What Did You Add to My Additive Manufacturing Data?: Steganographic Attacks on 3D Printing Files

Mark Yampolskiy
Auburn University
USA

Lynne Graves
University of South Alabama
USA

Jacob Gatlin
Auburn University
USA

Anthony Skjellum
University of Tennessee at Chattanooga
USA

Moti Yung
Google LLC.
Columbia University
USA

ABSTRACT

Additive Manufacturing (AM) adoption is increasing in home and industrial settings, but information security for this technology is still immature. Thus far, three security threat categories have been identified: technical data theft, sabotage, and illegal part manufacturing. In this paper, we expand to a new threat category: misuse of digital design files as a subliminal communication channel. We identify and explore attacks by which arbitrary information can be embedded *steganographically* in the most common digital design file format, the *STL*, **without distorting the printed object**. Because the technique will not change the manufactured object's geometry, it is likely to remain unnoticed and can be exploited for data transfer. Further, even with knowledge of our methods, defenders cannot distinguish between actual data transfer and random manipulation of the files. This is the first info-hiding attack on this system, conducted despite the fact that random changes may spoil the physical artifact and result in detection.

CCS CONCEPTS

• **Applied computing** → **Computer-aided manufacturing**; • **Security and privacy** → **Cryptography**.

KEYWORDS

Information Hiding Attacks, Additive Manufacturing, 3D printing, File Transfers, Encryption, Cryptovirology, Steganographic channels, Traffic Analysis, Malware.

ACM Reference Format:
Mark Yampolskiy, Lynne Graves, Jacob Gatlin, Anthony Skjellum, and Moti Yung. 2021. What Did You Add to My Additive Manufacturing Data?: Steganographic Attacks on 3D Printing Files. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3471621.3471843

RAID '21, October 6–8, 2021, San Sebastian, Spain

ACM ISBN 978-1-4503-9058-3/21/10…$15.00
https://doi.org/10.1145/3471621.3471843

1 INTRODUCTION

Additive Manufacturing (AM), often referred to as *3D Printing*, is a computerized direct digital manufacturing technology in which three-dimensional objects are built up, layer by layer, based on the information specified in a design file [34]. Because of the numerous advantages that this technology brings, over the past 29 years AM enjoyed an impressive 26.6% average annual growth rate of worldwide revenues for products and services [64]. Over the last three years alone, the number of enterprise-class printer manufacturers has more than doubled [23]. Meanwhile, more than 1200 companies offer 3D printing as a service [2], allowing flexible on-demand manufacturing based on customer designs. Additionally, the hobbyist 3D-printing community is growing rapidly, a trend supported by a confluence of factors like falling desktop 3D printer prices, increased automation and quality of 3D-printed objects, as well as access to a wide variety of design files on the internet.

As a consequence of these developments, the design files used in 3D printing are becoming increasingly available. These files are downloaded from a variety of free or paid websites, uploaded to cloud services offered by printer manufacturers, and/or exchanged among friends or business partners via e-mail, making them a common payload in internet traffic. The same files are also routinely stored on personal or company computers along with the .docx, .pdf, .jpg, .mp3, and other common file formats.

The information security community is well aware of how easy it is to embed additional data in pictures, text, or video files. Even though the embedded covert information may change the underlying data, the introduced distortions are small enough they will be hard to detect by humans. The common nature of the files provides malware (Trojan) or other malicious actors an opportunity to "blend in" secret messages with normal network traffic. Considering the increased proliferation of digital design files for 3D printing, we must investigate whether these files can be similarly exploited as a covert channel for arbitrary information stealing.

The answer, however, is nontrivial. As opposed to images, text, and video files, 3D-printing design files are interpreted by software to produce (3D-print) carefully designed physical artifacts. Particularly with functional parts, the 3D-printed objects must adhere to tight tolerances. Alterations to design files can have detrimental effects on their ability to be integrated into a system, or impact the part's mechanical properties (known as *function*) [66]. Less obvious alterations of the 3D object printing orientation can also degrade

a part's function because of the inherent anisotropic properties of 3D printing [67, 73].

Changes to design files have already been exploited by intentional cyber-physical sabotage attacks [7, 56] and in a proposal to prevent part infringement [15]. In the case of the *dr0wned* attack [7], the failure of a sabotaged 3D-printed propeller mid-flight led to the destruction of the quad-copter UAV on which it was installed[1]. Observable impacts like this will draw attention to a covert communication channel, an outcome that a malicious actor normally would like to avoid.

Therefore, the central question we pose in this work is: How easy is it, despite the required accuracy for 3D printing, to use the exchange of 3D-printing files as a channel for arbitrary information stealing? Then, if yes, can this information be hidden? (Note that important trade secrets can reside in the environment of 3D-printing files, so covert information hiding is very tempting!) In this paper, we answer this question for STL (*STereoLithography*) files, which are the most widely adopted digital design file format used in AM [18], both in industrial and hobbyist settings.

The remainder of this paper is organized as follows. In Section 2 we introduce considered attack characteristics, including the concepts of steganography and subliminal channels, selected use cases for the considered attack, and an outline of mechanisms that determine the strength of attack. In Section 3, we provide further background necessary for the discussion of our proposal. This includes a brief introduction of the STereoLithography (STL) design file format, and an overview of the related literature. In Section 4, we present our proposal for encoding individual bits into an STL file, then show how it can be scaled up to a raw steganographic channel that can encode or decode bytes while updating internal positioning in a carrier STL file. Then, we present how an arbitrary binary file can be stored in and recovered from a carrier STL file using the defined functions. In Section 5, we present our proposal for the strong attack, which constitutes a fully encrypted unidirectional steganographic covert channel. We demonstrate the validity of the proposed attack using the legally questionable *Liberator* gun files in Section 6. Next, in Section 7, we offer further discussion, such as the implications of the proposed attack on the AM Security field as a whole, three different contexts of attack distinguishability, and the robustness of the introduced channel against sanitation efforts. We conclude, in Section 8, with a brief review and summary of the major results of this paper.

2 ATTACK CHARACTERISTICS

In this section, we first discuss steganography and subliminal channels, then consider use cases for our attacks, and then cover the attack mechanisms.

2.1 Steganography and Subliminal Channels

Steganography is a term of art associated with techniques to hide the presence of a secret message. With its recorded origins dating back to 440BC [48], steganography has had numerous implementations. In the classical, pre-computer age, examples include the insertion of secret messages in a painting (e.g., in the form of water ripples representing Morse code, or in a handwritten letter, or as the *N*th character of each line or sentence).

In the computer age, despite the rise and pervasive use of cryptography, steganography has found a variety of new forms [48, 51]. The theoretical possibilities of steganography in digital documents has been extensively studied in the literature [4, 6, 19, 30, 60]. If an underlying channel contains a source of *entropy*, i.e., redundant information that can be modified without disrupting the usability of the digital data, it can be used for embedding a secret message. A classical example is the encoding of an arbitrary bit stream in a .bmp file, where each pixel can be represented through its three RGB (red, green, blue) components. To encode individual bits of the stream, the least significant bits of the RGB values can be modified. As the resulting distortions of color are not perceptible by a human eye, the encoded data has the potential to remain undetected. Steganography can be found in numerous practical applications, such as digital media watermarking [13, 17, 26, 40] or in circumventing censorship [21, 33, 62, 63].

Closely related to steganography are *subliminal* (and *covert channels*). Here, the idea is to "piggyback" on the legitimate or legitimate-looking communication channel between two parties and embed in it an additional secret message. For example, a .bmp file could be sent that contains a secret message that was embedded using the approach outlined above.

In addition to the possibility of message encoding is the question of its *distinguishability*. Ideally, the distribution of the encoded message is close to the distribution of a normal message, making it impossible to determine whether a hidden message is present or not.

For a more formal treatment of steganography and information hiding in media channels see [10, 36]. For kleptographic attacks and exploitation of channels combined with cryptographic tools see [69–71].

2.2 Use Cases for Considered Attacks

We consider attacks that attempt unauthorized transmission over hidden channels, as well as concealment of the associated stolen or illegal data. While this behavior can be exhibited by all kinds of attacks, a certain degree of concealment is common to real-world *targeted attacks* and *attack campaigns* [57]. Balancing between risks and rewards [54], exploiting existing communications channels like the transfer of design files would allow criminals to significantly reduce both the risk of and investment in the attack. We offer these for a few realistic examples.

Information Stealing/Leaking: Malicious insiders remain one of the biggest security threats for both public and private organizations [29, 59]. They often have access to classified, proprietary, or otherwise sensitive information that can be of great value to external actors, and can themselves initiate an attack [8]. Since internet access is ubiquitous, malicious actors might be tempted to use it to *exfiltrate*[2] stolen information. However, external network connections are routinely monitored for suspicious activities and

[1] The attack is summarized in a short YouTube video: https://www.youtube.com/watch?v=zUnSpT6jSys

[2] Henceforth in this paper we use term *exfiltration* to indicate an unauthorised data transfer. It can be conducted either by a malicious actor or by malware. The same technique can be used to transfer data out of or into a protected environment. To clearly distinguish the latter case, we will use term *infiltration*.

outgoing communications containing encrypted content can be explicitly blocked. In this setting, the question faced by the malicious insider becomes whether the stolen data can be exfiltrated through legitimate-appearing internet activities.

A similar scenario applies to malware, such as spyware Trojans that steal information and transmit it to external parties automatically. If such activities are discovered, they can be blocked and investigated, eventually leading to the discovery and removal of the malware. To prevent or delay detection, communication has to appear legitimate and blend in with other network traffic.

Downloading Malicious Content: To obtain access to the desired information, malicious insiders might need specialized tools, such as network sniffers or password crackers. However, potentially malicious software is routinely recognized and blocked by firewalls. This raises a reverse challenge to the malicious insider: how to infiltrate the necessary malware while disguising the traffic with a legitimate purpose.

Malware can be confronted with similar obstacles. In the case of droppers, malware whose sole purpose is to install additional malware, this functionality depends on the ability to download additional malware from the internet while bypassing firewall checks.

Enabling Reverse Shell to a Command & Control (C&C) Server: Among the malware that a malicious insider or dropper can install on a corporate computer is a reverse shell: malware that "calls out" to an external *Command & Control* (C&C) Server and provides it with remote shell access to the compromised computer. It is common to use reverse shells to browse through files, download and upload files, and execute arbitrary commands including the installation of additional programs. On the C&C Server side, either a malicious actor or automated malware exercises control and communicates commands. Obviously, this communication, including commands, console outputs, and file transfers, needs to avoid detection by firewalls or other sophisticated intrusion detection and prevention systems (IDS/IPS).

