Self-embedding watermarking method for G-code used in 3D printing

Zhenyu Li*, Daofu Gong, Lei Tan, Xiangyang Luo, Fenlin Liu
State Key Laboratory of Mathematical
Engineering and Advanced Computing
Zhengzhou, China
zheenyuli@gmail.com

Adrian G. Bors
Department of Computer Science
University of York
York, UK
adrian.bors@york.ac.uk

***Abstract*—3D printing is faced with a lot of security issues, such as malicious tampering, intellectual property theft and so on. This work aims to protect the G-code file which controls the 3D printing process by proposing a self-embedding watermarking method for G-code file. This method groups the G-code lines into code blocks and achieves a random mapping relationship for each code block. Each code block is divided into two parts, carrying the authentication and recovery bits, respectively. The tampered regions are detected by leveraging the authentication bits in each code block. Meanwhile, the G-code files are restored based on the recovery bits and the geometric information of the neighboring code blocks. Experimental results indicate that the proposed method can effectively detect the tampered region and restore the G-code file to a large extent, while limiting the distortion caused to the 3D printed object by the watermarking.**

***Index Terms*—Watermarking, G-code, 3D Printing, Self-embedding**

I. Introduction

3D printing, also known as Additive Manufacturing (AM), is a significant technology for Industry 4.0. It has been widely applied to many critical industries, such as car manufacturing, healthcare, aeronautics, construction [1]. Therefore, the quality and integrity of the 3D printed product is of great importance to the security and reliability of its applications.

Disconcertingly, as more and more network-based 3D printers are emerging, 3D printing is facing a lot of security challenges and cyber-physical vulnerabilities. For example, an attacker can get access to the PC connected to the 3D printer and then operate various kinds of attacks. It is demonstrated that an attacker can create defective parts by adding a malicious infill void in the Computer-Aided Design (CAD) files [2]. Meanwhile, it is possible to manipulate the printing settings, such as high-temperature heaters, to cause a physical hazard, similar to the Stuxnet [3]. In addition, if an attacker would successfully apply man-in-the-middle attack between the PC and the 3D printer, it may maliciously tamper the G-code files controlling the movement of the tools in 3D printer. Comparing to tampering the CAD files, tampering the G-code files is more imperceptible, because usually little attention is paid to the G-code file. Besides, the intellectual property of the to-be-printed 3D model may be stolen by the attacker as well.

3D security issues are attracting increasing attention from academia [4]–[11], especially the security of 3D printing. Gao *et al.* [12] proposed an online monitoring approach to defend against the adverse modifications of critical printing attributes specified by the firmware of 3D printers. This approach leverages multiple sensors to monitor the printing process, which has the ability of reconstructing the infill path and printing speed, analyzing the layer thickness and estimating the fan speed. The attacks can be detected by comparing the results obtained by the monitoring system and the original printing settings. More recently, Rais *et al.* [13] proposed a framework for checking the integrity of 3D printing process, utilizing the G-code as the ground truth to verify the acquired sensors' data, in order to determine if the printing process has been attacked. It is obvious that this framework is based on the assumption that the security of G-code is not compromised by an attacker. Besides, both of the methods above have the limitation that they can only detect the attack after the printing is finished, while cannot prevent the happening of the comprised printing process in advance.

In order to overcome these limitations, the authentication of the G-code files before printing is essential, as the G-code files represent the ultimate 3D printer control mechanism by means of machine readable command lines.

Fragile watermarking has been widely adopted for the aim of authentication of digital images, audio, video, 3D objects and other digital files [14]–[18]. For instance, fragile watermarking algorithms for digital images have the ability of detecting, localizing and recovering tampered pixels in the image. There are some inspiring watermarking methods for 3D CAD object authentication [19], [20], but they are not applicable in the context of 3D printing. The G-code file, which is obtained after the slicing of a CAD object, is dramatically different from the structure of the CAD object.

In this study, we propose a self-embedding watermarking method for the G-code file for 3D printing. Furthermore, a novel measurement for assessing the difference between two G-code files is proposed in this paper. To the best of our knowledge, this is the first study for the G-code file

This work was supported by National Natural Science Foundation of China (62002387, U1736214, 61872448, 62182435, 61772549) and Zhongyuan Science and Technology Innovation Leading Talent Project (No. 214200510019).*The corresponding author is Zhenyu Li.

authentication and recovery using watermarking.

