Every Signature is Broken: On the Insecurity of Microsoft Office's OOXML Signatures

Simon Rohlmann, Vladislav Mladenov, Christian Mainka,
Daniel Hirschberger, and Jörg Schwenk, *Ruhr University Bochum*

https://www.usenix.org/conference/usenixsecurity23/presentation/rohlmann

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the
32nd USENIX Security Symposium
is sponsored by USENIX.

Every Signature is Broken: On the Insecurity of Microsoft Office's OOXML Signatures

Simon Rohlmann
Ruhr University Bochum

Vladislav Mladenov
Ruhr University Bochum

Christian Mainka
Ruhr University Bochum

Daniel Hirschberger
Ruhr University Bochum

Jörg Schwenk
Ruhr University Bochum

Abstract

Microsoft Office is one of the most widely used applications for office documents. For documents of prime importance, such as contracts and invoices, the content can be signed to guarantee authenticity and integrity. Since 2019, security researchers have uncovered attacks against the integrity protection in other office standards like PDF and ODF. Since Microsoft Office documents rely on different specifications and processing rules, the existing attacks are not applicable.

We are the first to provide an in-depth analysis of Office Open XML (OOXML) Signatures, the Ecma/ISO standard that all Microsoft Office applications use. Our analysis reveals major discrepancies between the structure of office documents and the way digital signatures are verified. These discrepancies lead to serious security flaws in the specification and in the implementation. As a result, we discovered five new attack classes. Each attack allows attackers to modify the content in signed documents, while the signatures are still displayed as valid.

We tested the attacks against different Microsoft Office versions on Windows and macOS, as well as against OnlyOffice Desktop on Windows, macOS and Linux. All tested Office versions are vulnerable. On macOS, we could reveal a surprising result: although Microsoft Office indicates that the document is protected by a signature, the signature is not validated. The attacks' impact is alarming: attackers can arbitrarily manipulate the displayed content of a signed document, and victims are unable to detect the tampering. Even worse, we present a universal signature forgery attack that allows the attacker to create an arbitrary document and apply a signature extracted from a different source, such as an ODF document or a SAML token. For the victim, the document is displayed as validly signed by a trusted entity.

We propose countermeasures to prevent such issues in the future. During a coordinated disclosure, Microsoft acknowledged and awarded our research with a bug bounty.

1 Introduction

Microsoft Office is one of the most important tools to manage word documents, presentations, and spreadsheets. For Office 365 alone, there were nearly 300 million paying users worldwide in 2021 [1]. Starting with Office 2007, all documents by default are stored as *Office Open XML* documents (OOXML [2]).

OOXML Document Signatures. Similar to competing office formats like PDF and ODF, Microsoft offers digital signatures to protect their electronic documents, for instance, Word, Excel, and Powerpoint.

> A digital signature is an electronic, encrypted, stamp of authentication on digital information such as email messages, macros, or electronic documents. A signature confirms that the information originated from the signer and has not been altered [3].

These strong security guarantees can be used to protect critical office documents, such as contracts and invoices, against tampering. The Office Open XML (OOXML) standard defines how office documents are digitally signed, using a specific variant of XML signatures [4]. OOXML digital signatures on office documents are used, among others, by governmental agencies like the Defense Counterintelligence and Security Agency (DCSA) [5], the government of Canada [6], the Federal register [7], and by the Federal Identity, Credential, and Access Management (FICAM) program [8].

Prior Work. In recent years, several academic publications [9–12] addressed the security of digital signatures on office document formats like PDF and ODF, which are different from OOXML. These works identified several attacks where the contents of the documents could be altered, but the signature still was displayed as valid. However, these attacks are not applicable to OOXML documents, since they rely on the specific structure of the different data formats. Similarly, Microsoft macro code can be digitally signed, but a different data structure is used for this purpose. So recent

attacks on macro signatures, such as CVE-2020-0991 and CVE-2020-0760, are not related to our work.

A Complex Container of Linked XML Structures. Both the rendering flow and the signature verification flow of OOXML documents are complex. Various files of the OOXML package and cross-references in multiple files are involved in the rendering process. Besides the main `document.xml` there is a relationship file specifying which other files should be included during rendering. The names of these other files suggest a specific use: For example, the file `people.xml` is intended to store only the names of people who made comments in a document. We discovered that such files could also store any renderable content shown after opening the document.

The signature verification flow uses the complex XML signature standard with different types of references to signed parts (URIs, IDs), and increases this complexity by adding another layer of references to the hash tree of XML signatures.

Specification Flaws. We show that the complexity of the signature verification and on the rendering side, lead to vulnerabilities on the standards level which undermine the security goals stated by Microsoft [3]. More precisely, we show that the following three attack classes require fixes in the current specifications.

Content Injection Attack (CIA) abuses the standard's discrepancy to place rendered content in files that should be used for different purposes, for example, holding meta-information.

Content Masking Attack (CMA) manipulates styling or font information after a document is signed in such a way, that different content is displayed.

Legacy Wrapping Attack (LWA) embeds a signature of a legacy document (e.g., `*.doc`) into an OOXML document.

All attacks are compliant with the standard while exploiting edge-cases: some well-specified parts are not protected by the signature since they are expected to be harmless. Our attacks prove this assumption wrong.

Implementation Flaws. In addition, to the specification flaws, we discovered two implementation flaws in *all* Microsoft Office applications:

Universal Signature Forgery (USF) exploits serious flaw in the verification logic allowing attacker to manipulate *any* content of the document. From a single valid XML signature token, taken from any other source (e.g., ODF, SAML), arbitrary OOXML files can be constructed that will be displayed as validly signed.

Malicious Repair Attack (MRA) abuses the repair functionality in Microsoft Office to hide harmless content and present the malicious one. This is the only attack requiring user interaction – the victim clicks on the repair prompt.

Research Methodology. The security of office document signatures depends on the interplay between the rendering flow, and the signature validation flow: Only validly signed parts of the document should be rendered. We studied the OOXML standard (6730 pages) and systematized both the document rendering flow and the signature validation flow. It turned out that both flows are very complex. For instance, the signature validation flow increases the complexity of the signature verification by adding an additional level of references to the hash tree used by XML signatures. The rendering flow, on the other side, sometimes recognizes partially signed documents and sometimes it does not. We systematize the core insights of the standard and describe them in Figure 1.

We carefully analyzed both flows for vulnerabilities. In the rendering flow, we looked for content that can be added after signing. We placed this content at different locations in the OOXML package ensemble and tested if this content was made visible by the rendering flow. For the signature validation flow, we carefully analyzed the different reference types. We identified structures within the OOXML package ensemble which were only partially signed, and which could be manipulated. These partly signed structures are fixed by the OOXML standard, so we classified any attacks which could be traced back to this root cause as standard-level attacks. We also discovered implementation errors in Microsoft Office and OnlyOffice. For macOS versions of Microsoft Office, we were able to show that no signature validation is performed. By combining these two analysis approaches, we were able to discover the above-mentioned vulnerabilities.

Contributions. We make the following key contributions:

- We are the first to provide a systematic analysis of OOXML signatures (Section 4). We extract and analyze systematically both the rendering and the signature validation.
- We identify three major issues (Section 5): (1) OOXML uses *partial signatures*. (2) The rendering flow does not differ between signed and unsigned content. (3) The cryptographic verification of the digital signatures is complex and requires the correct validation of references to multiple elements and files, transformations of the signed content, hash computations, and public-key operations.
- We built every issue into an attack class that manipulates the displayed content of a signed OOXML document in such a way, that a victim opening the document is unable to detect the manipulation. In summary, we created more than 470 attack vectors.
- We evaluate Microsoft Office on Windows and macOS in different versions, as well as OnlyOffice Desktop on Windows, macOS and Linux (Section 7). While all Windows versions of Microsoft Office are entirely vulnerable to each attack, the result for macOS is even worse. We

discovered that on macOS, it is sufficient to include a `sig1.xml` without any content to force the application to show a security banner stating that the document is protected by a signature.

- We propose countermeasures (Section 8). We describe one particular mitigation for each attack class that addresses the attack's root cause.

We have created proof of concept (PoC) files for all attacks described in this paper and made them available at the following URL: https://github.com/RUB-NDS/OOXML_Signature_Security

Coordinated Vulnerability Disclosure. We have reported all vulnerabilities found during our investigations to Microsoft, OnlyOffice, as well as to the responsible standardization committee ISO/IEC JTC 1/SC 34. The vulnerabilities have been acknowledged by Microsoft. However, Microsoft has decided that the vulnerabilities do not require immediate attention. According to Microsoft, a potential fix in the future is not excluded. We have not received any feedback from OnlyOffice as of October 4th, 2022.

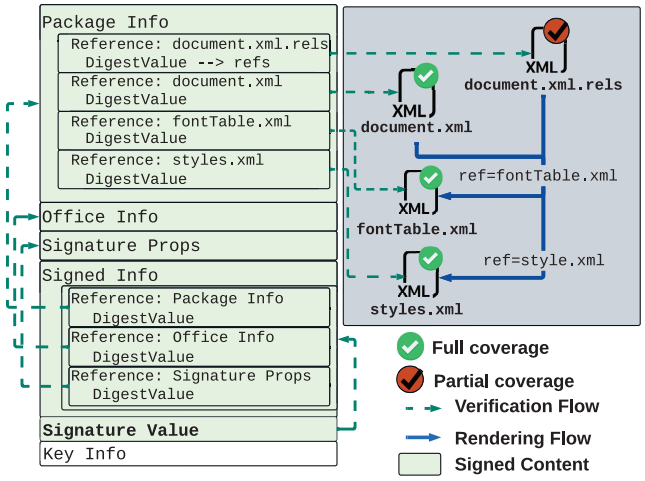

Figure 1: The digital signature in OOXML is stored in an Extensible Markup Language (XML) file. The starting point of the verification flow is the *Signature Value*. All files are implicitly protected by storing and verifying their digest values. The rendering flow processes first `document.xml` and `document.xml.rels` to load all needed files.