Storing Stolen/Illegal Data on a Computer: Since sensitive information might be accessible on an agency or corporate network, activities like storing such information on a computer connected to the Internet might be tightly monitored. This poses a challenge for malicious actors, such as those connected to a compromised computer via a reverse shell, who need to temporarily store the stolen data prior to its leaking without raising red flags.

Eventually, malicious actors on the outside would also face a similar challenge of hiding the presence of stolen and illegally possessed data. In order to avoid prosecution if incriminating evidence should be discovered, they might want to conceal the illegally obtained data from digital investigation efforts. This applies also to data that is outright illegal in a country, such as the designs of 3D-printed guns, the production of which has already led to a jail sentence in Japan [22].

2.3 Attack Mechanisms

Our Steganographic/info-hiding attack stores or sends data in a disguised manner so the fact of sending the data is, to an extent, concealed. In our attacks, the payload can be any information, as in the examples above, to be sent or stored secretly. The "carriers" (stego-containers) which we examine are STL (*STereoLithography*) files. The carrier will be put in a channel to be communicated (or stored). The channels considered are between network elements which are part of the additive manufacturing ecosystem. The overall "Stego-system", which is the methods and means used to create the concealed channel for communicating information, will be a software system that performs the embedding in and extraction from the carrier and the communication/storage. It is assumed that the system is either run by the adversary, or runs in an undetected way as a Trojan in a compromised hosting environment.

The actual attack will consist of identifying redundant data in the STL files, and methods which embed secret information in them for the above attacks. Such attacks are known as "Payload Embedding." Based on the degree a channel is resistant against discovery and extraction of the transported secret message, we distinguish between the following three categories:

- **Weak Attack: Plaintext Steganographic Channel.** The simplest attack is to embed the information directly, as is, and rely on the fact that as long as the carrier STL files produce the expected result (i.e., the proper 3D object is printed without distortions) no investigation will be started that could reveal the presence of the covert channel. We will present our proposal for the weak attack in Section 4.
  Claim 1: If one relies on the AM process to be successful as an indication of non existence of a hidden channels, one will fail.
- **Regular Attack: Encrypted File over a Steganographic Covert Channel.** The weak attack implicitly assumes that the embedded information is "plaintext." A commonly used improvement is to embed and encrypted file, such as a password-protected zip archive. We will use this approach when we demonstrate the attack in Section 6
  Claim 2: Assuming the cryptosystem is strong: If the regular attack is used and the cryptographic key is not found in the malware, even knowing the attack, one cannot find after the fact what information was embedded.
- **Strong Attack: Fully Encrypted Steganographic Covert Channel.** In the regular encryption attack, if one analyzes the malware one may find the key, which is a weakness of this attack. A strong attack would implement a fully cryptographically encrypted communication channel that is transported over the steganographic covert channel, where analyzing the malware will not help in recovery of the file stolen. We propose to use a KEM/DEM[3] approach. While the principles of this approach are well established, this approach needs to be adjusted based on the characteristic aspects of the underlying steganographic channel. In Section 5, we derive our approach for KEM and DEM from the ransomware/ kleptographic type of attack [68, 70], and show which adjustments need to be made to ensure that the actual payload consisting of the *{KEM||DEM}* pair remains indistinguishable from a random string of bits after the attack, or:

[3]KEM - Key Encapsulation Mechanism. DEM - Data Encapsulation Mechanism. DEM is used to encrypt the payload message with a symmetric session key, while KEM encrypts the used session key with the public key of the message recipient. The results of both are concatenated and constitute the message that is embedded.

Claim 3: Assuming the KEM and DEM generated are cryptographically strong: With access to the output file, the receiver public key, and full access to the malware, one cannot guess after the fact what the information embedded in the file is.

- **Note: Fighting Traffic Analysis.** Given that we can either encrypt a file in our channel or embed an utterly random string instead, the cryptographic attacks when the key is not known can be done in a way that the intended recipient receives encrypted data while the rest get a file with random information.
  Claim 4: In a cryptographic attack where files sent to intended recipients embed an encrypted data, and other files embed properly randomized information, one can view the traffic but cannot tell who is the actual recipient.

The two crypto transformations of the basic weak channel are generic strengthening of the subliminal channel attack. As has been demonstrated with other attacks, such as in ransomware or doxware attacks [37, 72], detecting the above mentioned attacks poses difficulties to end-users. The systematic methods required for dealing with them (combining early detection and elimination of malware, recovery, backups, etc.) are out of scope for this paper, but our point is that they need to be addressed in the area of AM.

3 FURTHER BACKGROUND

We discuss the STL file format and other related work here.

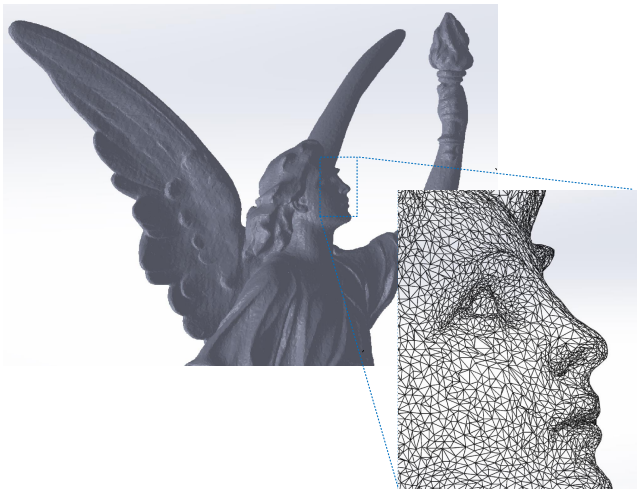
3.1 STereoLithography (STL) File Format

STereoLithography (STL) files describe a 3D body by specifying a set of triangles that enclose its surface. They are usually generated by a computer-aided design (CAD) program, but can also be produced by scanning real-world objects. STL can be in either ASCII or binary formats, which are semantically equivalent in the AM process. The binary format significantly reduces file size, while the ASCII format is human-readable and can be easily edited or inspected for errors manually.

Figure 1 shows an example of a 3D model represented as a STL file. Figure 1a depicts the model of the 3D object itself, as well as a zoom-in excerpt showing its composition from adjacent triangular surfaces, known as *facets*. Figure 1b is an excerpt from the ASCII STL file that describes this model; this excerpt shows two of the object's facets, each of which is defined by three vertices with their respective x, y, and z coordinates.

In the STL file format, the description of 3D objects is defined as a series of nested blocks. The outermost block describes a single solid body and is enclosed by the tags *solid* and *endsolid*. Both tags can be followed with additional comment text, which is commonly used to specify the model name or author.

Each solid 3D object is described as a series of triangular surfaces defined within the *solid* block. The description of each *facet* is enclosed within *facet* and *endfacet* tags. Each facet is defined by two distinct elements: its three vertices define the boundaries of the facet, and the normal vector defines the outwards orientation of the surface. The latter is indicated by the *normal* element, which provides the vector's x, y, and z coordinates. The STL format defines a relationship between the normal vector and the vertices describing the facet: The normal vector should follow the "right hand" rule,



(a) Solid and Wired (Triangular Facets) Representations

```
solid StanfordLucy
  facet normal -0.1128 -0.818 -0.5641
    outer loop
      vertex -13.101 0.527998 52.206
      vertex -13.035 0.791999 51.81
      vertex -12.771 0.527998 52.14
    endloop
  endfacet
  facet normal -0.0573 0.774 0.6306
    outer loop
      vertex 5.906999 7.589998 50.886
      vertex 5.972999 7.325997 51.216
      vertex 6.236998 7.722    50.754
    endloop
  endfacet
       ...
endsolid StanfordLucy
```

(b) ASCII STL File (Excerpt: Two Triangular Facets Shown)

Figure 1: 3D Digital Design Model—Stanford Lucy [1]

meaning that the vertices are listed in counter-clockwise order when viewed from the normal.

The vertices of a facet are defined in the innermost nested block, which is enclosed by the tags *outer loop* and *endloop*. Each vertex is specified as *vertex* v_x v_y v_z. The three v's are single-precision floating point values that represent the x, y, and z coordinates, respectively.

For both facets and vertices, there is some freedom in their ordering within blocks. Facets can be freely reordered within the *solid*; there is no specification for them. Vertex orderings are required to form a "right-hand" rule with the normal vector, but there are multiple valid orders for any facet. This freedom for semantically equivalent representations in STL format is what allows us to use it as a carrier.

3.2 Related Work

Several authors survey the entire field of AM Security [43, 50, 66]. So far, three security threat categories have been identified for AM:

theft of technical data, sabotage attacks, and illegal part manufacturing [66]. While these threats are not unique to AM, it has been shown that in the AM context numerous characteristic aspects exist that require special consideration [24, 65]. To the best of our knowledge, the security threat discussed in this paper has not been identified in the research literature before.

Kuznetsov et al. [39] recognized that 3D Printing technology can be used to steganographically hide information. However, as opposed to the attack settings considered in this paper, the authors propose to hide information inside of a 3D-printed object. The authors encode a secret message as a three-dimensional matrix "embedded" within the boundaries of the actual 3D-printed object. While this approach can be used in certain attack settings, it violates the constraint considered in this paper – avoiding modifications of the 3D model. When the original 3D geometry is known, such insertions in the model can be detected with non-destructive testing. Most importantly, in the case of functional parts, such modifications will likely impact the part's mechanical characteristics, as has been demonstrated by several publications focusing on intentional sabotage attacks [7, 56, 73].

The remaining related literature focuses on addressing the threat of Intellectual Property (IP) violation. Numerous works suggested watermarking techniques that could be used for copyright protection, theft deterrence, and inventory. Wang et al. [61] summarized 3D mesh watermarking techniques and their attack robustness in their survey. They categorized techniques as *fragile*, for authentication and integrity applications, or *robust*, for copyright protection. The authors identified the two major categories of attacks as *geometric* (modifying vertex positions) or *connectivity* (modifying vertices, edges, and facets adjacency relations). In an AM watermark integration survey, Macq et al. [42] assessed watermarking as a traceability mechanism to be used against already-printed models. They focused on each method's resistance to alteration (*robustness*) and visual imperceptibility (*fidelity*). The authors concluded that, while attainable, no method currently provided a satisfactory watermarking solution.

