </pic:nvPicPr>
- <pic:blipFill>
  <a:blip r:embed="rId5" cstate="print" />
- <a:stretch>
  <a:fillRect />
  </a:stretch>
  </pic:blipFill>
- <pic:spPr>
- <a:xfrm>
  <a:off x="0" y="0" />
  <a:ext cx="5943600" cy="4457700" />
  </a:xfrm>
- <a:prstGeom prst="rect">

```

```

<a:avLst />
  </a:prstGeom>
  </pic:spPr>
  </pic:pic>
  </a:graphicData>
  </a:graphic>
  </wp:inline>
  </w:drawing>
  </w:r>
</w:p>
- <w:sectPr w:rsidR="00401AD8" w:rsidRPr="0091776E" w:rsidSect="00DC51FF">
  <w:pgSz w:w="12240" w:h="15840" />
  <w:pgMar w:top="1440" w:right="1440" w:bottom="1440" w:left="1440" w:header="720"
    w:footer="720" w:gutter="0" />
  <w:cols w:space="720" />
  <w:docGrid w:linePitch="360" />
  </w:sectPr>
- <a:IgnoreMe>
- <w:p w:rsidR="00401AD8" w:rsidRPr="0091776E" w:rsidRDefault="0091776E"
  w:rsidP="0091776E">
- <w:pPr>
- <w:tabs>
  <w:tab w:val="left" w:pos="1155" />
  </w:tabs>
  </w:pPr>
- <w:r>
- <w:rPr>
  <w:noProof />
  </w:rPr>
  <w:lastRenderedPageBreak />
- <w:drawing>
- <wp:inline distT="0" distB="0" distL="0" distR="0">
  <wp:extent cx="5943600" cy="4457700" />
  <wp:effectExtent l="19050" t="0" r="0" b="0" />
  <wp:docPr id="2" name="Picture 1" descr="Garden.jpg" />
- <wp:cNvGraphicFramePr>
  <a:graphicFrameLocks
    xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
    noChangeAspect="1" />
  </wp:cNvGraphicFramePr>
- <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
- <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
- <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
- <pic:nvPicPr>
  <pic:cNvPr id="0" name="Raffay.jpeg" />
  <pic:cNvPicPr />
  </pic:nvPicPr>
- <pic:blipFill>
  <a:blip r:embed="rId8" cstate="print" />
- <a:stretch>
  <a:fillRect />
  </a:stretch>
  </pic:blipFill>
- <pic:spPr>
- <a:xfrm>

```

```
<a:off x="0" y="0" />
<a:ext cx="5943600" cy="4457700" />
  </a:xfrm>
- <a:prstGeom prst="rect">
  <a:avLst />
    </a:prstGeom>
    </pic:spPr>
    </pic:pic>
    </a:graphicData>
    </a:graphic>
  </wp:inline>
</w:drawing>
</w:r>
</w:p>
</a:IgnoreMe>
</w:body>
</docxml>
</GlobalView>
```

Appendix D –Xquery Code of OOMXQA Algorithm

```
xquery version "1.0";
declare default element namespace
"http://schemas.openxmlformats.org/package/2006/relationships";
declare namespace ve="http://schemas.openxmlformats.org/markup-compatibility/2006";
declare namespace o="urn:schemas-microsoft-com:office:office";
declare namespace
r="http://schemas.openxmlformats.org/officeDocument/2006/relationships";
declare namespace m="http://schemas.openxmlformats.org/officeDocument/2006/math";
declare namespace v="urn:schemas-microsoft-com:vml";
declare namespace
wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing";
declare namespace w10="urn:schemas-microsoft-com:office:word";
declare namespace w="http://schemas.openxmlformats.org/wordprocessingml/2006/main";
declare namespace wne="http://schemas.microsoft.com/office/word/2006/wordml";
declare namespace a="http://schemas.openxmlformats.org/MyExtension/p1";

for $rels in doc("c:/Users/Raffay/Desktop/RnDxr.xml") //GlobalView/rels/Relationship
where not(ends-with($rels/@Type, "extended-properties") or ends-with($rels/@Type, "core-
properties") or ends-with($rels/@Type, "officeDocument") )
return <Pkg-Relationships> { $rels/@Id } { $rels/@Type } { $rels/@Target }
{ for $dx in doc("c:/Users/Raffay/Desktop/RnDxr.xml") //GlobalView/docxml/w:body
for $drels in doc("c:/Users/Raffay/Desktop/RnDxr.xml") //GlobalView/dxrels//Relationship
where
(ends-with($drels/@Type, "customXmlData") and matches($dx//body, string($drels/@Id) ))
or
not(ends-with($drels/@Type, "signature") or ends-with($drels/@Type, "origin") or ends-
with($drels/@Type, "customXml") or ends-with($drels/@Type, "customXmlProps") or ends-
with($drels/@Type, "thumbnail") or ends-with($drels/@Type, "aFChunk") or ends-
with($drels/@Type, "comments") or ends-with($drels/@Type, "settings") or ends-
with($drels/@Type, "endnotes") or ends-with($drels/@Type, "fontTable") or ends-
with($drels/@Type, "footer") or ends-with($drels/@Type, "footnotes") or ends-
with($drels/@Type, "glossaryDocument") or ends-with($drels/@Type, "header") or ends-
with($drels/@Type, "numbering") or ends-with($drels/@Type, "styles") or ends-
with($drels/@Type, "webSettings") or ends-with($drels/@Type, "attachedTemplate") or ends-
with($drels/@Type, "frame") or ends-with($drels/@Type, "subDocument") or ends-
with($drels/@Type, "mailMergeSource") or ends-with($drels/@Type,
"mailMergeHeaderSource") or ends-with($drels/@Type, "transform") or ends-
with($drels/@Type, "image") or ends-with($drels/@Type, "theme") or ends-
with($drels/@Type, "font") or ends-with($drels/@Type, "hyperlink") or ends-
with($drels/@Type, "vmlDrawing") or ends-with($drels/@Type, "printerSettings") or ends-
with($drels/@Type, "custom-properties")) or ($dx//a:IgnoreMe//@r:embed=$drels/@Id)
return <Doc-Relationship> { $drels/@Id } { $drels/@Type } { $drels/@Target } </Doc-
Relationship>}
</Pkg-Relationships>
```

Appendix E – XQuery Code of Conventional Algorithm