2 The OOXML Document Format

The OOXML standard is divided into four parts, published in the 5th edition in 2015-2021 by Ecma International [2], and as ISO/IEC Standard 29500 in the 4th edition in 2015-2021 [13]. It relies on the Open Packaging Conventions (OPC), for instance, multiple files are zipped in a container. In this section, we explain the meaning of these files and describe how digital signatures are applied.

Document Structure. Each OOXML document contains multiple files in a zipped package. The `[Content_Types].xml` holds for the contained files of the OOXML the corresponding content type and their relative path. The relationship file contained in `_rels/.rels` contains the references to the main document file `word/documents.xml`, and to the properties files `docProps/app.xml` and `docProps/core.xml`. The properties files contain, among other things, information about the author, creation time, or the Office version. The document's content, which is presented once the document is opened, is mainly stored in `word/document.xml`. The `word/_rels/document.xml.rels` relationship file defines a catalog with all further files. Thus, the application knows which files are included and where to find them (see Figure 1). The files `word/styles.xml` and `word/fontTable.xml` specify different preferences, such as character spacing and fonts.

Rendering Process. The `word/document.xml` file contains the displayed content of the document in a XML *body* element and serves as the basis for the rendering process. During the rendering process, other files of the OOXML package can be included. For this, the `word/document.xml` contains ID based references which must correspond to the entries within the relationship file at `word/_rels/document.xml.rels`. The files included in this way, for example `word/styles.xml` or `word/fontTables.xml`, are thus integrated into the rendering process in order to adjust the appearance of texts accordingly. The files referenced like that, e.g., `word/fontTables.xml`, can also include their own relationship files under `word/_rels/`, which reference embedded fonts, for example.

Digital Signature. Figure 1 depicts the structure of the digital signature of an OOXML document. Each signature is stored in an XML file which contains six main areas: *Package Info*, *Office Info*, *Signature Properties*, *Signed Info*, *Signature Value*, and *Key Info*.

Package Info lists all protected files by referencing their name and path. The signature always protects the `document.xml` file. In addition, for each file referenced in `document.xml.rels`, the hash value of the entire content is computed and stored. An important exception is `document.xml.rels` itself since it is partially signed. *Office Info* stores the application preferences, such as the used Office and Windows versions. Also, resolution preferences are stored. *Signature Properties* defines metadata regarding the signature generation like timestamps and used certificates. *Signed Info* references the *Package Info*, *Office Info*, and *Signature Properties*. For each area, a digest value is computed. Finally, all references are digitally signed. *Signature Value* stores the digital signature computed over *Signed Info*. *Key Info* refers to the keys used to sign the document. This area is not signed so that attackers can manipulate it.

Verification Flow. The verification process of a signed document begins with the *Signature Value* – the application cryp-

tographically verifies the correctness of the value, using the public key extracted from *Key Info*. This step ensures that the references and the corresponding digest values of *Package Info*, *Office Info*, and *Signature Properties* have not been modified. Second, the application computes the digest values of each reference and compares them with the stored digest values. A successful verification means that all three areas have not been modified. Then, the hash values of all referenced files in the *Package Info* are computed and compared. As a result, the application verifies that the files have not been modified. Finally, the certificate's trustworthiness that the *Key Info* holds is validated. For instance, Microsoft Office relies on the Windows Certificate Store for the PKI-based trust validation.

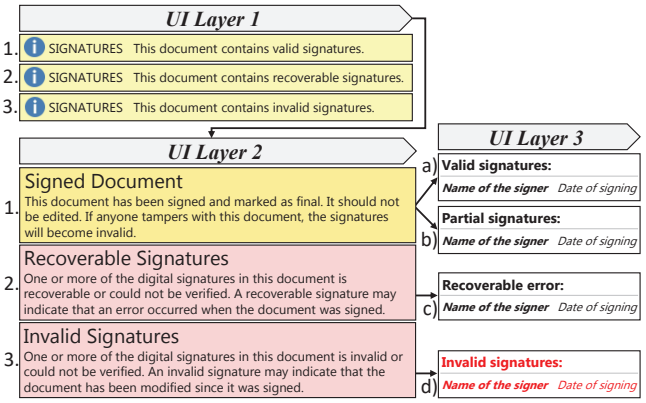

Figure 2: Representation of the possible signature states of a signed OOXML document, within the User Interface (UI) layers 1, 2, and 3 of Microsoft Office: (1) The signature is valid and trusted, (2) the signature is valid but not trusted, and (3) the signature is invalid.

UI Layer. To reflect the signature status of a document, Microsoft Office uses different UI layers (see Figure 2). The first UI layer is shown as a banner directly when opening a signed document. Here the signature status is indicated as 1. valid (signature is valid and the certificate is trusted), 2. recoverable (signature is valid, but the certificate is not trusted) or 3. invalid (signature is invalid). UI layer 2 reflects the signature status of UI layer 1 and contains detailed information about the signature status. UI layer 3 contains information about the signer and distinguishes between a) full and b) partial signature for a valid signed document. When Microsoft Office signs a document, the signature status is displayed in UI layer combination (1. a) despite the partial signature.

3 Attacker Model

In this section, we define a set of capabilities and rules that attackers and victims follow during the attack execution. The attacker creates a document using the given *capabilities*. Then, the victim receives that document and opens it. Depending on the attacker's *goal*, the attack can succeed or fail.

Attacker Capabilities. In this section, we define two requirements. The attacker fulfills one of them to carry out the attacks described in this paper.

Trustfully Signed Document () The attacker has access to a trustfully signed document. Such a document may be publicly available on the Internet, for example, a signed law document.

Signing Oracle () In 2021, a new attacker model on signed documents was introduced [10]. In this case, the attacker has access to a signing oracle, similar to a chosen-message attack. Thus, the attacker can choose – wholly or partly – the document's content to be signed. The victim trusts that document's signature. We assume that the malicious payload must be invisible/not executed by the signing oracle. Thus, the document appears to be legit. After signing, the attacker reveals the malicious payload in a subsequent document manipulation.

Attacker Goal: Data Manipulation (). The attacker aims to manipulate the content that the application renders. Since a signature protects the document's integrity, a desirable attack goal is to change the content of the document without invalidating the signature status. An attack is considered successful if the signature displays as valid and trustworthy while malicious content is rendered and shown to the victim.

Victim's Behaviour. We assume that the victim expects to open a document that is signed by a trusted authority. The victim does not trust the document when warnings are thrown regarding an invalid or untrusted signature. We exclude documents signed with untrusted keys since such documents throw a warning after opening them.

4 Systematic Analysis

We divided our analysis in five phases. First, we analyzed the OOXML standard in the 5th edition [2] and its implementation in different versions of Microsoft Office. Consequentially, we studied the processing of documents in Microsoft Office and documented edge-cases with potential security impact. Based on our observations, we developed the attacks described in this paper and evaluated them.

Phase 1: Security Issues in the OOXML Standard. In the first phase, we analyzed the OOXML standard regarding the implementation of signatures. We identified a severe issue – the OOXML standard uses partial signatures that do not protect the entire OOXML package [2, Part 2, p. 45] (see Appendix A.1). More concrete, the relationship files, which are responsible for referencing the files, contain unprotected parts. The signature only protects individual strings, indexed by unique IDs, but the relationship file as a whole remains unsigned [2, Part 2, p. 50] (see Appendix A.1). This feature allows attackers to include references to subsequently added files after signing.

For individual files within the OOXML package, such as `styles.xml`, the standard defines the root element of the respective XML file. However, it is not forbidden to use elements defined for other files, e.g., `documents.xml`. Thus, the `body` element responsible for the rendered content can be defined in other files besides `documents.xml`, such as `styles.xml`. A distinction whether the inserted content is signed or unsigned does not take place and is not distinguishable in any way.

Result: The creation and verification of digital signatures in OOXML contradicts at least the security goal of full integrity protection.

Phase 2: OOXML Signatures in Microsoft Office. In Phase 2, we analyzed how Microsoft Office builds and signs OOXML packages in practice. We could verify that Microsoft Office supports the concept for partial signatures. For example, the digital signature is computed over all files within the `word/` subdirectory as well as their reference in the relationship file. However, there are also files that are not part of the signature. We identified that the file describing the content types `[Content_Types].xml` of the package is unprotected. There are additional unprotected properties files, such as `docProps/app.xml` and `docProps/core.xml` containing, among other things, the creator of the document.

Result: Through the analysis in Phase 1 and 2, three possible points of attack could be identified: (1) manipulation of existing but unsigned files, (2) manipulation of the signature file, and (3) manipulation by subsequently added files or references.

Phase 3: Manipulation of Unsigned Files. We analyzed how Microsoft Office reacts to changes within the `[Content_Types].xml`, `docProps/app.xml`, `docProps/core.xml` and the corresponding relationship files. For this purpose, we changed the content type and reference in the relationship file. We gradually added new content to the property files that should either change the style element of the OOXML or add new content to be rendered in the document. As soon as the XML root element within a properties file deviated from the content type or defined reference, Microsoft Office considers the OOXML document corrupt. Style elements or content inserted within the property files were not actually displayed as document content with any of the attack vectors. Only the meta information of the document, such as creator, last modifier, and the associated timestamps could be modified without invalidating the signature. However, since these files are not included in the signature calculation, this result was expected.