Ohbuchi et al. [45, 46] presented algorithms for embedding data into 3D models using vertex coordinates and topology. One approach, the *Triangle Strip Peeling Symbol (TSPS)* sequence, embedded a sequence of binary digits by constructing a strip using triangle edges and adjacency states [45]. Flaws in their method included reaching mesh boundary limits, circling back, and erroneous extraction. Kanai et al. [35] suggested embedding watermarks in models using wavelet transforms. The embedded watermarks were perceptually invisible and invariant to affine transformations; however, the algorithms produced geometric errors and were restricted to a specific topological class of mesh. Also in the frequency domain, Praun et al. [49] proposed a method to determine the probability that two models were created independently through watermark comparisons. However, the method resulted in vertex modifications which impacts geometry. Ohbuchi et al. [47] integrated a watermark into 3D mesh models using the spectral domain. Using vertex connectivity and coordinates, the authors embedded a binary bitstream. Their method perceptibly affected the object. Cayre and Macq [11] introduced a substitutive, blind scheme in the spatial domain. The process introduced modifications of vertex positions. Based on experiments with 11 different models, the authors reported distortion rates ranging from 0.15 to 0.25%. Hou et al. [31] specifically considered 3D mesh watermarking with regards to 3D printing. The authors used the layers created by the STL 2D cross-sections to design a rotating disk embedding system where the relative rotation contains the payload. Their method required knowledge of the original model, and, while visually imperceptible, resulted in geometric distortion. Hou et al. [32] proposed estimating the printing direction from the printed layer artifact as a way to synchronize and recover a cylindrical coordinate system watermark. Although their solution was blind, among the listed limitations was that the printing direction and z-axis must be aligned and fixed without any model rotation after watermarking.

Recent efforts in addressing AM IP protection have focused on modifying the printed object. For example, Delmotte et al. [20] modified the layer thickness in multiple locations to embed a watermark, an approach which resulted in object deformation. Silapasuphakornwong et al. [55] modified the object material, using a two nozzle print system with iron added to the material in the second nozzle. While this produced a re-writable watermark, it requires adding a magnetic material to the object. Chen et al. [14, 16] introduced an approach to imprint a QR code in the 3D printed part. Individual segments of QR code can be distributed across multiple layers and produced either with voids or support material substituting for source material; a CT scanner can be used to read out such a code. In all these approaches, the proposed modifications might not be acceptable for functional parts.

Additionally, patents have been filed for STL model watermarking [41, 74], digital rights management of modeling data using passwords with barcodes [44], model protection via machine instruction masking [58], and a 3D printing environment watermark embedding and detecting apparatus [27]. Recently, Treatstock has begun offering *Watermark3D*, an online watermarking solution for STL files [3, 53]. Under their patent pending system [53], a user uploads an STL file, provides information to be used for a watermark, chooses whether to make the watermark public, and then downloads a watermarked file [3].

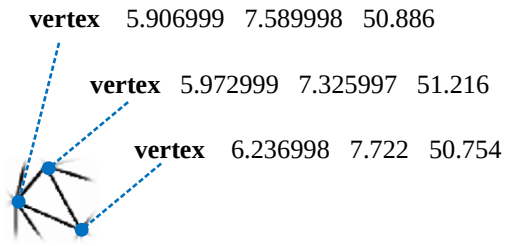
4 WEAK ATTACK: PLAINTEXT STEGANOGRAPHIC COVERT CHANNEL

Our new approach for establishing a steganographic covert channel in STL files consists of three parts: (i) a transformation primitive that can be used to encode/decode individual bits of information, (ii) a raw steganographic channel that consists of functions to encode and decode individual bytes bit-by-bit while advancing internal position in the carrier STL file, and (iii) an approach building upon these functions to encode an arbitrary binary file in a carrier STL file.

4.1 Encoding/Decoding Individual Bits

We identified several sources of entropy in the STL file format[4], each of which can be used to encode individual bits of information. These are: the position of the facet descriptions in the file, compliance of the normal vector to the right hand rule, and the order in which vertices are listed within the facet definitions; additionally, number

[4]Due to the simplicity of the format, this was done manually rather than with an automated tool.



(a) 3D Facet Defined by its Three Vertices

```
facet normal -0.0573 0.774 0.6306
  outer loop
    vertex 5.906999 7.589998 50.886
    vertex 5.972999 7.325997 51.216
    vertex 6.236998 7.722    50.754
  endloop
endfacet
```

```
facet normal -0.0573 0.774 0.6306
  outer loop
    vertex 5.972999 7.325997 51.216
    vertex 6.236998 7.722    50.754
    vertex 5.906999 7.589998 50.886
  endloop
endfacet
```

(b) Two Options Representing Bit Value 0

```
facet normal -0.0573 0.774 0.6306
  outer loop
    vertex 6.236998 7.722    50.754
    vertex 5.906999 7.589998 50.886
    vertex 5.972999 7.325997 51.216
  endloop
endfacet
```

(c) Represents Bit Value 1

Figure 2: Steganographic Encoding of a Single Bit in a Facet

representations and spaces can be used with the ASCII-style STL file format. We present a brief discussion of alternative bit encoding approaches and their drawbacks in Appendix A.1.

After considering these options, we elected to encode individual bits of information via the order in which vertices are specified within a facet. This choice is motivated by several factors. First, we think that this approach renders the steganographically encoded information least likely to be detected either manually or by an automatic distinguisher. Second, using the approach described below, a recipient can decode the transmitted information "blindly," i.e., without having access to the original STL file. Lastly, the proposed bit encoding primitive should neither interfere with the printability of nor introduce any distortions in the 3D geometry of the object specified in the STL file.

The proposed bit encoding approach is based on the following observation. Each facet is uniquely defined and identified by its three vertices, which we can refer to as v_1, v_2, v_3 (see Figure 2a). In a well-formed STL file, the order of these vertices should follow the *right hand* rule with the normal vector (see Section 3). However, beyond this, the order in which the vertices should be specified is neither defined nor impacts the triangular facet they describe. This means that the sequence in which the three vertices are listed can be used to encode bits of information. The correlation with the normal vector restricts the valid sequences of listed vertices to the cyclical rotation of their order, i.e., (v_1, v_2, v_3), (v_2, v_3, v_1), and (v_3, v_1, v_2).

If all three vertices are distinct, i.e., no two vertices have the same x, y, z coordinates, up to three values (0, 1, and 2) can be encoded per facet. This, however, would require that the bit stream is converted to base 3 (instead of base 2 for the binary value representation). While the obvious advantage would be the increase of encoding capacity to up to $\lfloor \log_2(3^{F_N}) \rfloor$ binary bits of information in a STL file with F_N facets, this will increase the complexity.

Alternatively, if only a single binary value is encoded using vertices rotation, the encoding capacity is limited to F_N bits. While decreasing capacity, this approach eliminates base 2 to 3 conversion. In the remainder of this paper, we will proceed with the encoding of a single bit of information per facet.

In theory, all the vertices should be distinct, with different x, y, z coordinates. This is not guaranteed in real STL files, which often need to be "repaired" before 3D printing. To account for this situation, we will skip facets in which all three vertices have exactly the same coordinates. For the remaining facets, we elect to encode a 0 bit value by starting with the vertex whose combined coordinate value is minimal, and 1 otherwise. For the example facet in Figure 2a, steganographic representations of bit values 0 and 1 are depicted in Figures 2b and 2c, respectively).

More formally, for any facet with at least two distinct vertices we define the encoded bit value as follows:

$$\textit{Encoded Bit} = \begin{cases} 1, & \text{if } v_1 = max(v_1, max(v_2, v_3)) \\ 0, & \text{otherwise.} \end{cases}$$

The comparison between two arbitrary vertices v_i and v_j can be defined as a successive comparison of their respective x, y, z coordinates, treating these as components of a larger number.

The *max* function can be implemented as follows:

```
max (v1, v2)
{
  if (v1.x < v2.x) return v2;
  if (v1.y < v2.y) return v2;
  if (v1.z < v2.z) return v2;

  return v1;
}
```

Please note that, even for the selected vertex order, alternative encoding strategies exist.

Figure 3 shows pseudo-code for encoding and decoding a single bit of information using the proposed approach. In both functions, the parameter *facet* is a reference to a facet in which a bit should be encoded or from which it should be decoded. Both functions rely upon the function *max* that compares two vertices based on their combined x-y-z values. In the *EncodeBit* function, if the bit value 1 should be encoded and the first vertex already has the greater value, no additional action is required. Otherwise, the order in which vertices are described in the facet should be cyclically rotated either left (if v_2 is the maximum value) or right (if v_3 is the maximum). The encoding of bit value 0 is similar: a rotation is needed only if the value of the first vertex is maximum. In the pseudo-code, we

```
EncodeBit (facet, bitValue)
{
  if (bitValue == 1)
  {
  if (facet.v1 == max(facet.v1,
                      max(facet.v2, facet.v3)))
    return;

  else if (facet.v2 == max(facet.v1,
                           max(facet.v2, facet.v3)))
    // Rotate Left
    facet.v1, facet.v2, facet.v3 =
    facet.v2, facet.v3, facet.v1;

  else
    // Rotate Right
    facet.v1, facet.v2, facet.v3 =
    facet.v3, facet.v1, facet.v2;
  }
  else
  {
  if (facet.v1 == max(facet.v1,
                      max(facet.v2, facet.v3)))
    // Rotate Left
    facet.v1, facet.v2, facet.v3 =
    facet.v2, facet.v3, facet.v1;
  }
}
```

```
DecodeBit (facet)
{
  if (facet.v1 == max(facet.v1,
                      max(facet.v2, facet.v3)))
    return 1;

  return 0;
}
```

Figure 3: Encoding/Decoding of a Single Bit in/from a Facet

rotate left. The *DecodeBit* function checks whether the value of the first vertex described in the facet is the greatest of all three. If so, the function recognizes it as bit value 1, which is returned to the caller; otherwise, it returns 0.

4.2 Raw Steganographic Channel: Encoding/ Decoding Bytes with File Position Update

The bit encoding/decoding primitives described in Section 4.1 can be used as a building block for the encoding/decoding of bytes. Encoding individual bits in a byte can be done in an arbitrary order as long as it is "mirrored" by the decoding routine. The two simplest bit encoding orders would be to start with the most significant bit (MSB) and proceed bit by bit to the least significant bit (LSB), or similarly from LSB to MSB.