```
xquery version "1.0";
declare default element namespace "http://schemas.openxmlformats.org/package/2006/relationships";
declare namespace ve="http://schemas.openxmlformats.org/markup-compatibility/2006";
declare namespace o="urn:schemas-microsoft-com:office:office";
declare namespace r="http://schemas.openxmlformats.org/officeDocument/2006/relationships";
declare namespace m="http://schemas.openxmlformats.org/officeDocument/2006/math";
declare namespace v="urn:schemas-microsoft-com:vml";
declare namespace
wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing";
declare namespace w10="urn:schemas-microsoft-com:office:word";
declare namespace w="http://schemas.openxmlformats.org/wordprocessingml/2006/main";
declare namespace wne="http://schemas.microsoft.com/office/word/2006/wordml";
declare namespace a="http://schemas.openxmlformats.org/MyExtension/p1";

for $rels in doc('jar:file:///C:/Users/Raffay/Desktop/All_Tech.zip!/_rels/.rels')
//Relationships/Relationship
where not(ends-with($rels/@Type, "extended-properties") or ends-with($rels/@Type, "core-
properties") or ends-with($rels/@Type, "officeDocument") )
return <Pkg-Relationships> { $rels/@Id } { $rels/@Type } { $rels/@Target }
{ for $dx in doc('jar:file:///C:/Users/Raffay/Desktop/All_Tech.zip!/word/document.xml')
//w:document/w:body
for $drels in doc('jar:file:///c:/Users/Raffay/Desktop/All_Tech.zip!/word/_rels/document.xml.rels')
//Relationships/Relationship
where
(ends-with($drels/@Type, "customXmlData") and matches($dx/body, string($drels/@Id) )) or
not(ends-with($drels/@Type, "signature") or ends-with($drels/@Type, "origin") or ends-
with($drels/@Type, "customXml") or ends-with($drels/@Type, "customXmlProps") or ends-
with($drels/@Type, "thumbnail") or ends-with($drels/@Type, "aFChunk") or ends-
with($drels/@Type, "comments") or ends-with($drels/@Type, "settings") or ends-with($drels/@Type,
"endnotes") or ends-with($drels/@Type, "fontTable") or ends-with($drels/@Type, "footer") or ends-
with($drels/@Type, "footnotes") or ends-with($drels/@Type, "glossaryDocument") or ends-
with($drels/@Type, "header") or ends-with($drels/@Type, "numbering") or ends-with($drels/@Type,
"styles") or ends-with($drels/@Type, "webSettings") or ends-with($drels/@Type, "attachedTemplate")
or ends-with($drels/@Type, "frame") or ends-with($drels/@Type, "subDocument") or ends-
with($drels/@Type, "mailMergeSource") or ends-with($drels/@Type, "mailMergeHeaderSource") or
ends-with($drels/@Type, "transform") or ends-with($drels/@Type, "image") or ends-
with($drels/@Type, "theme") or ends-with($drels/@Type, "font") or ends-with($drels/@Type,
"hyperlink") or ends-with($drels/@Type, "vmlDrawing") or ends-with($drels/@Type,
"printerSettings") or ends-with($drels/@Type, "custom-properties")) or
($dx//a:IgnoreMe//@r:embed=$drels/@Id)
return <SubPkg-Relationship> { $drels/@Id } { $drels/@Type } { $drels/@Target } </SubPkg-
Relationship> }
</Pkg-Relationships>
```

NOTE: For Altova XML Spy 2010 to read from the zip archive, given code has to be amended as follows: **doc("C:/All_Tech.zip|zip/_rels/.rels")**

Appendix F – XQuery code for creating GMV

```
xquery version "1.0";
declare default element namespace
"http://schemas.openxmlformats.org/package/2006/relationships";
declare namespace ve="http://schemas.openxmlformats.org/markup-compatibility/2006";
declare namespace o="urn:schemas-microsoft-com:office:office";
declare namespace
r="http://schemas.openxmlformats.org/officeDocument/2006/relationships";
declare namespace m="http://schemas.openxmlformats.org/officeDocument/2006/math";
declare namespace v="urn:schemas-microsoft-com:vml";
declare namespace
wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing";
declare namespace w10="urn:schemas-microsoft-com:office:word";
declare namespace w="http://schemas.openxmlformats.org/wordprocessingml/2006/main";
declare namespace wne="http://schemas.microsoft.com/office/word/2006/wordml";
declare namespace a="http://schemas.openxmlformats.org/MyExtension/p1";
let $rels := doc('jar:file:///C:/Users/Raffay/Desktop/All_Tech.zip!/_rels/.rels')
//Relationships/Relationship
return <GlobalView><rels> {$rels}</rels>
{let $rels1 :=
doc('jar:file:///C:/Users/Raffay/Desktop/All_Tech.zip!/word/_rels/document.xml.rels')
//Relationships/Relationship
return <dxrels>{$rels1}</dxrels>}
{let $doc := doc('jar:file:///C:/Users/Raffay/Desktop/All_Tech.zip!/word/document.xml')
//w:document/w:body
return <docxml>{$doc}
</docxml>}
</GlobalView>
```