Result: In this analysis phase, 24 OOXML with different attack vectors were created. We confirmed known threats on the current version of the OOXML standard [14], but no new attacks have been found.

Phase 4: Manipulation of the Signature File. A characteristic for the signature file in OOXML packages is that the files-to-be-signed and their reference strings are not directly placed in *Signed Info*.

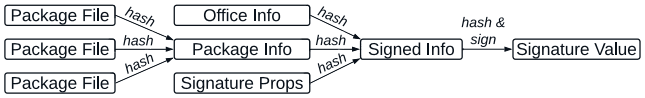

Figure 3: The signature flow generates multiple relations between the protected content. If any of the relations is not properly verified, attacks are possible.

Within *Signed Info*, a reference is given to *Package Info*. In *Package Info*, the digest values of multiple files are stored and compared during the signature verification. If the reference to *Package Info* is missing, the rendered content of the OOXML package is not protected by the signature. In our analysis, we evaluated how Microsoft Office reacts if the referenced element is missing. For this purpose, we took an XML signature that has not been created with Microsoft Office. Thus, the signature is valid, but it does not contain the reference. The result is a novel attack variant on OOXML documents which we introduce in Section 6.1.

In addition, we analyzed how Microsoft Office handles signatures created with untrusted keys (Key Confusion [12, 15, 16]). We also evaluated the behavior in case of multiple elements in *Key Info*. The attack's idea is that one key could be used to validate the signature (key/certificate controlled by the attacker) and a second key to validate the signer's trustworthiness. For this attack, we considered different key formats allowed in XML and different positions within *Key Info*. We did not find any vulnerabilities for this attack class for Microsoft Office.

XML Signature Wrapping (XSW) is another attack class on XML signatures [12, 17, 18]. We used XSW and evaluated the manipulation of objects referenced within the signature file. We could not identify any vulnerabilities with this approach.

We checked signed OOXML documents also against further known attacks on XML signatures: By simply removing relevant parts of the signature (Signature Exclusion [18]), we tested whether the processing of an XML signature is disrupted by the verification logic and falsely displays a valid signature.

We also analyzed attacks like Certificate Faking [12, 15] which rely on an incorrect verification of the certificate trustworthiness. Thus, documents created by the attacker by using untrusted keys are still displayed as trusted and valid to the victim.

Another known attack on XML signatures is the Node Splitting attack [19, 20] which confuses the XML parser. The idea is to insert a comment or XML entity into a value so that an implementation can split it into two parts. The rendering logic could display different values than the content processed by the verification logic.

We also examined possible attacks using HMAC trunca-

tion [21]. However, HMAC is not supported for OOXML signatures.

For none of the attacks Signature Exclusion, Certificate Faking, Node Splitting, HMAC Truncation we could find vulnerabilities in signed OOXML documents.

Result: By failing to verify that the files in the OOXML package are actually referenced in *Signed Info*, it is possible to create universally usable valid signatures that do not provide any integrity protection over the files contained in the package. A total of 211 OOXML documents with different test vectors were generated in this phase. Known attacks on XML signatures were adapted and tested on OOXML documents.

Phase 5: Manipulation Based on Partial Signatures. In the fifth analysis phase, we investigated the support of partial signatures in OOXML. First, we tested which manipulations are possible based on an already signed OOXML document. Since both the `[Content_Types].xml` file is not signed at all and the relationship files are only partially signed, there is always a possibility of adding more files to a signed OOXML package. The main challenge is to force the application to process and render the injected files. For instance, the individual files in the package must also be used within the `document.xml` to be presented after opening the document. As a result, we addressed two main questions:

(1) Do files exist that are rendered by opening the document, but not referenced in `document.xml`?
(2) Can we exclude files referenced by `document.xml` from the signature computation?

We found solutions to both questions resulting in two different attacks classes which are described in Section 5.1 and Section 5.2.

We have also identified a problematic feature related to signed OOXML documents. In OOXML documents, graphics can be used as external resources. Here, only the URL of the graphic is stored in the document and automatically reloaded when opened. The automatic reloading of the graphic can also be observed in signed documents and can be used to change the graphic within a signed document by replacing the image stored under the URL.

In the last analysis step of the fifth phase, we looked at the properties of Microsoft Office's automatic repair feature. This feature is used, for example, when a OOXML package contains an unexpected file combination. This led to the attacks described in Section 6.2. However, the limitation of this attack class is that once repaired, documents cannot be saved without changing the file structure and thus invalidating the signature. Thus, they are only suitable for a one-time attack.

Result: A total of 252 attack vectors emerged in this analysis phase, resulting in six novel attacks on OOXML signatures. Most of the attacks exploit the OOXML's support for partial signatures and show that partial signing does not preserve the integrity of a document.

Attack Vector Generation. A simple Word document signed with Microsoft Office has at least two relationship files and one file to describe the content type of the different files in the OOXML package. This simple Word document contains a total of 9 references with different types and IDs. In addition, the `[Content_Types].xml` contains 3 default type definitions and also a content type for each reference. There are different possible combinations of the entries within these three files which need to be tested. If the document is extended with comments, footers, headers and footnotes, the entries in the relationship files and `[Content_Types].xml` will be extended accordingly. If graphics or fonts are embedded, even new relationship files with corresponding references to the document are added. This resulted in the described multitude of test vectors in the different analysis phases, which were created by various permutations based on the following initial questions: What happens if we ...

- exchange the entries between the relationship files?
- refer to files that do not exist?
- declare XML elements that do not match the XML schema?
- define multiple entries with the same or different IDs?
- reuse XML elements?
- have different content types?
- combine any of the manipulation techniques?

The creation of the test vectors for the signature file attacks also requires a large number of permutations to check the vulnerability of the applications. For example, to carry out XSW the variety arrangement of XML elements using the same or different ID should be considered. With respect to Key Confusion, the use of different X509 Data elements in different order leads to the creation of multiple documents. While most of the attack vectors were created manually, the correct signing with an RSA or DSA key required the programming of a tool written in C# by using the functions of the `System.Security.Cryptography` namespace. In total, the analysis phase produced over 430 attack vectors, which resulted in 7 successful PoCs, which are described in detail in Section 5 and Section 6.

5 Specification Attacks on OOXML

This section describes attacks that create documents which are compliant to the OOXML specification. Thus, implementations that strictly implement the OOXML specification are vulnerable. We use Word documents as the basis for the attacks. Similar to Word, signatures are also applied to Excel and PowerPoint documents. However, we stress that the files within Excel and PowerPoint OOXML packages are different from Word documents.

5.1 Content Injection Attack (CIA)

The goal of this attack is to manipulate a trustfully signed OOXML document by adding arbitrary content to the doc-

ument while hiding the original content. It abuses a lack in the OOXML specification allowing to add unsigned files and reference these.

Attack Requirements. The attacker requires an OOXML document signed by a trusted entity.

Manipulation Technique. The attack exploits the partial signature coverage of the relationship file `document.xml.rels`. Thus, it is possible to subsequently add XML files to a signed OOXML package and reference them correctly. Usually, these added files do not influence the content of the document because they are not referenced in the `document.xml` file. However, this does not apply to `people.xml` since this file is processed without prior referencing. Thus, arbitrary content can be added and presented by opening the signed document. This content includes, for example, text, text boxes, or graphics. The attacker can use text boxes or graphics to cover the original text and add arbitrary new content above it (see Figure 4).

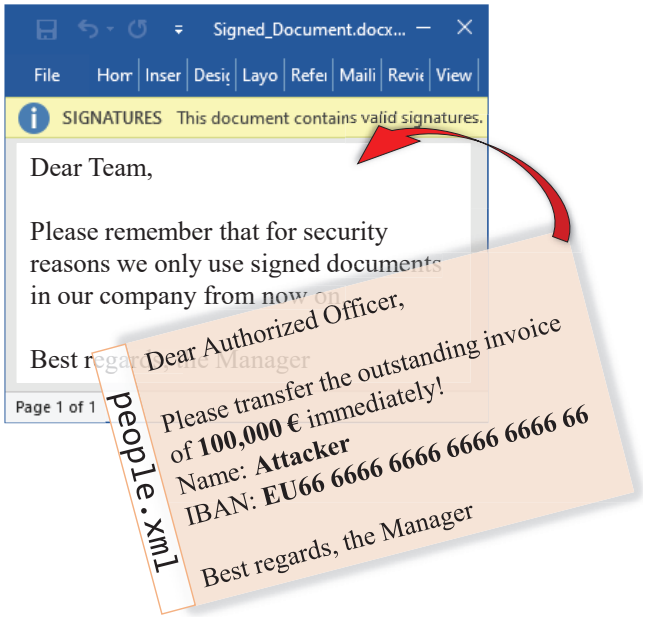

Figure 4: Content Injection Attack (CIA) attack on a signed document of a trusted entity. The attacker uses the `people.xml` file to subsequently modify the signed document and place own content over the original content.

Manipulation Steps.

(1) The attacker requires an OOXML document validly signed by a trusted entity.
(2) The attacker creates a `people.xml` file.
(3) In this file, the attacker places multiple text boxes with a white background overlaying the original text and sets up their positions in the document.
(4) The attacker adds the `people.xml` file into the `word` subfolder of the OOXML package.
(5) The attacker manipulates `[Content_Types].xml` and adds a reference to the file `people.xml`. This manipulation is permitted because `[Content_Types].xml` is not signed.
(6) The attacker manipulates `document.xml.rels` by referencing the malicious file `people.xml`. This manipulation is also permitted because the relationship file is only partially signed. With the support of partial signatures in the specification, already signed elements cannot be tampered, but new files can be added without invalidating the signature.
(7) (Optional) Graphics referenced by `people.xml` should be declared in `people.xml.rels` and placed in the `word/media` folder.