Pseudo-code for the encoding and decoding of individual bytes is shown in Figure 4. The most notable feature is the use of the *GetNextFacet* function which, in addition to returning a reference to the facet, advances an internal pointer in the STL file to the next facet. Both functions also use a *mask* byte to either determine which bit value should be stored in the facet (for the *EncodeByte* function) or to reflect the decoded bit value in the return byte (for the *DecodeByte* function).

The encoding/decoding of an individual byte can in turn provide a building block for the encoding/decoding of other data types, such as double word integers or binary byte arrays. The latter can be used to encode a raw byte stream, an approach that can be used in a strong attack with a block cipher over the proposed steganographic STL channel. Similarly, the bit encoding/decoding combined with

```
EncodeByte (byteValue)
{
  bitMask = 0x80;
  for (i=0; i<8; i++)
  {
    facet = GetNextFacet();
    if (byteValue & bitMask)
      EncodeBit (facet, 1);
    else
      EncodeBit (facet, 0);

    bitMask = bitMask >> 1;
  }
}
```

```
DecodeByte ()
{
  byteValue = 0x00;

  bitMask = 0x80;
  for (i=0; i<8; i++)
  {
    facet = GetNextFacet();
    if (DecodeBit(facet) == 1)
      byteValue = byteValue | bitMask;

    bitMask = bitMask >> 1;
  }

  return byteValue;
}
```

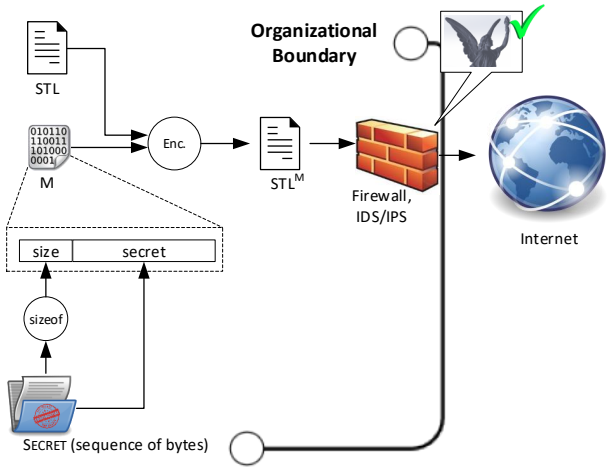
Figure 4: Encoding/Decoding of a Single Byte (MSB to LSB)

the advancing facet pointer can be used in a strong attack using a stream cipher over the steganographic channel. Below we outline how the functions of a raw steganographic channel can be used to encode a single binary file in a carrier STL file.

4.3 En-/Decoding Arbitrary Binary Files

Our proposal for an attack using an STL file to encode an arbitrary sequence of bytes (such as a binary file) is depicted in Figure 5. The binary file is the secret message that needs to be transmitted. To facilitate the encoding and decoding of variable-length binary files, the size has to be included in the encoded message. Therefore, the encoded message M consists of a fixed-length *size* field (specifying a number of bytes) and a variable length *secret* field (see Figure 5a). We use a *size* field 4 bytes long, supporting the encoding of up to 4GB long *secret* byte streams. The result of encoding a message M in an innocent-seeming carrier STL file is STL^M, which contains both the original 3D model and the steganographically encoded message. When sent to an external source over the Internet (e.g., as a part of the attack setting described in Section 2), this approach aims to fool firewall and IDS/IPS systems, which should let it through without raising a red flag.

Figure 5b depicts pseudo-code for an *EncodeFileInSTL* function that implements the encoding part of the approach. The encoding function takes three parameters: a carrier STL file name, *fnSTL*, a binary file name, *fnSecret*, and a destination STL file name, *fnSTLdest*. The carrier STL file is the original *STL* file that will be encoded with the secret message. The binary file contains the secret message to be encoded. The destination STL file will contain the combined carrier file and binary file. To start encoding, the entire mesh of the carrier STL file is loaded. Next, the entire secret binary file is read, obtaining a sequence of bytes as well as the total number of bytes in the file. At this stage, the carrier STL file is evaluated for

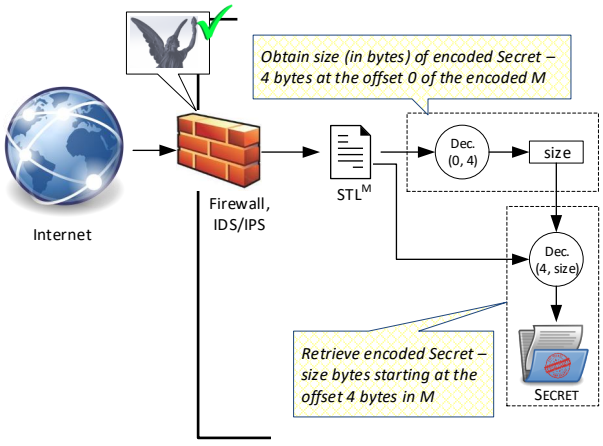


(a) Encoding Approach

```
EncodeFileInSTL (fnSTL, fnSecret, fnSTLdest)
{
  secretBytes = ReadFileBytes(fnSecret);
  secretSize  = sizeof(secretBytes);

  carrierSTL  = LoadSTL(fnSTL);
  stlCapacity = FacetCount(carrierSTL)/8;

  if (stlCapacity >= secretSize+4)
  {
  STLSeek2Facet(SEEK_SET, 0);

  EncodeLong(carrierSTL, secretSize);
  EncodeBytes(carrierSTL, secretBytes, secretSize);

  SaveSTL(carrierSTL, fnSTLdest);
  }
}
```

(b) Encoding Function, Pseudo-Code

Figure 5: Encoding Single Binary File into an STL File



(a) Decoding Approach

```
DecodeFileFromSTL (fnSTL_M, fnSecret)
{
  carrierSTL = LoadSTL(fnSTL_M)

  STLSeek2Facet(SEEK_SET, 0);
  secretSize  = DecodeLong(carrierSTL);
  secretBytes = DecodeBytes(carrierSTL, secretSize);

  SaveInFile(fnSecret, secretBytes, secretSize);
}
```

(b) Decoding Function, Pseudo-Code

Figure 6: Decoding Single Binary File from an STL File

sufficient capacity to encode the secret message together with the size field. If the capacity is sufficient, the encoding is conducted on the loaded STL mesh. The call to function *STLSeek2Facet* positions the seek pointer on the first facet in the carrier STL file. Four bytes indicating the secret size are encoded at this position, offset 0. The secret is then encoded starting at an offset of 4 encoded bytes. In the pseudo-code, all encoding functions are assumed to advance the facet pointer, similar to the common behavior of file write functions. After the encoding is completed, the mesh is saved to the destination file indicated by *fnSTLdest*.

Conceptually, before decoding can start, the STL^M file received from the Internet might also need to go through firewall and IDS/IPS checks (see Figure 6a). The decoding of the message is also a two stage process. It starts with the decoding of the first four bytes from STL^M at offset 0, which represent the *size* field. Then the *size* bytes of *secret* are decoded, starting at an offset of 4 encoded bytes in the encoded message. All decoding operations are conducted byte by byte and bit by bit, mirroring the encoding procedure.

The decoding function *DecodeFileFromSTL* closely follows the outlined process (see 6b). It takes two parameters: an STL file name that contains encoded secret information, *fnSTL_M*, and a name for a binary file in which the decoded secret will be saved, *fnSecret*. To start decoding, the mesh of the STL_M file is loaded. Then, at offset 0, the four bytes indicating secret size are decoded. At an offset of 4 encoded bytes, the remaining bytes are read and decoded to extract the secret. Finally, the extracted byte stream is saved in the binary file indicated by *fnSecret*.

5 STRONG ATTACK: FULLY-ENCRYPTED STEGANOGRAPHIC COVERT CHANNEL

A strong attack can use the raw steganographic covert channel defined in Section 4 as a carrier, similar to how higher level network protocols are transported over the underlying network layers. For a strong attack, several more factors have to be taken into account.

First, classical bi-directional key exchange protocols might not be possible, for example because transfer of STL files (used as a carrier) may be permitted only in one direction. Furthermore, bi-directional communication, implemented via consecutive sending and receiving of STL files, might be identified as a suspicious activity and lead to the discovery of the communication channel. Thus the communication channel should be unidirectional.

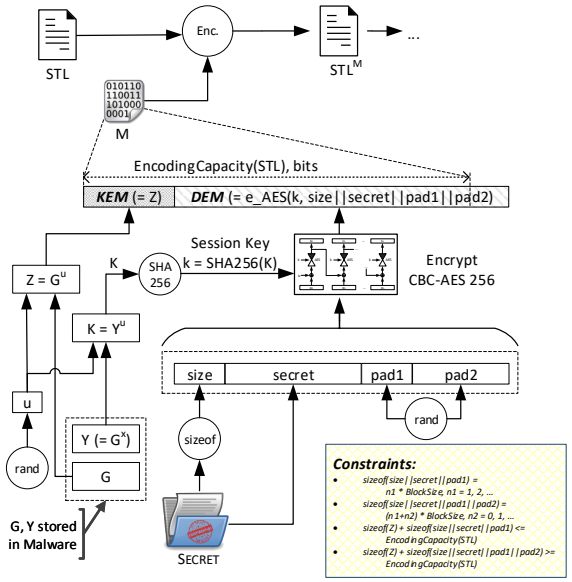


Figure 7: Strong Attack, Encoding Approach (Padding: *pad1* is used to increase length of *size*||*secret* to a multiple of a block size in used symmetric encryption (128 bits for AES); *pad2* increases the length of DEM, so that together with KEM it overwrites all bits in the carrier STL file)

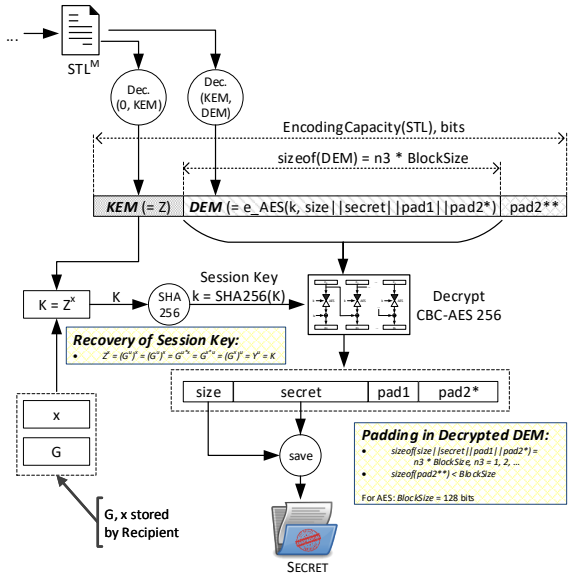


Figure 8: Strong Attack, Decoding Approach (Padding: *pad2 is a multiple of block size (128 bits for AES), and corresponds to the part of *pad2* that could be encoded in the carrier STL file)**

Second, while encryption using a symmetric key stored in malware is possible, the detection and reverse engineering of the malware would recover this key. This will degrade the cryptographic channel to the level of the weak attack, and eventually allow identification of secretly transmitted data if transmitted STL files are logged in their entirety.