Impact. After opening the document, the content injected by the attacker is displayed. The applied signature of the trusted entity is still successfully verified. The attacker, however, can hide original content and overlay it with arbitrary text. It is also possible for the attacker to increase the original page count by injecting new content. Only the reduction of the original number of pages is not possible.

5.2 Content Masking Attack (CMA)

The idea of content masking is similar to CIA: the attacker includes unsigned malicious files after the document is signed. In comparison to CIA, the requirements for content masking attacks are different.

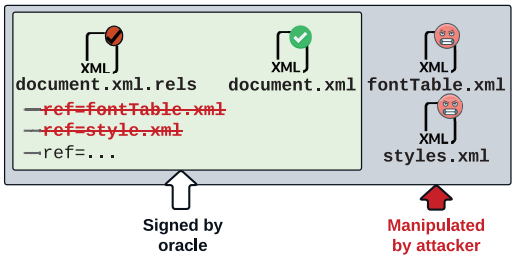

Figure 5: The attacker sends a harmless-looking document to the signing oracle which does not contain references to contained fonts and styles. As a result, the fonts and styles will be not signed. After signing the attacker can arbitrarily manipulate these files and change the text appearance of the signed document.

Attack Requirements. The attack requires a signing oracle. The attacker creates a harmless-looking document that a trusted entity signs. After signing, the attacker manipulates the signed document and distributes it to the victim.

Manipulation Technique. First, the attacker creates a malicious document that they want to sign. Then, the attacker removes the references to `fontTable.xml` or `style.xml` in the `document.xml.rels`. Since only referenced files in `document.xml.rels` are protected by the signature, the attacker can manipulate later all unsigned files.

In comparison to CIA, where the injection of the `people.xml` is always possible, the attacker needs to prepare the malicious document before it is signed. For example, text areas in `document.xml` should refer to fonts or styles defined in `fontTable.xml` or `style.xml`. Otherwise, the manipulations will not affect the appearance of the corresponding text areas.

Font Injection Attack (FIA). For OOXML documents it is possible to embed fonts. If fonts have been used that are neither embedded nor present on the viewer's system, an operating system font that is as similar as possible can be selected automatically [2, Part 1, pp. 669-670]. An attacker can use these features to mask content or display it differently. The attacker creates custom fonts, for example, with the tool FontForge[1]. The attacker must choose a different name from the fonts available on the system (e.g., `Arial1` instead of `Arial`). Within the custom fonts, the attacker can remove the graphic representation of individual letters to hide specific text passages. Within additional custom fonts, the attacker exchanges letters with each other to change specific text passages. By using multiple custom fonts for different text passages, the attacker can create a document that displays the same content entirely differently depending on whether or not custom fonts are embedded and referenced in the document. After the document is created, the attacker removes the reference to the embedded fonts and passes it to the signing oracle. The attacker's harmless content is rendered since the embedded fonts are not referenced and the signing oracle does not have the custom fonts installed. After signing, the attacker inserts the reference to the custom fonts back into the document. This insertion masks the content or renders it differently based on the custom fonts.

Style Injection Attack (SIA). Similar to FIA, the attacker uses a signing oracle to get a self-created Word document signed without the reference to the `styles.xml` file. The attacker can add the reference to the `styles.xml` back to the relationship file without breaking the signature. By manipulating `styles.xml`, the appearance of individual elements of the displayed content can vary. In this way, the attacker can, for example, selectively hide or overlay individual text passages.

Impact. By adding content via the *body* element, new content can also be injected via the *styles.xml*. In comparison to CIA, the impact is lower since the attacker needs a signing oracle for every manipulated document.

5.3 Legacy Wrapping Attack (LWA)

The goal of Legacy Wrapping Attack (LWA) is to use the signature of a Word document in the old binary file format (e.g., `.doc`) to display a valid signature to the victim over content controlled by the attacker. The victim sees a valid signature of a trusted entity while opening the signed document (`.docx`).

[1] https://fontforge.org

Attack Requirements. The attack requires a signing oracle, which means that the attacker creates a malicious but harmless-looking document that a trusted entity signs.

Manipulation Technique. This attack combines legacy document formats with the modern XML signature. Legacy documents use the Compound File Binary format as a container format [22]. When a Microsoft Office version that supports XML-based signatures signs a legacy document, it creates a new folder `_xmlsignatures` in the legacy document. This folder contains a single file with a random number as the file name. This signature file is identical to an `sig1.xml` file that the modern OOXML format uses. The signature of a `.doc` legacy document includes `1Table`, `[0x01]CompObj`, `Data`, and `WordDocument`. The byte before `CompObj` is a byte with the value `1`. Word requires that all files in OOXML documents have a content type assigned in `[Content_Types].xml` Otherwise, it considers the document corrupted and prompts for repairing the document. Most non-printable ASCII characters, including 0x01, cannot be used directly. The attacker can customize the legacy document with a hex editor to work around this limitation. By manipulating the `0x01` byte into a printable character, for example, into `A`, the attacker keeps the structure of the document intact.

Manipulation Steps.

(1) The attacker creates a legacy document (`.doc`) with content that the signing oracle will agree to.
(2) The attacker transforms the byte with value `0x01` before `CompObj` into a printable byte (in this case `A`).
(3) The signing oracle signs the document.
(4) The attacker creates a new `.docx` document and chooses the content arbitrarily.
(5) The attacker injects `1Table`, `ACompObj`, `Data`, and `WordDocument` into the root directory of the `.docx` package.
(6) The attacker sets the content types to `application/octet-stream` in `[Content_Types].xml`.
(7) The attacker copies the `_xmlsignatures` from the signed `.doc` document to the `.docx` package.
(8) The attacker renames the signature file in `_xmlsignatures` (which has a file name consisting of a random number) to `sig1.xml`.
(9) The attacker creates the `_xmlsignatures/origin.sigs` and `_xmlsignatures/_rels` file for `sig1.xml`.
(10) The attacker adds the content types for `_xmlsignatures/origin.sigs` and `_xmlsignatures/sig1.xml` in `[Content_Types].xml`.
(11) The attacker adds a digital signature relationship to `_rels/.rels`.

Impact. This attack takes advantage of legacy documents (`.doc`) that are differently structured in comparison to OOXML documents. When a legacy document is signed with a newer version of Microsoft Office, an XML signature is created that is compatible with the XML signatures of OOXML

documents. By obtaining an OOXML signature over a legacy document, it is possible to insert the legacy document and signature into any OOXML document to obtain a universally valid signature. This document displays a valid signature of a trusted entity to the victim when opening the manipulated document while allowing the attacker to choose the content arbitrarily.

6 Implementation Attacks on OOXML Office Applications

This section describes attacks whose success depends on underlying peculiarities of individual implementations, for instance, Microsoft Office. These quirks can result from cryptographic implementation bugs (Section 6.1) or implementation features, such as document repair (Section 6.2).

6.1 Universal Signature Forgery (USF)

The attacker's goal with Universal Signature Forgery (USF) is creating a signed OOXML document with arbitrary content. Once the victim opens the document, the application displays a valid signature belonging to a trusted entity.

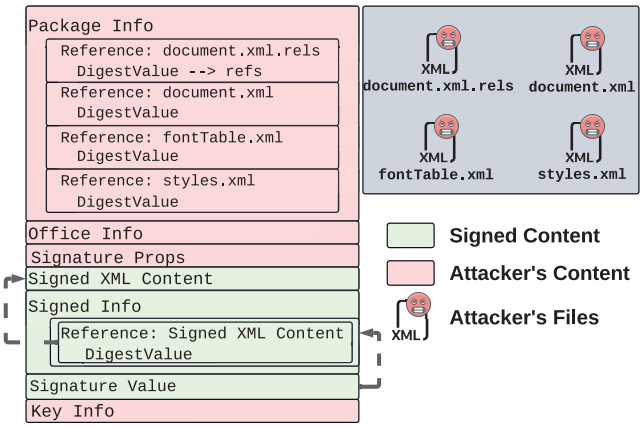

Figure 6: The attacker places a trustfully signed XML content and reference it in *Signed Info*. Only this content will be verified. Thus, the integrity verification of all files within the package is skipped.

Attack Requirements. We assume that the attacker possesses a valid XML signature of a trusted entity. The attacker could extract a suitable XML signature from any other application that uses XML Signatures, such as signed OpenDocument Format (ODF) documents or SAML authentication tokens. This requirement differs to other attacks. The attacker neither needs a trustfully signed document nor a signing oracle . They only need an XML Signature from any other source that the victim trusts.

Manipulation Technique. The attacker creates a self-signed OOXML document with arbitrary content. However, instead of referencing the *Signature Properties*, *Package* and *Office Info*, the attacker pastes a valid XML signature and the signed content extracted from a different data format than OOXML, for example, SAML.

The cryptographic signature verification processes the *Signed Info* which does not reference the *Package Info*. Thus, the attacker can change all digest values to correspond to the modified content.

Manipulation Steps.

(1) The attacker creates a self-signed OOXML document with arbitrary content. The self-signing process is only used to let the application automatically create the correct references and hash values in *Package Info*.
(2) The attacker extracts the signature file from the OOXML package.
(3) The attacker replaces the *Signed Info*, *Signature Value*, and *Key Info* with a valid XML signature of a trusted entity. For example, from a signed ODF document. If the new *Signed Info* element contains references to internal objects within the XML signature, these must also be included.
(4) The attacker adds the thus manipulated signature file back to the OOXML package.