To address these concerns, we argue that the strong attack should build upon public key technology, and the KEM/DEM approach in particular. In this approach, for every new transmitted message a new session key is generated. In the Data Encapsulation Mechanism (DEM), this key is used to encrypt the exfiltrated message with a state-of-the-art symmetric encryption algorithm, such as AES (Advanced Encryption Standard) based method. In general, the encoded payload will be larger than the block size of AES (which is 128 bits). Therefore, modes of operation, such as a variant of Cipher Block Chaining (CBC), have to be used, or other modes like GCM. A Key Encapsulation Mechanism (KEM) generates information that a recipient can use to recover the session key, e.g., by encrypting it with a public key (stored in malware) of the recipient. A state-of-the-art asymmetric encryption algorithm, such as Elliptic Curve Cryptography (ECC), e.g., EC-DH, can be used. The results of both KEM and DEM are concatenated and together (i.e., KEM||DEM) constitute the transmitted message. The local malware erases both his random choice for the session and the keys derived from it.

The selection of a KEM/DEM approach has an immediate impact on the size of the secret message that can be transported. As the STL files describing different models have different amounts of vertices, so too will their encoding capacity vary. This encoding capacity should be sufficient to incorporate the entire KEM||DEM outcome. The size of KEM depends on the selected approach, but for the purpose of discussion we can consider it fixed. Similar to the discussion of the weak attack (see Section 4), the message processed by DEM should consist of at least the fields *size* and *secret*, so that the recipient can extract the secret. The DEM can be based on stream cipher or a block cipher mode and may include an authentication field as well.

A carrier STL suitable to transport the message will generally have encoding capacity exceeding the combined KEM||DEM size. This raises the concern of distinguishability of the encoded KEM/-DEM from the rest of the file. Should the distinction be possible, it could act as an indicator of the covert message's presence and simplify cipher extraction for the analysis. To prevent this from happening, all remaining bits in the carrier STL should be encoded with an extended DEM part. This can be achieved by padding the input of the DEM encryption (*size*||*secret* padded by *pad1* to a multiple of the block size) with additional padding *pad2* of the length sufficient to extend the encrypted random blocks till the end of the file; in general, not all bits of the final cipher block will be encoded.

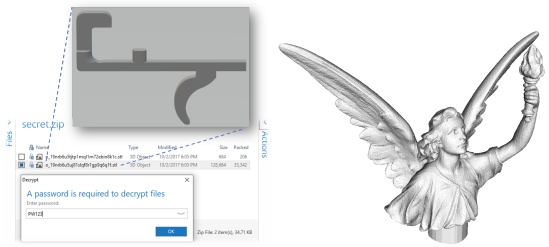
Our proposal for KEM and DEM are as follows. The KEM can be realized using either off-the-shelf implementation of state of the art asymmetric encryption, such as Elliptic Curve Cryptography (ECC), or with custom-made derivatives. ECC is using an elliptic curve as a generator of a group, and exponentiation as an operation on the group elements known as "multiplication along the

curve." The group generator and the operations on the group are shared by both the attacker (on the receiving end of the message) and the malware (that sends out the message). Given that G is a generator in the right group, an attacker can generate a public key as $Y=G^x$, where x is only known to the attacker and Y can be integrated in the malware. At the time when malware needs to send a secret message, it randomly chooses a value u and uses it to compute $Z=G^u$ and $K=Y^u$. The value Z is embedded in the channel as a result of the KEM mechanism. The attacker who receives the message can use it to recover the value of K as follows: $Z^x=(G^u)^x=G^{u\cdot x}=G^{x\cdot u}=(G^x)^u=Y^u=K$.

In the DEM, a session key k for the data encryption can be derived by what is called KDF (Key Derivation Function) which is based on cryptographic hash functions K: $k=KDF(K)$. The result is a (say, 256 bit random looking) value that can be used as keys by encrypting the payload with AES in some mode: $cipher=e^{AES}(k, message)$. As K can be recovered by the recipient of the message, so can the session key k, and the message can be decrypted. After the message is encoded in the carrier STL file, the malware erases K and k, preventing recovery of the sent message at the compromised site, even if the malware is discovered and reverse-engineered. This attack follows the malicious cryptographic approach of Adam Young and Moti Yung [70, 72]. To get the key one has to break the Decisional Diffie-Hellman assumption.

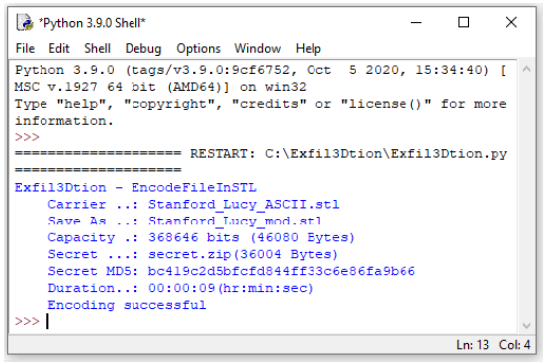
Encoding and decoding in this way are summarized in Figures 7 and 8, respectively, with some concrete functions choices. Encoding a message in an STL file can be easily implemented with the functions defined in Section 4; that pseudo-code is omitted here. In Figures 7, *pad1* expands *size*||*secret* to a multiple of the block size in the symmetric encryption algorithm, while *pad2* does the same to ensure that the cipher overrides all bits in the carrier file; both pads are generated randomly. In Figure 8, the encoded portion of encrypted *pad2* is indicated by two fields *pad2** and *pad2***. The first indicates the portion of encrypted *pad2* that resulted in the cipher blocks encoded in their entirety; the second is the very last cipher block generated over *pad2* which could be only partially encoded in the carrier STL file. This is just a demonstration, and we can employ other DEM methods (e.g. GCM which takes care of encryption and authentication). The decoding approach presented in Figure 8 simplifies its visualization, i.e., the decryption is conducted over *size*||*secret*||*pad1*||*pad2**. The decoding performance can be optimized by first extracting and decrypting the very first cipher block of DEM and then, based on the value of *secret*, determine how many further blocks should be decoded and decrypted; this will reduce the operation to *size*||*secret*||*pad1*.

Lastly, while the description above defines a fully-encrypted steganographic channel over STL files, one concern remains - the ability of a defender to distinguish between STL files containing secret messages and those that don't. Even without the ability to decrypt the embedded information, defenders might identify an attacker's e-mail or IP address (this is akin to "traffic analysis" methods). To establish indistinguishability, automated malware can generate a random bit sequence r composed of a group element and a random string, and embed a new version in each STL files on the compromised system using the above described method. This way, all STL files sent from the compromised system to both



(a) Secret: Selected *Liberator* Design Files in a Password-Protected ZIP File

(b) *Stanford Lucy* used as a "Carrier" STL File



(c) Exfil3Dtion - Encoding, Console Output

Figure 9: Encoding Secret in STL Carrier File, before Sending to a Recipient over Internet or Covertly Storing on PC

attacker and to innocent recipients will be indistinguishable, as they all contain the secret message look-alike part. Distinguishing dummy-cipher from cipher with any advantage higher than $1/2+\epsilon$ for non-negligible ϵ means that the *Decisional Diffie-Hellman problem* DDH in the group is violated or the symmetric encryption semantic security is violated (i.e., the encryption is distinguishable from a random string, violating its Pseudo-random function property). After the embedding when the keys are erased, the address of the intended receiver is erased as well. Namely, in compromised systems operating this way, we cannot tell to where stolen information is being sent (the thief is hidden in the crowd).

The above can be described in a formal proof essentially following the security proof of the original KEM/DEM hybrid encryption system.

6 EXPERIMENTAL EVALUATION: HIDING *LIBERATOR* GUN DESIGN FILES

We implemented the described weak attack approach for embedding and extracting a single binary file as a Python script, calling the program *exfil3Dtion*. To demonstrate the feasibility of our method with a real-life application, we simulated a malicious actor scenario in which "restricted" files were encrypted, password protected, and hidden inside a benign STL file. As our restricted content we selected a few *Liberator* gun [28] design files.

The *Liberator* gun is a 3D printed handgun whose manufacturing and design file distribution has been controversial [9, 12]. Although file distribution by the designer is currently blocked, the files were previously available online and were downloaded over 100,000 times in a two-day period [25]. Given the popularity and continued online presence of the files [52], it is reasonable to assume that interested parties might attempt clandestinely distributing the files using the exfiltration approach presented in Section 4.

To begin, we selected two of the Liberator design files and compressed them into a single *.zip* file. The .zip file was encrypted using password *PW123*, with a resulting file size of 36,004 bytes. Figure 9a depicts the files.

For the benign carrier file, we used the *Stanford_Lucy_ASCII.stl* file depicted in Figure 9b. The file contained 68,646 facets. Based on our usage of the *vertex order* encoding primitive, the maximum capacity of the carrier file was 46,080 bytes. Considering the reservation of four bytes for the *size* field, up to 46,076 bytes of *secret* can be encoded in this carrier STL file, which is sufficient for the selected example.

The console output of the encoding process is presented in Figure 9c. With the above mentioned carrier file and payload, the overall duration of encoding was 9 sec and of decoding 5 sec. To verify the lossless extraction of the data, we calculated and printed out on the console the MD5 cryptographic hash sum of the encoded secret. We further calculated the checksum over the secret extracted from the used carrier STL file. The MD5 value for the decoded secret was exactly the same. This indicates that the hidden binary file can be retrieved without loss.