Impact. If the victim opens the attacker-manipulated OOXML document, the office application verifies the stored digest values in *Package Info* against the existing files. Since these hash values match the content created by the attacker, this check is successful. Since *Signed Info* is taken from a valid XML signature of a trusted entity, this validation is also successful. Thus, the application presents a valid signature over the document to the victim. The certificate's trusted entity contained in *Key Info* is shown as the signer.

6.2 Malicious Repair Attack (MRA)

These attacks abuse Microsoft Office's repair feature to spoof a trusted entity's signature over attacker-controlled content in automatically created temporary documents.

Attack Requirements . The attacker needs an OOXML document signed by a trusted entity.

Manipulation Technique. This attack abuses the fact that only the relationships present at the signing time are signed. Thus, it is possible to add more relationships while preserving the signature. This allows an attacker to create documents containing signed but not rendered files.

Duplicate Document Attack (DDA). For this attack, the attacker, obtains a validly signed OOXML document from a trusted entity. Then, the attacker prepares a new OOXML document which should be displayed instead. The attacker renames all files in the new document such that they do not collide with the default names. Finally, the attacker adds the

files of the newly created OOXML package to the signed document and inserts the corresponding references to the `[Content_Types].xml`. The attacker also adds a new relationship with the target of the new `document.xml`. This is allowed since the relationship file is only partially signed. The attacker copies the `document.xml.rels` file from the attacker's document and renames it so that the name of the relationship file matches the renamed `document.xml`.

Evil Type Attack (ETA). The attacker uses a signed Excel or PowerPoint document to inject the contents of a Word package into it. Using Microsoft Office's automatic repair feature, the victim is displayed the content of the Word package while the signature is calculated over the original Excel or PowerPoint document and thus remains valid.

To execute the attack, the attacker obtains a valid signed Excel or PowerPoint document from a trusted entity. The attacker creates a Word document whose content is freely definable. Then, the attacker copies the Word-related files into the signed Excel or PowerPoint document. The attacker copies the content types of Word-related files into the `[Content_Types].xml` of the signed document. The attacker inserts the references of the Word package files into the relationship file of the signed Excel or PowerPoint document. Finally, the attacker changes the file extension to `.docx`

Impact. When the victim opens the manipulated document, Microsoft Office prompts to repair the file. Through the automated repair process, Microsoft Office creates a new temporary document. The repair occurs because of the duplicate file structure contained in the document. After the repair process, the application displays only the content chosen by the attacker. However, the signature verification is performed over the content of the original document. Thus, a valid signature of a trusted entity is displayed to the victim.

Limitation. A limitation of the attack exists as soon as the victim saves the document repaired by Microsoft Office. In this case, only the content chosen by the attacker is saved, while the signature is kept over the original content. This results in an invalid signature when the signed document is reopened.

7 Evaluation

We evaluated our attacks against all versions of Microsoft Office and OnlyOffice Desktop which are still in the product lifecycle. For Microsoft Office, this includes the Windows versions 2013, 2016, 2019, 2021, 365, as well as Microsoft Office 2019, 2021, 365 for macOS. For OnlyOffice Desktop, this includes the 7.1 versions on Windows, macOS, and Linux. In this section, we describe the test environment and the results of our evaluation. Table 1 provides a summary of the results.

Test Environment. We divided our test environment into three system landscapes of Virtual Machines (VMs) based on Windows 10 and Ubuntu 22.04.1, as well as a hardware environment based on macOS Monterey. The signing oracle relies on a VM with Microsoft Office 2019 installed, which is configured with a private key and public certificate. The attacker's system deploys Microsoft Office version 2019 in a VM and does not have access to the signer's private key. The victim's systems consist of different VMs, each of which has one of the examined Microsoft Office or OnlyOffice Desktop versions installed and trusts the public certificate of the signing system. Furthermore, the victim system also relies on a hardware environment with macOS incl. the examined versions of Microsoft Office and OnlyOffice Desktop.

Excluded Applications. During our analysis, we analyzed other popular office applications such as Google Workspace, LibreOffice, OpenOffice, Collabora Office, NeoOffice and the Digital Signature Services (DSS). Google Workspace, OpenOffice, NeoOffice and DSS were excluded from the evaluation due to a lack of native support for OOXML signatures. LibreOffice and Collabora Office recognize signed OOXML documents, but even for validly signed OOXML documents a warning is thrown that the signature has problems. This is due to the default handling of partial signatures in LibreOffice. This makes it impossible to evaluate the attacks, since there is no status of a valid signature for signed OOXML documents in these applications. For this reason, LibreOffice and Collabora Office were also excluded from the evaluation.

Microsoft Office for Windows. When evaluating the attacks described in Section 5 and Section 6, we did not find any difference in the processing of signatures between the tested Windows versions of Microsoft Office, so the evaluation showed a consistent result. To determine the success of an attack, we evaluated the different UI layers of Microsoft Office for each attack vector. Here, the attack must result in the signature state of a valid trusted signature (1. a) or (1. b) (see Figure 2). In both signature states, UI layers 1 and 2 are identical. The victim is displayed the following message under UI layer 2:

"This document has been signed and marked as final. It should not be edited. If anyone tampers with this document, the signatures will become invalid."

UI layer 3 indicates a partial signature in combination with (1. b), but the victim has no way to investigate which parts of the document the partial signature refers to.

CIA: This attack allows attacker-controlled content to be added to the signed document. However, it is not possible to remove the existing content without invalidating the signature. We solved this limitation by using text boxes and graphics to place new content over the original one. When the document is opened, a valid trusted signature is displayed in the UI layer combination (1. b). For the victim, only the attacker's content is visible due to the overlay. A limitation exists if the entire content is selected and copied by using `CTRL+A` / `CTRL+C`. This would copy the entire content, including the original content of the document.

	Microsoft Office	Build	Specification Flaws: CIA	Specification Flaws: Content Masking Attack – Font Inj.	Specification Flaws: Content Masking Attack – Style Inj.	Specification Flaws: Legacy Wrapping	Implementation Flaws: USF	Implementation Flaws: Malicious Repair Attack – Dup. Doc	Implementation Flaws: Malicious Repair Attack – Evil Type
Windows	2013	15.0.5423.1000	⊗	⊗	⊗	⊗	⊗	◐	◐
	2016	16.0.5278.1000	⊗	⊗	⊗	⊗	⊗	◐	◐
	2019	16.0.10386.20017	⊗	⊗	⊗	⊗	⊗	◐	◐
	2021	16.0.14332.20303	⊗	⊗	⊗	⊗	⊗	◐	◐
	365	16.0.15028.20248	⊗	⊗	⊗	⊗	⊗	◐	◐
macOS	2019	16.61.22050700	⊗. Direct content manipulation without any detection						
	2021	16.61.22050700	⊗. Direct content manipulation without any detection						
	365	16.61.22050700	⊗. Direct content manipulation without any detection						
	OnlyOffice Desktop								
	Windows	7.1.1.57	⊗	✓	◐	⊗	✓	⊗	⊗
	macOS	7.1.1 (533)	⊗	✓	◐	⊗	✓	⊗	⊗
	Linux	7.1.1.57	⊗	✓	◐	⊗	✓	⊗	⊗

Legend ✓: Not Vulnerable ⊗: Vulnerable ◐: Limited Vulnerability

Table 1: All Microsoft Office Variants in Windows are vulnerable to signature forgery attacks. For USF, Microsoft Office shows *valid signatures* and for the remaining attacks *valid partial signature* (cf. Figure 2) when the victim opens the manipulated documents. Only the Malicious Repair Attacks (MRAs) require user interaction to repair the signed document so that we count the applications *limited vulnerable*. macOS does not verify the signature at all so that attackers can simply change the content in `document.xml` – Microsoft Office on macOS always displays that the document is protected by a signature. OnlyOffice is fully vulnerable to the CIA, LWA and Malicious Repair attacks. The Style Injection attack allows hiding content under OnlyOffice, new content cannot be added, thus OnlyOffice is only limited vulnerable here.

CMA: In this attack class, a part or the entire content of the document controlled by the attacker can be changed in its presentation using style elements or prepared custom fonts. When opened, Microsoft Office only displayed the attacker's intended content to the victim, while the signature state remains valid and trusted in the UI layer combination (1. b). Content Masking Attack (CMA) has the same limitation as CIA: `CTRL+A` / `CTRL+C` can reveal the original content.

LWA: When opening a signed document based on LWA, the attacker-controlled content is displayed while the signature is considered valid and trusted, in the combination UI layer (1. b), by Microsoft Office. The original signed content of the document based on the old binary format is not visible.

USF: In this attack, the attacker controls the complete document content of the OOXML package. When the document is opened, there is no indication of tampering and the UI layer combination (1. a) is displayed. As a signer, the original issuer of the XML signature is displayed to the victim, even if the signature does not protect the document content in any way.

Repair attacks: This attack class requires user interaction to start the automated repair process. After the repair, Microsoft Office displays the attacker's injected document content to the victim while the signature status of a trusted valid signature is displayed in UI layer combination (1. b). The valid signature status is preserved only for the temporary repaired document and is invalidated by saving the document. Thus, we classify these attacks as limited.

Microsoft Office for macOS. We evaluated the macOS variants of Microsoft Office and expected to find the same results as for Windows. To our surprise, we were proved wrong: macOS does not correctly validate signatures at all. We used versions 2019, 2021 and 365, each with build number 16.61.22050700. If a validly signed unmanipulated OOXML document is opened under macOS, the UI layers of the Windows variants described in Figure 2 differ. UI layer 1 shows that the document is protected by a signature (see Figure 7). Microsoft Office also displays an icon of a signed document at the bottom of the application with the following tooltip:

"This document contains signatures."