We verified the ability of the proposed approach to avoid detection by loading models in two different slicers (*Cura* and *CatalystEx*), and 3D printing both the original design and the carrier file with encoded secret on two different 3D printers *Lulzbot Taz 6* and *Stratasys Dimension Elite* [5]. Neither showed indicators of manipulation from the embedded payload. Screenshots and photos for side-by-side comparison are in Appendix A.2.

7 DISCUSSION

7.1 Steganography with Engineering Designs

There is a large body of literature covering both theoretical and practical steganography (see Section 3.2 for a brief outline). This raises the question of novelty, considering the existence of steganographic techniques for other document formats. A major difference is that, while the file formats traditionally used in steganography are "end-user facing" digital audio, video, image, or text documents, the proposed approach manipulates an engineering design file.

In any approach, the steganographically encoded information should not interfere with the ability of programs to correctly process it. With data formats for audio, video, and textual information, this corresponds to the ability to represent it to an end-user. In the case considered in this paper, modifications introduced in the digital design file should not interfere with the ability to 3D print the object. In Section 6 we demonstrated that encoding data in an STL using the proposed approach does not interfere with 3D printing.

[5]Equipment and software identified in this paper do not imply recommendation or endorsement by the authors or their organisations.

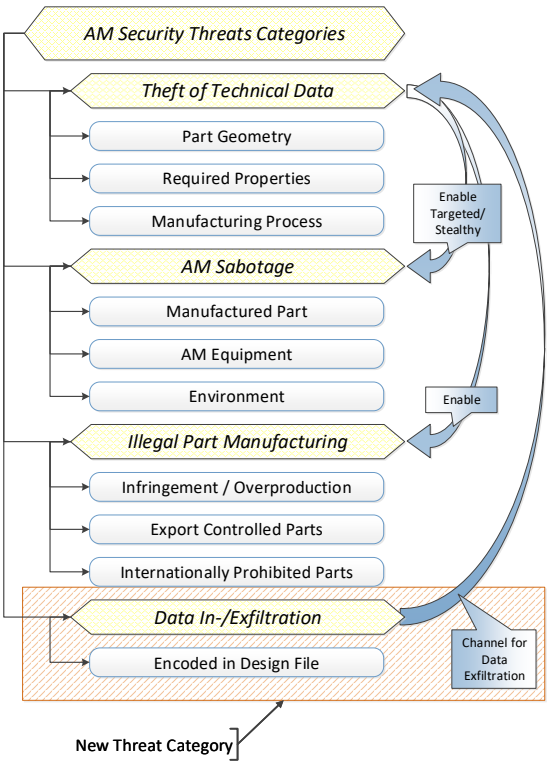


Figure 10: Extended Taxonomy of AM Security Threat Categories – Introduced in this Paper Use of 3D Printing Design Files for Data In- and Exfiltration is New (based on [24, 66])

To evade suspicion, the distortions introduced in user-facing data formats should be imperceptible to a human. In the case of engineering design files, it is more complicated. Geometric distortions introduced in manufactured parts could interfere with their integration in a designated system. Furthermore, distortions introduced in the 3D printing process could degrade the part's mechanical characteristics, and cause its premature failure (like in the case of deliberate sabotage attacks outlined in Section 3.2). All these "side effects" would likely trigger an investigation and lead to the discovery of the channel.

7.2 Implications for the AM Security Field

The presented attack has profound implications for the AM Security field as a whole. As discussed in Section 3.2, up until now only three security threat categories have been identified and discussed (theft of technical data, sabotage attacks, and illegal part manufacturing) [66], with the theft of technical data being a common prerequisite for illegal part manufacturing and targeted sabotage attacks [24]. The presented work extends the threat categories with yet another, *Data In-/Exfiltration* (see Figure 10).

As mentioned throughout this paper, the introduced steganographic communication channel can be used in a variety of attacks (see specifically Section 2). Among others, this channel can be used for malicious actions like the exfiltration of stolen information,

infiltration of malicious software, or hidden storage of content the possesion of which is illegal.

In the AM Security context, the attacks introduced in this paper can be used as an additional communication channel through which the stolen AM technical data can be exfiltrated. Therefore, it can also act as an additional precursor for two other threat categories.

Considering that the works on the 3D printed object watermarking via modifications of the 3D object geometry (discussed in Section 3.2) can be re-purposed as a hidden communication channel, we propose two sub-categories for the new security threat: *Encoded in Design File* and *Encoded in 3D Printed Object.*

7.3 Three Contexts of Distinguishability

Throughout Sections 4 and 5, we introduced and discussed three distinct contexts of *distinguishability*. We think that this is important to reiterate these in a condensed manner, because each and any of these can determine a defender's ability to detect the subliminal steganographic channel and even extract the encoded message for further analysis. Such detection is critical for follow-on security procedures, such as attribution of the communication to IP/e-mail addresses and thus attribution of the malicious actors.

First, in the case of the weak attack (defined in Sections 4), the question is whether it is possible to distinguish between parts of the carrier STL file that are steganographically encoded with the subliminal message and those that are not. This distinction would allow extraction and analysis of the embedded message. This depends on the behavior of the CAD program using the modified STL file. Should the order in which vertices are specified be fully random, the weak attack could be distinguishable because the encoded content is not distributed pseudo-randomly. Should a CAD program exhibit deterministic behavior here instead, both the altered distribution of a weak attack and the pseudo-random distribution of the encrypted file in a regular attack could stand out when the encoded file does not exhaust the encoding capacity of the carrier STL.

In the strong attack (described in Section 5) we addressed the concern of distinguishability within a carrier file by encoding every single bit with either the secret message or a random pad. This ensures that the distribution of vertex order within STL files with an encoded message is always pseudo-random. However, we identified a second distinguishability context – between STL files that contain a fully-encrypted steganographically encoded message and those that don't. To address this, we proposed to encode a random bit stream even when no real message is embedded in the STL.

This led us to the third context of distinguishability – between system(s) on the corporate network that are compromised and encode messages (either real or random) in all STL files and those that are benign. Currently, we see no way an attacker can address this kind of distinguishability. Therefore, this might be used by a defender to identify systems that are compromised.

We should note that the distinguisheability has to be addressed for all alternative encoding approaches described in Appendix A.1.

7.4 Robustness Against STL File Sanitation

Countermeasures to the proposed attack can fall into different categories, similar to ransomware countermeasures [5, 38]. In addition to the question of detection (based on the distinguishability), there is the question of prevention. Specifically, how robust is the proposed steganographic communication channel against disruption efforts that are limited by the same constraint – avoiding impacts on the 3D design? Such efforts are especially relevant in environments where exfiltration of highly sensitive information might be a security concern, and can be applied in addition to attack detection.

With the proposed bit encoding scheme, the "sanitization" of STL files is fairly simple and straightforward. Defenders can overwrite all facets in the file with either the same value (0 or 1) or with a completely random value. As mentioned before, due to the semantic equivalency of the vertex cyclical rotation, it should not affect the described 3D object geometry. However, while theoretically sound, this assumption needs to be verified empirically; we plan to conduct such an investigation in a follow-up work. Consequently, the proposed approach is fragile, and can be easily disrupted by the described sanitation approach.

We should note, however, that alternative encoding techniques are possible (see Appendix A.1). Similar to distinguishability, each of these encoding techniques would require a different sanitation approach. Any technique meeting the same criteria as our selected encoding, non-interference with printed geometry, would exhibit the same level of fragility against sanitation.

Alternatively, incoming and outgoing STL files can be read into a CAD program and re-saved, assuming that this step will remove all possible steganographically encoded information, regardless of the encoding primitives used. However, if such a CAD program is trojanized, it can be used by a malicious actor for the encoding of secret information instead of removing it.

8 CONCLUSION

In this paper we demonstrated that even engineering design files can be used as steganographic channels. While the ability to encode information is not surprising, the restrictions on such files are significantly tighter than in the case of user-facing audio, video, and text data formats. In the case of Additive Manufacturing (AM), encoding at a minimum should not disrupt the ability to manufacture the part; furthermore, in the case of functional parts, even slight distortions can be unacceptable.

We focus on the most popular digital design format in 3D printing, STL. We identified several sources of entropy that can be used to encode individual bits of information. Upon choosing one—the order in which vertices are listed—we showed how a raw steganographic channel can be defined. We further introduced a strong attack, an approach that illustrates how the proposed steganographic channel can be fully encrypted. To demonstrate a legally questionable application, we encoded and later recovered several design files for the *Liberator* printed pistol. We used this example to verify that the proposed approach does not interfere with printability nor does it introduce any visually noticeable distortions.

In the future, we plan to develop an automatic distinguisher for the sources of entropy we identified. We intend to use it to study the actual distribution of STL files produced by various CAD programs, and to explore all three contexts of distinguishability we identified for STL. We further believe that the steganographic channel could be used to integrate watermarks in STL files, thus protecting IP in Additive Manufacturing.

REFERENCES

[1] 2014. The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/. (2014). http://graphics.stanford.edu/data/3Dscanrep/

[2] 2018. 3D Printing Business Directory. https://www.3dprintingbusiness.directory. (2018). [Online; accessed 29-May-2019].

[3] 2019. Watermark3D. https://www.watermark3d.com. (2019). [Online; accessed 10-December-2019].

[4] Thomas Agrikola, Geoffroy Couteau, Yuval Ishai, Stanisław Jarecki, and Amit Sahai. 2020. On pseudorandom encodings. In *Theory of Cryptography Conference*. Springer, 639–669.

[5] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. 2018. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security* 74 (2018), 144–166.

[6] Ross J Anderson and Fabien AP Petitcolas. 1998. On the limits of steganography. *IEEE Journal on selected areas in communications* 16, 4 (1998), 474–481.

[7] Sofia Belikovetsky, Mark Yampolskiy, Jinghui Toh, Jacob Gatlin, and Yuval Elovici. 2017. dr0wned – Cyber-Physical Attack with Additive Manufacturing. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, Vancouver, BC, 16. https://www.usenix.org/conference/woot17/workshop-program/presentation/belikovetsky

[8] Matt Bishop, Sophie Engle, Deborah A Frincke, Carrie Gates, Frank L Greitzer, Sean Peisert, and Sean Whalen. 2010. A risk management approach to the "insider threat". In *Insider threats in cyber security*. Springer, 115–137.