Further UI layers, which display, for example, the signer or the signing date, do not exist in the macOS versions of Microsoft Office. Still, the document is locked for editing by

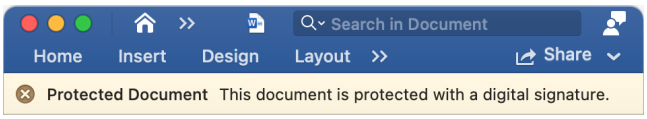

Figure 7: When opening an attacker-manipulated signed document under macOS, the UI layer 1 shows that the document is protected with a signature, even if the signature file has no content.

the graphical user interface and displayed in a read-only state. The hints of a protected document indicate that Microsoft Office cryptographically verifies the document's signature. However, a direct manipulation of the signed content within the `document.xml` file of the OOXML package showed that tampering is not detected. The signature hint messages remain identical. In the next step, we removed all the content from the signature file `sig1.xml`. This removal showed that Microsoft Office under macOS displays the same indication of a signed document, even though there is no longer any signature-relevant information in the signature file. Manipulation of signed OOXML documents under the macOS variants of Microsoft Office is thus trivial and does not require any particular attack vectors as presented for the Windows variants.

OnlyOffice Desktop. For the evaluation of OnlyOffice, we analyzed the versions on Windows, Linux and macOS. We obtained the same results for all versions. OnlyOffice uses only one UI layer for the validation status of signatures. This UI layer is similar to the UI layer 3 of Microsoft Office (see Figure 2). Here, the signer of the document as well as the status of the signature valid (see Figure 2 → (a)) or invalid (see Figure 2 → (d)) is displayed to the user. Only attacks resulting in a valid signature status are classified as vulnerable.

CIA: For OnlyOffice, the attack was slightly improved and allows attackers not only to add new content to a signed document, but also to completely remove the old content from the document view. For this purpose, the malicious `people.xml` file is defined as a second main document in `_rels/.rels` with `Id0`. Due to the lower ID in contrast to the `document.xml`, the malicious `people.xml` is chosen as the main document by OnlyOffice. Thus, only the content of `people.xml` is displayed and the originally signed content is hidden. The attack thus exploits the possibility given by the specification to manipulate the relationship file subsequently, as well as an implementation flaw of OnlyOffice that allows two correctly referenced main documents in one OOXML package. Only the malicious content is displayed to the victim, while the signature remains valid.

CMA: OnlyOffice does not support embedded fonts and thus is not vulnerable to the FIA. SIA can be used under OnlyOffice to subsequently hide predefined content by applying style elements. Since OnlyOffice processes content such as text or graphics only within the main document, the attack cannot be used to inject new content. Thus, OnlyOffice has limited vulnerability to this attack.

LWA: When the manipulated document is opened, a valid signature is displayed to the victim. The content controlled by the attacker is displayed, while the originally signed content remains completely hidden.

USF: During the signature validation OnlyOffice checks if the *Signed Info* element contains a reference to the *Package Info* element. This makes OnlyOffice not vulnerable to this attack.

Repair attacks: Unlike Microsoft Office, OnlyOffice does not display a repair prompt. The malicious content is displayed directly and the signature remains valid for both attack variants. Thus, OnlyOffice is vulnerable to this attack class.

Additional Findings. We additionally tested all versions of OnlyOffice Desktop and Microsoft Office (Windows) listed in Table 1 against known attacks on XML signatures (see Table 2). Since Microsoft Office does not perform any signature verification on macOS, no special attack vectors are necessary. While evaluating the known XML signature attacks under OnlyOffice and the Windows versions of Microsoft Office, we could not identify any vulnerabilities in this regard. However, in the course of our evaluation, we were able to identify a feature in Microsoft Office that can also be used for signed OOXML documents, which is problematic with regard to the integrity of the document. Graphics can be added to the document as external resources through a URL. When the document is opened, the graphic is automatically reloaded. Since only the URL is signed when signing the OOXML document, an attacker can change the image displayed in the document by replacing the graphic stored under the URL. The signature remains valid, even if the displayed content of the document differs. OnlyOffice does not support graphics dynamically reloaded via URL and is therefore not vulnerable.

Attack	**MS Office** (Win) **2013-2021, 365**	**OnlyOffice Desktop**
External Resources [23, 24]	⊗	✓
Signature Exclusion [18]	✓	✓
Certificate Faking [15]	✓	✓
Node Splitting [19, 20]	✓	✓
Key Confusion [25, 26]	✓	✓
HMAC Truncation [21]	✓	✓
Signature Wrapping [17, 18]	✓	✓

Legend ✓: Not Vulnerable ⊗: Vulnerable ◐: Limited Vulnerability

Table 2: Additonal Findings: No vulnerabilities were found for known XML signature attacks for all versions of OnlyOffice and Microsoft Office (Windows). External resources that contain graphics via a URL can be replaced on Microsoft Office (Windows) after signing without invalidating the signature.

8 Countermeasures

We divided the attacks into specification flaws and implementation flaws. Various countermeasures are conceivable here and will be discussed below.

Specification Flaws. The *CIA*, *CMA* and *LWA* attacks exploit the fact that the standard supports partial signatures. The main problem comes from the relationship files, which are not signed as a whole, but only the included references during the signing flow. By adding additional files and including their correct references, the rendered content of the signed document can be modified. This also takes advantage of the fact that the standard does not restrict rendered content to the main part `document.xml`. This means that content from other files, such as `style.xml` or `people.xml`, can also be used to inject visible content.

From a security point of view, the question arises why rendered content should be modifiable at all after signing. Therefore, the standard should exclude the variant of partial signature support and at least include the relationship files completely in the signature calculation to prevent referencing of files added afterwards. Accordingly, the XML schema files should be used more restrictive. Thus, unsigned files could not place content over the original content defined in `document.xml`

Implementation Flaws. *USF* forms a very powerful attack, since the attacker controls the entire content of the signed document without any restrictions. The problem here is the lack of verification that the *Package Info*, which contains all the references and hash values for the files within the OOXML package, is actually referenced within the *Signed Info* element. If the referencing is missing, the attacker can arbitrarily choose the content of the document by simply generating the hash values over the files without performing any signature computation. The effective countermeasure is a mandatory check for the presence of this reference in the *Signed Info* element.

The *MRA* exploits the flexibility of office applications in processing of supposedly damaged documents. Here, it would be conceivable to dispense with the repair of signed documents at all or, alternatively, to have the signature check performed before the repair attempt. In the case of the preceding signature check, the subsequently inserted files would lead to an invalid signature evaluation.

9 Related Work

In this section we systematize the related work and highlight the contributions of our paper. Additionally, we summarize the prior work paving the road for the attacks described in this paper in Table 3.

Attacker's Capabilities and Targets. In addition to the capabilities and targets described in Section 3, there are further

Technique	Cap.	Target	Applicability PDF	ODF	OOXML
Untrusted Keys			[23]	[12, 27, 28]	✗
Unsigned Content Inj.			[9, 11]	[14, 29]	[28–30]
Signature Wrapping			[9]	[12]	✗
Shadow Attacks			[10]	[12]	✗

Capabilities
- Public Knowledge
- Trustfully Signed Document
- Signing Oracle

Target
- Data Manipulation
- Code Execution

Table 3: Our analysis on the related work reveals a gap in the prior work on OOXML documents. There is no related work to three of four attack techniques. The attacks targeting the unsigned content injection shows only how to manipulate the metadata of signed documents.

possible attacker models described in previous research.

Capability: Public Knowledge ( ). The attacker has only access to publicly available data. This public knowledge includes, inter alia, the specification of a document format (e.g., OOXML) and corresponding sub-specifications (e.g., XML). The attacker could create cryptographic keys, for example, to sign the document, but the victim does not trust these keys. Similarly, the attacker knows which public keys the victim trusts. Nevertheless, the attacker does not have any private keys that the victim trusts.

Target: Code Execution ( ). The attacker can run arbitrary code on the victim's computer. Usually, this target is commonly used in malicious documents exploiting insecure features [31] or implementation bugs. In the years 2006 to 2022, several researches focusing on ODF security have been published [12, 14, 28, 29, 32, 33]. The authors analyzed the security of OpenOffice.org respectively OOXML and proposed different attack and obfuscation techniques to stealthy execute malicious code. The authors highlighted security issues in the design of ODF and also problems with digital signatures. In 2015, Lax *et al.* documented potential security topics related to digitally signed documents [24]. The authors concentrated on issues related to the signature generation process, signed documents containing dynamic content, and polymorphic documents similar to [34]. Recently, a research group discovered several possibilities to execute malicious code by spoofing signed PDF and ODF documents [10–12].

Beyond attacks spoofing document's content, several researchers discovered possibilities to inject malicious macros and mask them as a signed content [35–41]. Due to diversities between signing document's content and macro code in OOXML, we concentrate in this paper on attacks spoofing the document's content.

Attack Techniques. Attack techniques are the glue between the attacker's capabilities and the attacker's targets. They represent the concrete approach of how the attacker carries out a specific attack. While studying the related work, we

extracted the different attacks, systematized, and categorized them in the following categories.

Untrusted Keys. In 2002, Kain *et al.* described possible risks related to digitally signed documents like Microsoft Word, Microsoft Excel, or PDF. The core of the described issues lies in PKI problems, dynamic content loaded from a website, and code execution by supported programming languages within documents [23]. In 2007, Kasinath and Armstrong discusses the PKI applicability on digitally signed and encrypted ODF documents [27]. In 2022, Rohlmann *et al.* systematically analyzed the ODF specification and discovered, among other attacks, the possibility to sign ODF documents with untrusted keys that are successfully verified without any warnings.

Unsigned Content Injection. This technique summarizes attacks appending or injecting new content after the document is being signed. The injection is made outside the signed area. Thus, the cryptographic signature value remains valid while malicious content is rendered after opening the document. In the recent years multiple attacks on PDFs were reported [9, 11]. In 2022, Rohlmann *et al.* applied similar attacks on ODF documents without any success [12]. The reason for the results was the strict signature validation detecting files added after signing the document. In 2007, a CVE abusing partial signatures in OOXML documents was reported [30]. The authors discovered that the metadata of a signed OOXML document is not protected and thus can be changed by attackers. Additional analysis regarding potential risks in OOXML documents have been provided by Pöhls and Westphal and Filiol [28, 29]. We extended the attack ideas and applied them on the current OOXML specification. The Content Injection Attack, Duplicate Document Attack, and Evil Type Attack rely on the concepts of this attack technique.

Shadow Attacks. In 2021, a new attacker model applied on PDF documents was introduced [10]. One year later, the same attacker models was applied on ODF documents without any success [12]. We are the first, adapting this attacker model on OOXML and succesfully discovering new attacks – Font Injection Attack, Style Injection Attack, and Legacy Wrapping Attack.

Signature Wrapping. The general concept of wrapping attacks has been applied to XML-based messages before – the attack allows the relocation of the hashed part of a document and the injection of malicious content. In 2005, McIntosh and Austel described an XML rewriting attack on SOAP web services [17]. Somorovsky *et al.* extendend the attack and adapted it on authentication protocols [18]. With respect to documents, Mladenov *et al.* discovered possibility to carry out the attack on signed PDF documents [9]. Two years later, Rohlmann *et al.* successfully applied the attack on ODF documents [12]. To the best of our knowledge, we are the first evaluating the attack on OOXML. Our study revealed the Universal Signature Forgery based on the concept of wrapping attacks.

10 Discussion and Future Research

Status Quo. Through our research, we could discover potential security issues abusing partial signatures given by the standard. We prove the applicability of our attacks with a practical evaluation bypassing the integrity verification of *all* implementations. We extended our research by developing new attacks targeting implementation errors and testing known attacks on XML signatures. We discovered several critical implementation errors abusing lack in the OOXML signature verification. Luckily, the developers learned from previous vulnerabilities focusing explicitly on XML signatures and addressed the risks properly, see Table 2.

Flawed Specifications. It is surprising that security research focusing on older OOXML specifications and clearly describing risks related to partial signatures have been ignored for decades. The result is devastating. We extended the existing concepts and systematically prove the flaws in the current OOXML specification and implementations.

Considering other document formats such as PDF, we showed that partially signed documents decrease the security of the integrity protection while increasing the attack surface. Thus, it is questionable whether the integrity of the content of a signed document can ever be guaranteed if the standard flexibly supports partial signatures. Therefore, standards should follow a stricter signature policy and exclude partial signatures. We are encouraged in our view by the ODF standard, which enforces full signature coverage except for the signature file and the `external-data` directory [42, Part 3, p. 98]. Thus, attackers rely only on implementation errors which can be fixed.

Automated Analysis. Currently, there is no automated approach to analyze, create and evaluate the security of digitally signed documents. In a long-term, the community needs a more sustainable approach to generate and test attack vectors.

Formal Model. In 2007, [14] described a graph based model depicting the changes on a document [14]. This model seems a good starting point for the formalization of document's states and changes. Another formal model was introduced for PDF documents in 2021 [43]. Based on a formal model, expected states and deviations can be documented and later evaluated. Also manipulations and unwanted behaviour can be modelled. Currently, state machine learning is applied on cryptographic protocols to evaluate the security [44]. Similar approach could be applicable on documents, too.

Tool-support. Currently, there is no tool automating the generation of document-based attacks and evaluating the results. There exist only isolated solutions solving one explicit problem in one specific document format. A suitable starting point for a generic *Document Attacker*-tool could rely on the similarities between ODF and OOXML. An automated tool could automatically learn the document's structure, transfer it in a model and adapt all known attacks on this model. Thus, *all*

XML-based document formats, such as ODF, OOXML, 3MF, and CAD, could be covered automatically for each known attack.

Cryptographic Attacker Models. The standard cryptographic security assumption for digital signature schemes is EUF-CMA [45]. Here the attacker is allowed to choose a sequence of messages that will be signed, and they succeed if they can compute a valid signature for some arbitrary message m not in this sequence. A valid signature s for a given message m is defined as satisfying $\mathsf{Sig.Vrf}(pk, m, s) = TRUE$, where pk is the public signature verification key. This cryptographic security assumption roughly matches our *Signature Oracle* (see Section 3) based attacks. Our attacks on *Trustfully Signed Documents* (see Section 3) do not require the attacker to be able to *choose* the messages, it is sufficient if they *know* a signed message. Thus, the attacker model closest to these attacks is EUF-KMA [46]. Still, our attacks break neither security assumption: In both cases, the term *message* refers to the byte string that is actually hashed – we do not modify this byte string, but add additional bytes that are not hashed.

A formal model for our attacks – and, in fact, for all UI-based signature verification applications (S/MIME, OpenPGP, PDF, ...) – would have to replace the cryptographic function $\mathsf{Sig.Vrf}()$ with a more complex construct $\mathsf{View.Sig.Vrf}()$. This new function must take into account how the message itself and the result of the signature validation are displayed in the UI at different UI levels. Such a formal model would certainly be helpful in understanding UI-based verification of digital signatures.

Acknowledgment

Simon Rohlmann was supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) project "Industrie 4.0 Recht-Testbed" (13I40V002C). This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[1] Microsoft Corporation. "Transcript: Microsoft FY21 Third Quarter Earnings Conference Call." (2021), [Online]. Available: https://view.officeapps.live.com/op/view.aspx?src=https://c.s-microsoft.com/en-us/CMSFiles/TranscriptFY21Q3.docx?version=5000c435-108f-a064-0035-be555b8a57ff.

[2] Ecma International, *ECMA-376, Office Open XML file formats, 5th edition (Part 1-4)*, 2015-2021. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-376/.

[3] Microsoft Corporation. "Digital signatures and certificates." (2022), [Online]. Available: https://support.microsoft.com/en-us/office/digital-signatures-and-certificates-8186cd15-e7ac-4a16-8597-22bd163e8e96.

[4] *XML-Signature Syntax and Processing*, Feb. 2002. [Online]. Available: https://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.

[5] Defense Counterintelligence and Security Agency. "Signed Document by the Defense Counterintelligence and Security Agency." (2021), [Online]. Available: https://www.dcsa.mil/Portals/91/Documents/IS/DCII/Documentation/Agency_Request_Form_DCII_TEMPLATE_Aug2021.docm.

[6] Government of Canada. "E-Signature Options 2020-04." (2020), [Online]. Available: https://wiki.gccollab.ca/E-Signatures_in_the_GC/E-Signature_Options_Blog_2020-04.

[7] Office of the Federal Register's (OFR). "Federal Register Document Submission Portal." (), [Online]. Available: https://webportal.fedreg.gov/(S(i0ruumi41rz5ao2ftsvkqof2))/resources/howtosubmit.pdf.

[8] The Federal Identity, Credential, and Access Management (FICAM) program. "Digitally Sign a Microsoft Word Document." (), [Online]. Available: https://playbooks.idmanagement.gov/playbooks/signword/.

[9] V. Mladenov, C. Mainka, K. Meyer zu Selhausen, M. Grothe, and J. Schwenk, "1 Trillion Dollar Refund: How To Spoof PDF Signatures," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: ACM, Nov. 2019, pp. 1–14, ISBN: 9781450367479. DOI: 10.1145/3319535.3339812. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3339812.

[10] C. Mainka, V. Mladenov, and S. Rohlmann, "Shadow Attacks: Hiding and Replacing Content in Signed PDFs," in *Proceedings 2021 Network and Distributed System Security Symposium*, Reston, VA: Internet Society, 2021, ISBN: 1-891562-66-5. DOI: 10.14722/ndss.2021.24117. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/shadow-attacks-hiding-and-replacing-content-in-signed-pdfs/.

[11] S. Rohlmann, V. Mladenov, C. Mainka, and J. Schwenk, "Breaking the Specification: PDF Certification," in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, May 2021, pp. 1485–1501, ISBN: 978-1-7281-8934-5. DOI: 10.1109/SP40001.2021.00110. [Online]. Available: https://ieeexplore.ieee.org/document/9519390/.

[12] S. Rohlmann, C. Mainka, V. Mladenov, and J. Schwenk, "Oops... Code Execution and Content Spoofing: The First Comprehensive Analysis of OpenDocument Signatures," in 31st *USENIX Security Symposium (USENIX'22)*, Ruhr University Bochum, Boston, MA, 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/rohlmann.

[13] International Organization for Standardization (ISO), *ISO/IEC 29500, Document description and processing languages — Office Open XML file formats, 4th edition (Part 1-4)*, 2015-2021.

[14] E. Filiol and J.-P. Fizaine, "OpenOffice security and viral risk – part one," in *Virus Bulletin Journal*, Sep. 2007. [Online]. Available: https://www.virusbulletin.com/virusbulletin/2007/09/openoffice-security-and-viral-risk-part-one.