[9] Danton Bryans. 2015. Unlocked and loaded: government censorship of 3D-printed firearms and a proposal for more reasonable regulation of 3D-printed goods. *Ind. LJ* 90 (2015), 901.

[10] Christian Cachin. 2024. An information-theoretic model for steganography. *Inf. Comp.* (2024), 41–56.

[11] François Cayre and Benoit Macq. 2003. Data hiding on 3-D triangle meshes. *IEEE Transactions on signal Processing* 51, 4 (2003), 939–949.

[12] CBS. 2020. Austin-based company attempting to distribute 3D gun blueprints online hits legal setback. (Jan 2020). https://cbsaustin.com/news/local/austin-based-company-attempting-to-distribute-3d-gun-blueprints-online-hits-legal-setback

[13] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. 2010. Digital image steganography: Survey and analysis of current methods. *Signal processing* 90, 3 (2010), 727–752.

[14] Fei Chen, Yuxi Luo, Nektarios Georgios Tsoutsos, Michail Maniatakos, Khaled Shahin, and Nikhil Gupta. 2019. Embedding tracking codes in additive manufactured parts for product authentication. *Advanced Engineering Materials* 21, 4 (2019), 1800495.

[15] Fei Chen, Gary Mac, and Nikhil Gupta. 2017. Security features embedded in computer aided design (CAD) solid models for additive manufacturing. *Materials & Design* 128 (2017), 182–194.

[16] Fei Chen, Jian H Yu, and Nikhil Gupta. 2019. Obfuscation of embedded codes in additive manufactured components for product authentication. *Advanced engineering materials* 21, 8 (2019), 1900146.

[17] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. 2007. *Digital watermarking and steganography*. Morgan kaufmann.

[18] Howard Kuhn David L. Bourell, William Frazier and Mohsen Seifi. 2020. *ASM Handbook Volume 24 – Additive Manufacturing Processes*. ASM International.

[19] Nenad Dedić, Gene Itkis, Leonid Reyzin, and Scott Russell. 2009. Upper and lower bounds on black-box steganography. *Journal of Cryptology* 22, 3 (2009), 365–394.

[20] Arnaud Delmotte, Kenichiro Tanaka, Hiroyuki Kubo, Takuya Funatomi, and Yasuhiro Mukaigawa. 2019. Blind Watermarking for 3D Printed Objects by Locally Modifying Layer Thickness. *IEEE Transactions on Multimedia* (2019).

[21] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 46–64.

[22] Carl Franzen. 2014. 3D-printed gun maker in Japan sentenced to two years in prison. https://www.theverge.com/2014/10/20/7022809/3d-printed-gun-maker-in-japan-sentenced-2-years. (2014).

[23] Sarah Goehrke. 2019. *A Look Ahead in 3D Printing with Gartner's Pete Bailiere*. https://www.fabbaloo.com/blog/2019/1/10/a-look-ahead-in-3d-printing-with-gartners-pete-basiliere

[24] Lynne MG Graves, Joshua Lubell, Wayne King, and Mark Yampolskiy. 2019. Characteristic Aspects of Additive Manufacturing Security From Security Awareness Perspectives. *IEEE Access* 7 (2019), 103833–103853.

[25] Andy Greenburg. 2013. 3D-Printed Gun's Blueprints Downloaded 100,000 Times In Two Days "("With Some Help From Kim Dotcom")". (May 2013). https://www.forbes.com/sites/andygreenberg/2013/05/08/3d-printed-guns-blueprints-downloaded-100000-times-in-two-days-with-some-help-from-kim-dotcom/#86151c710b88

[26] Frank Hartung and Martin Kutter. 1999. Multimedia watermarking techniques. *Proc. IEEE* 87, 7 (1999), 1079–1107.

[27] LEE Heung-Kyu, Jong-Uk Hou, Hak-Yeol Choi, SONG Hyun-Ji, Do-Gon Kim, and Han-Ul Jang. 2019. Watermark embedding apparatus and method, and watermark detecting apparatus and method for 3D printing environment. (Oct. 2019).

[28] Whitney Hipolite. 2015. 3D printable files for Cody Wilson's Liberator Gun are Now Available to All on 3DShare. (Jun 2015). https://3dprint.com/73842/download-3d-printed-gun/

[29] Ivan Homoliak, Flavio Toffalini, Juan Guarnizo, Yuval Elovici, and Martín Ochoa. 2019. Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 1–40.

[30] Nicholas J Hopper, John Langford, and Luis Von Ahn. 2002. Provably secure steganography. In *Annual International Cryptology Conference*. Springer, 77–92.

[31] Jong-Uk Hou, Do-Gon Kim, Sunghee Choi, and Heung-Kyu Lee. 2015. 3d print-scan resilient watermarking using a histogram-based circular shift coding structure. In *Proceedings of the 3rd ACM Workshop on Information Hiding and Multimedia Security*. ACM, New York, NY, USA, 115–121.

[32] Jong-Uk Hou, Do-Gon Kim, and Heung-Kyu Lee. 2017. Blind 3D Mesh Watermarking for 3D Printed Model by Analyzing Layering Artifact. *IEEE Transactions on Information Forensics and Security* (2017).

[33] Amir Houmansadr, Thomas J Riedl, Nikita Borisov, and Andrew C Singer. 2013. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention.. In *NDSS*.

[34] ASTM International. 2013. F2792-12a-Standard Terminology for Additive Manufacturing Technologies. *Rapid Manufacturing Association* 12 (2013), 10–12.

[35] Satoshi Kanai, Hiroaki Date, and Takeshi Kishinami. 1998. Digital watermarking for 3D polygons using multiresolution wavelet decomposition. In *Proc. Sixth IFIP WG*, Vol. 5. 296–307.

[36] Stefan Katzenbeisser and Fabien Petitcolas. 2015. *Information Hiding*. Artech House, UK.

[37] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. Cutting the gordian knot: A look under the hood of ransomware attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[38] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. PayBreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 599–611.

[39] Alexandr Kuznetsov, Oleh Stefanovych, Yuriy Gorbenko, Oleksii Smirnov, Victor Krasnobaev, and Kateryna Kuznetsova. 2019. Information Hiding Using 3D-Printing Technology. In *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Vol. 2. IEEE, 701–706.

[40] Gerhard C Langelaar, Iwan Setyawan, and Reginald L Lagendijk. 2000. Watermarking digital image and video data. A state-of-the-art overview. *IEEE Signal processing magazine* 17, 5 (2000), 20–46.

[41] Heung-Kyu Lee, Han-Ul Jang, Hak-Yeol Choi, Jeongho Son, Seung-Min Mun, Dongkyu Kim, Jong-Uk Hou, and Wookhyung Kim. 2017. THREE-DIMENSIONAL MESH MODEL WATERMARKING METHOD USING SEGMENTATION AND APPARATUS THEREOF. (Dec 2017). https://patents.google.com/patent/US20170213310A1/en Patent No. US 2017/0213310 A1, Filed December 21, 2016.

[42] Benoît Macq, Patrice Rondao Alface, and Mireia Montanola. 2015. Applicability of watermarking for intellectual property rights protection in a 3D printing scenario. In *Proceedings of the 20th International Conference on 3D Web Technology*. ACM, New York, NY, USA, 89–95.

[43] Priyanka Mahesh, Akash Tiwari, Chenglu Jin, Panganamala R Kumar, AL Narasimha Reddy, Satish TS Bukkapatanam, Nikhil Gupta, and Ramesh Karri. 2020. A Survey of Cybersecurity of Digital Manufacturing. *Proc. IEEE* (2020).

[44] Toshio Motoki. 2013. Method and system for adding and confirming unique code for output object output from 3D printer. (Oct 2013). https://patents.google.com/patent/JP6331324B2 Patent No. JP6331324B2, Filed October 21, 2013.

[45] Ryutarou Ohbuchi, Hiroshi Masuda, and Masaki Aono. 1997. Watermarking three-dimensional polygonal models. In *ACM multimedia*, Vol. 97. Citeseer, 261–272.

[46] Ryutarou Ohbuchi, Hiroshi Masuda, and Masaki Aono. 1998. Watermarking three-dimensional polygonal models through geometric and topological modifications. *IEEE Journal on selected areas in communications* 16, 4 (1998), 551–560.

[47] Ryutarou Ohbuchi, Shigeo Takahashi, Takahiko Miyazawa, and Akio Mukaiyama. 2001. Watermarking 3D polygonal meshes in the mesh spectral domain. In *Graphics interface*, Vol. 2001. Citeseer, 9–17.

[48] Fabien AP Petitcolas, Ross J Anderson, and Markus G Kuhn. 1999. Information hiding-a survey. *Proc. IEEE* 87, 7 (1999), 1062–1078.

[49] Emil Praun, Hugues Hoppe, and Adam Finkelstein. 1999. Robust mesh watermarking. In *Siggraph*, Vol. 99. Citeseer, 49–56.

[50] Jaco Prinsloo, Saurabh Sinha, and Basie von Solms. 2019. A Review of Industry 4.0 Manufacturing Process Security Risks. *Applied Sciences* 9, 23 (2019), 5105.

[51] Niels Provos and Peter Honeyman. 2003. Hide and seek: An introduction to steganography. *IEEE security & privacy* 1, 3 (2003), 32–44.