[15] C. Mainka, V. Mladenov, F. Feldmann, J. Krautwald, and J. Schwenk, "Your Software at My Service: Security Analysis of SaaS Single Sign-on Solutions in the Cloud," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, Oct. 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2664168.2664172.

[16] A. Munoz and O. Mirosh, *SSO Wars: The Token Menace*, 2019. [Online]. Available: https://www.blackhat.com/us-19/briefings/schedule/index.html#sso-wars-the-token-menace-15092.

[17] M. McIntosh and P. Austel, "XML Signature Element Wrapping Attacks and Countermeasures," in *Proceedings of the 2005 Workshop on Secure Web Services*, ser. SWS '05, Fairfax, VA, USA: Association for Computing Machinery, 2005, pp. 20–27, ISBN: 1595932348. DOI: 10.1145/1103022.1103026. [Online]. Available: https://doi.org/10.1145/1103022.1103026.

[18] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On Breaking SAML: Be Whoever You Want to Be," in *21st USENIX Security Symposium*, Bellevue, WA, Aug. 2012. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final91.pdf.

[19] K. Ludwig, "Identity Theft: Attacks on SSO Systems," in *BlackHat USA*, Aug. 2018. [Online]. Available: https://i.blackhat.com/us-18/Thu-August-9/us-18-Ludwig-Identity-Theft-Attacks-On-SSO-Systems.pdf.

[20] RedTeam Pentesting GmbH. "Truncation of SAML Attributes in Shibboleth 2." (2017), [Online]. Available: https://www.redteam-pentesting.de/de/advisories/rt-sa-2017-013/-truncation-of-saml-attributes-in-shibboleth-2.

[21] "XML Signature HMAC Truncation Authentication Bypass Vulnerability." (2009), [Online]. Available: https://www.kb.cert.org/vuls/id/466161.

[22] M. Corporation, *[MS-CFB]: Compound File Binary File Format*, 2021. [Online]. Available: https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-CFB/%5bMS-CFB%5d.pdf.

[23] K. Kain, S. W. Smith, and R. Asokan, "Digital signatures and electronic documents: A cautionary tale," in *Advanced communications and multimedia security*, Springer, 2002, pp. 293–307. [Online]. Available: http://www.ists.dartmouth.edu/library/74.pdf.

[24] G. Lax, F. Buccafurri, and G. Caminiti, "Digital document signing: Vulnerabilities and solutions," *Information Security Journal: A Global Perspective*, vol. 24, no. 1-3, pp. 1–14, 2015.

[25] T. McLean, *Blog post: Critical vulnerabilities in JSON Web Token libraries*, 2015. [Online]. Available: https://www.chosenplaintext.ca/2015/03/31/jwt-algorithm-confusion.html.

[26] M. Heiderich *et al.*, *Pentest & Audit-Report SimpleSAMLphp 11.2017*, 2017. [Online]. Available: https://cure53.de/pentest-report_simplesamlphp.pdf.

[27] G. Kasinath and L. Armstrong, "Analysis of PKI as a Means of Securing ODF Documents," in *Proceedings of 5th Australian Information Security Management Conference*, Perth, Jan. 1, 2007. [Online]. Available: https://ro.ecu.edu.au/ecuworks/4951.

[28] E. Filiol, "OpenOffice v3.x Security Design Weaknesses," in *Black Hat Europe*, Apr. 2009. [Online]. Available: https://www.blackhat.com/presentations/bh-europe-09/Filiol_Fizaine/BlackHat-Europe-09-Filiol-Fizaine-OpenOffice-Weaknesses-slides.pdf.

[29] H. C. Pöhls and L. Westphal, "Die "Untiefen" der neuen XML-basierten Dokumentenformate," in *15. DFN CERT Workshop Sicherheit in vernetzten Systemen*, 2008. [Online]. Available: http://henrich.poehls.com/papers/2008_Poehls_Westphal_2008_DFN-CERT-WS_Untiefen_der_XML-Dokumentenformate.pdf.

[30] H. Poehls, D. Tran, F. Petersen, and F. Pscheid, *MS Office 2007: Digital Signature does not protect Meta-Data*, 2007. [Online]. Available: https://cxsecurity.com/issue/WLB-2007120035.

[31] J. Müller, D. Noss, C. Mainka, V. Mladenov, and J. Schwenk, "Processing Dangerous Paths – On Security and Privacy of the Portable Document Format," in *Proceedings 2021 Network and Distributed System Security Symposium*, Internet Society, 2021, ISBN: 1-891562-66-5. DOI: 10.14722/ndss.2021.23109. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_1B-2_23109_paper.pdf.

[32] D. de Drézigué, J.-P. Fizaine, and N. Hansma, "In-depth analysis of the viral threats with OpenOffice.org documents," in *Journal in Computer Virology*, vol. 2, Dec. 2006, pp. 187–210. DOI: 10.1007/s11416-006-0020-2. [Online]. Available: https://doi.org/10.1007/s11416-006-0020-2.

[33] P. Lagadec, "OpenDocument and Open XML security (OpenOffice.org and MS Office 2007)," in *Journal in Computer Virology*, vol. 4, May 2008, pp. 115–125. DOI: 10.1007/s11416-007-0060-2. [Online]. Available: https://doi.org/10.1007/s11416-007-0060-2.

[34] D.-S. Popescu, "Hiding Malicious Content in PDF Documents," *CoRR*, vol. abs/1201.0, 2012. arXiv: 1201.0397. [Online]. Available: http://arxiv.org/abs/1201.0397.

[35] V. Bontche. "pcodedmp.py - A VBA p-code disassembler." (2019), [Online]. Available: https://github.com/bontchev/pcodedmp.

[36] H. Ogden, K. Sayre, and C. Roberts, "VBA Stomping Advanced Malicious Document Techniques," in *DerbyCon*, 2018. [Online]. Available: https://github.com/clr2of8/Presentations/blob/master/DerbyCon2018-VBAstomp-Final-WalmartRedact.pdf.

[37] P. Ceelen and S. Hegt, "MS OFFICE FILE FORMAT SORCERY," in *Troopers*, 2019. [Online]. Available: https://github.com/outflanknl/Presentations/blob/master/Troopers19_MS_Office_file_format_sorcery.pdf.

[38] K. Sayre and C. Roberts, "Advanced Malware VBA Stomping," in *Sp4rkCon*, 2019. [Online]. Available: https://github.com/clr2of8/Presentations/blob/master/Sp4rkCon2019-VBAstomp.pdf.

[39] Didier Stevens. "Tampering with Digitally Signed VBA Projects." (), [Online]. Available: https://blog.nviso.eu/2020/06/04/tampering-with-digitally-signed-vba-projects/.

[40] Yu Kaijun. "Upgrade signed Office VBA macro projects to V3 signature." (2021), [Online]. Available: https://developer.microsoft.com/en-us/sharepoint/blogs/upgrade-signed-office-vba-macro-projects-to-v3-signature/.

[41] P. Lagadec and P. Lagadecl, "Advanced VBA Macros Attack & Defence," in *Black Hat Europe*, 2019. [Online]. Available: https://www.decalage.info/files/eu-19-Lagadec-Advanced-VBA-Macros-Attack-And-Defence.pdf.

[42] OASIS Open, *Open Document Format for Office Applications (OpenDocument) Version 1.3*, Apr. 2021. [Online]. Available: https://docs.oasis-open.org/office/OpenDocument/v1.3/.

[43] P. Wyatt, "Demystifying pdf through a machine-readable definition," in *LangSec Workshop at IEEE Security & Privacy*, 2021. [Online]. Available: https://github.com/pdf-association/arlington-pdf-model.

[44] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of DTLS implementations using protocol state fuzzing," in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, pp. 2523–2540, ISBN: 978-1-939133-17-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/fiterau-brostean.

[45] M. Green. "EUF-CMA and SUF-CMA." (2018), [Online]. Available: https://blog.cryptographyengineering.com/euf-cma-and-suf-cma/.

[46] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on computing*, vol. 17, no. 2, pp. 281–308, 1988.

A Appendix

A.1 Partial Signatures in the OOXML Standard

In this paper, we claim that three attack classes are based on a standardized feature of OOXML, namely on the fact that the relationship files are only partially signed. In the following, we cite the relevant parts of the OOXML standard which substantiate our claim.

Mutability as a Design Goal. Section 10.3 of the standard describes *mutability* as a design goal of OOXML.

> **10.3 Choosing content to sign** [2, Part 2, p. 45]
> *It is assumed that there is a signature policy to determine which parts and relationships to sign. This*

clause provides flexibility in defining the content to be signed, thus allowing other content to be mutable. [...]

For this purpose, the standard requires only the reference elements within a relationship file to be signed. The relationship file is only partially signed and thus enables the subsequent addition of reference elements.

Referencing Elements from the Relationships Files. The OOXML standard explicitly states how elements or groups of elements in the relationships file should be selected. This leaves no room of other types of referencing, for example the entire file.

> **10.5.9 `RelationshipReference` element** [2, Part 2, p. 50]
>
> The `RelationshipReference` element specifies which `Relationship` element is signed, and shall only occur as a child element of a `Transform` element representing a Relationships transform (10.5.8.2). This element is OPC-specific.

Attributes	Description
`SourceId`	The value of the `Id` attribute of the referenced `Relationships` element within the given Relationships part. This attribute is required. The range of values for this attribute shall be as defined by the `xsd:string` simple type of W3C XML Schema Datatypes.

As a second method, groups of elements can be selected based on their `ContentType`. Both referencing methods protect existing content against manipulation, but do not protect against inserting unsigned files to the OOXML document.