[52] Patrick Roberts. 2018. Where To Find 3D Printed Gun Files. (Aug 2018). https://www.recoilweb.com/where-to-find-3d-printed-gun-files-140438.html

[53] Clare Scott. 2019. *Treatstock Introduces Watermark 3D Software Solution for IP Protection of 3D Printable Files*. https://3dprint.com/196192/treatstock-watermark-

3d
[54] Yan Shoshitaishvili, Luca Invernizzi, Adam Doupe, and Giovanni Vigna. 2014. Do you feel lucky? A large-scale analysis of risk-rewards trade-offs in cyber security. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. 1649–1656.
[55] Piyarat Silapasuphakornwong, Hideyuki Torii, Masahiro Suzuki, and Kazutake Uehira. 2019. 3D Printing Technique That Can Record Information Inside An Object As Rewritable. In *NIP & Digital Fabrication Conference*, Vol. 2019. Society for Imaging Science and Technology, 158–161.
[56] L Sturm, CB Williams, JA Camelio, J White, and R Parker. 2014. Cyber-physical vunerabilities in additive manufacturing systems. *Context* 7 (2014), 8.
[57] Olivier Thonnard, Leyla Bilge, Gavin O'Gorman, Seán Kiernan, and Martin Lee. 2012. Industrial espionage and targeted attacks: Understanding the characteristics of an escalating threat. In *International workshop on recent advances in intrusion detection*. Springer, 64–85.
[58] Francesca Uccheddu. 2018. A method for masking instructions programs for numerical control machines for 3d printing. (Dec 2018). https://patents.google.com/patent/WO2018122597A1 Patent No. WO2018122597A1, Filed December 30, 2016.
[59] Verizon. 2020. *2020 Data Breach Investigations Report.* Technical Report. https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf
[60] Luis Von Ahn and Nicholas J Hopper. 2004. Public-key steganography. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 323–341.
[61] Kai Wang, Guillaume Lavoué, Florence Denis, and Atilla Baskurt. 2008. A comprehensive survey on three-dimensional mesh watermarking. *IEEE Transactions on Multimedia* 10, 8 (2008), 1513–1527.
[62] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. 2012. StegoTorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 109–120.
[63] Philipp Winter, Tobias Pulls, and Juergen Fuss. 2013. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. 213–224.
[64] Terry Wohlers. 2018. *Wohlers Report 2017 3D Printing and Additive Manufacturing State of the Industry Annual Worldwide Progress Report.* Wohlers Associates, Inc., Fort Collins, Colorado, USA. www.wohlersassociates.com.
[65] Mark Yampolskiy, Wayne King, Gregory Pope, Sofia Belikovetsky, and Yuval Elovici. 2017. Evaluation of additive and subtractive manufacturing from the security perspective. In *International Conference on Critical Infrastructure Protection*. Springer, 23–44.
[66] Mark Yampolskiy, Wayne E King, Jacob Gatlin, Sofia Belikovetsky, Adam Brown, Anthony Skjellum, and Yuval Elovici. 2018. Security of Additive Manufacturing: Attack Taxonomy and Survey. *Additive Manufacturing* (2018).
[67] Mark Yampolskiy, Anthony Skjellum, Michael Kretzschmar, Ruel A Overfelt, Kenneth R Sloan, and Alec Yasinsac. 2016. Using 3D Printers as Weapons. *International Journal of Critical Infrastructure Protection* 14 (2016), 58–71.
[68] Adam Young and Moti Yung. 1996. Cryptovirology: Extortion-based security threats and countermeasures. In *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 129–140.
[69] Adam L. Young and Moti Yung. 1996. The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone?. In *Crypto*. Springer, 89–103.
[70] Adam L. Young and Moti Yung. 1997. Kleptography: Using Cryptography Against Cryptography. In *Eurocrypt*. Springer, 62–74.
[71] Adam L. Young and Moti Yung. 1997. The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In *Crypto*. Springer, 264–276.
[72] Adam L. Young and Moti Yung. 2004. *Malicious Cryptography: exposing Cryptovirology*. Wiley, USA.
[73] Steven Eric Zeltmann, Nikhil Gupta, Nektarios Georgios Tsoutsos, Michail Maniatakos, Jeyavijayan Rajendran, and Ramesh Karri. 2016. Manufacturing and Security Challenges in 3D Printing. *JOM* (2016), 1–10.
[74] Li Zhuorong. 2019. A kind of digital watermarking system for stereolithography art threedimensional model file. (Mar 2019). https://patents.google.com/patent/CN109903213A Patent No. CN109903213A, Filed March 1, 2019.

A APPENDICES

A.1 Alternative Bit Encoding Primitives

In addition to bit encoding via vertex order in a facet, the STL file format allows several alternatives that we identified and discarded for this particular work. Below we outline those alternatives and provide reasons why we deemed these deficient for the purpose of data exfiltration. Please note that for other application scenarios, such as watermarking, these might be more valuable.

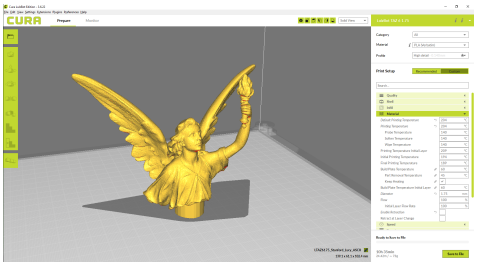
First, specifically for ASCII STL files, delimiters (space and tabulator characters) as well as comment characters can be used to encode information. For example, at the end of every line that does not contain a comment, a single space added can represent encoded bit value 0 and two spaces can represent encoded bit value 1, and any other number of spaces would indicate that this line contain no valid encoded bit and can be skipped. This is a very simple encoding scheme that would result in an encoding capacity of up to 7 bits per facet (corresponding to opening tags *facet normal* and *outer loop*, three *vertex* entries, and closing tags *endloop* and *endfacet* (see Figure 1); this encoding capacity can be further expanded if delimiters within the particular lines are used as well. The biggest drawback of this encoding scheme is that it can be immediately detected by a simple statistical analysis of the file. Its sanitation would also be simple by the replacement of any number of spaces by a single space.

All coordinate values in the description are float pointers and can be specified using an exponent. This can provide a way to encode one or more bits at once per value. For example, if 0 is the first or only character specified before decimal point, it can represent encoded bit value 0, and if the first number before the decimal point is non-zero - value 1. The impact of this bit encoding technique on the 3D printed object is not clear, because it can depend on how the 3D printing software imports and processes the values. This could cause slight but eventually impactful deviations due to numerical errors. The biggest drawback of this encoding technique is that the presence of the encoded bit stream can be easily spotted. The sanitation is also simple, but its potential impact on the 3D printed part has to be investigated.

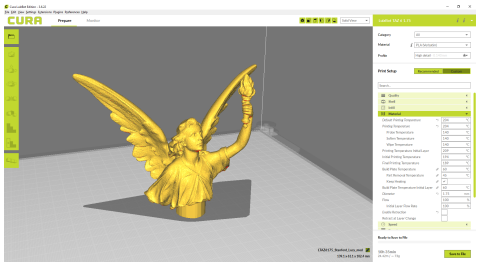
Another alternative for the bit encoding primitive is "flipping" the facet's normal. For example, the normal direction following the right hand rule could represent encoded bit value 0, and if it violates the rule - bit value 1. As the description indicates, it will violate the right hand rule and eventually impact the 3D printed object (even though some AM software simply ignores the specified normal value and re-calculates it from the vertices defining the facet). For data exfiltration, it will also provide an easily detectable encoding approach. Sanitation would require re-calculating and updating each normal value.

Something similar can be achieved by reversing the direction in which the vertices are listed. The only advantage of this approach is that it will increase the encoding capacity, because, in addition to selectively violating the right hand rule, the bit encoding can be structured similarly to the approach selected for this paper. The drawbacks are similar to flipping the normal vector direction - there is a potential impact on the 3D printed object and the approach is easily detected. The sanitation as well can be done by re-calculating normal value; this, however, might also have an impact on the 3D printed object.

The last encoding alternative that we identified is the representation of bits of information through the order in which individual facets are listed. Assuming that no two facets in a STL file have exactly the same coordinates of all three vertices, facets can be distinguished from each other, and as such their position in STL file can represent the encoded value. A simple encoding scheme based on this observation would be to leave two consecutive facets



(a) Original Carrier STL File (Unmodified), loaded in *Cura*



(b) Carrier STL File with Embedded Payload, loaded in *Cura*

(c) 3D-Printed Model using Original Design File, front view

(d) 3D-Printed Model using Modified Design File (with encoded information), front view

Figure 11: Side-by-Side Comparison using *Cura* Slicer and *Lulzbot Taz 6* 3D Printer

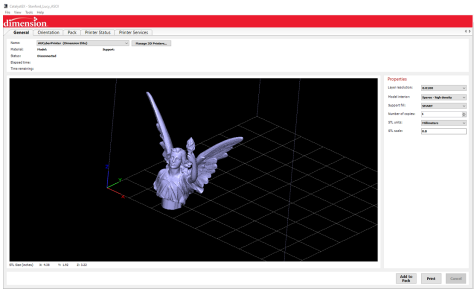
in their original order to represent bit value 0 and swap them to represent bit value 1. Whether or not this approach will cause slight deviations of 3D geometry rooted in numerical errors need to be investigated; this is a task that is outside of this paper's scope. The potential advantage of this encoding technique is that a bit stream encoded with it might be harder to detect in the STL^M file if the original STL file is not available to the analyst. However, this is also the biggest drawback of this technique – in order to decode a bit stream, a recipient needs to have access to the "original" STL file. Sanitation can be achieved by randomly swapping consecutive facets; like with the encoding itself, the implications of this approach on the 3D printed object are not clear.

A.2 Verification of 3D-Printability

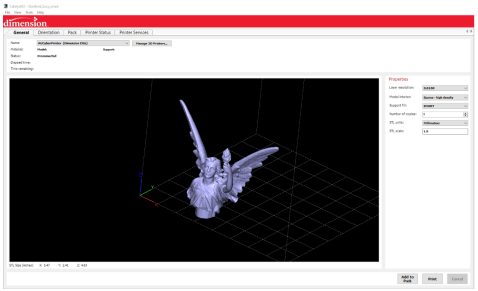
We verified the ability of the proposed approach to avoid detection by loading models in two different slicers (*Cura* and *CatalystEx*), and 3D printing both the original design and the carrier file with encoded secret on two different 3D printers, the *Lulzbot Taz 6* and the *Stratasys Dimension Elite*.

Figure 11 shows a side-by-side comparison of the STL files loaded in *Cura* and then 3D-printed on the *Lulzbot Taz 6*. Figure 12 shows a side-by-side comparison of the STL files loaded in *CatalystEx* and then 3D-printed on the *Stratasys Dimension Elite*.

Neither side-by-side comparison of the sliced digital models nor of the 3D printed objects show indicators of manipulation from the embedded payload. Therefore, if the detection of the presented attack depends solely on the ability to 3D print a correct model, it will fail.



(a) Original Carrier STL File (Unmodified), loaded in *CatalystEx*



(b) Carrier STL File with Embedded Payload, loaded in *CatalystEx*

(c) 3D-Printed Model using Original Design File, front view

(d) 3D-Printed Model using Modified Design File (with encoded information), front view

(e) 3D-Printed Model using Original Design File, rear view

(f) 3D-Printed Model using Modified Design File (with encoded information), rear view

Figure 12: Side-by-Side Comparison using *CatalystEx* Slicer and *Stratasys Dimension Elite* 3D Printer