II. Background

A. 3D Printing Preliminaries

There are several types of 3D printing technologies, while the one used by most desktop 3D printers is the Fused Deposition Modelling (FDM). We consider a FDM 3D printer as an example to explain the process of 3D printing, as illustrated in Fig. 1. First, a digital 3D object, which is in the format of StereoLithography (STL) file, is created by the user using CAD software. Then, the STL file is converted to the machine language, namely a G-code file, by the slicing software, such as Cura and Slic3r. In this step, the 3D object seems like sliced into multiple layers, whereas it is planning the sequence of operations for 3D printing of the object. Besides, many other parameters of printing are set by the slicing software, such as the heating temperature, fan speed, extrusion rate, and so on. Finally, the G-code file is sent to the 3D printer which is loaded with a spool of filament and a heated nozzle. The filament is melt by the nozzles and laid down at precise locations of a platform specified in the G-code. Once a layer is finished, the platform moves down and the process repeats until the part is complete.

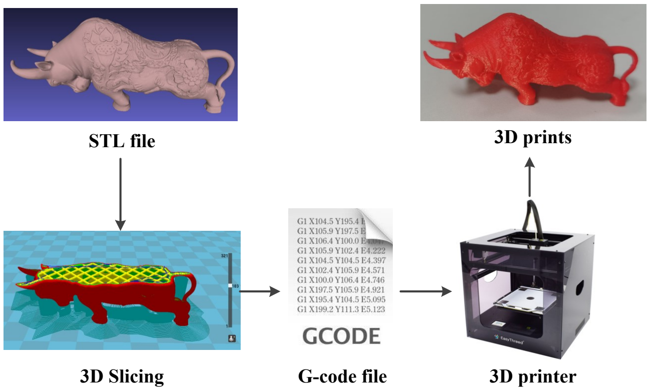
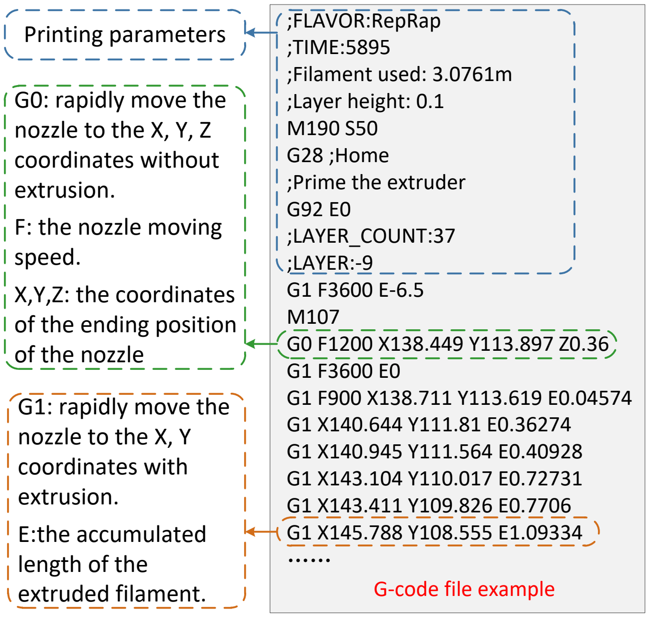


Fig. 1. The illustration of 3D printing process.

B. G-code File Format

G-code, which stands for “Geometric Code”, is the most widely used Computer Numerical Control (CNC) programming language in both subtractive and additive manufacturing. To put it simply, G-code instructs the tools where to move, how fast to move and what path to follow. In the case of 3D printing, a G-code file includes the printing parameters and the instructions controlling the movement of the nozzle. An example of G-code file is shown in Fig. 2.

G0 and G1 are the top two most frequently used codes in 3D printing. More specifically, G0 command moves the nozzle at maximum travel speed from a current position to the point specified by the X, Y and Z values without extrusion. The X and Y axes represent the lateral and longitudinal position of a point, while Z-axis represents the elevation of the point. G1 means move in straight line to a specific position with extrusion whose length is determined by E value, signifying the cumulative amount of filament that has been extruded so far. In addition, we have the F value, denoting the speed (in mm/min) at which the nozzle is traveling. G0 and G1 do not require values for all axes and only need to be set once and be valid until it would be changed. So the Z value is usually set at the beginning of building each new layer of the 3D print. The range of X and Y values is related to the size of the 3D printer's platform, because the 3D printed object is usually placed in the center of the platform during printing and the origin of the X and Y-axis is in one corner of the platform.


Printing parameters
G0: rapidly move the nozzle to the X, Y, Z coordinates without extrusion.
F: the nozzle moving speed.
X,Y,Z: the coordinates of the ending position of the nozzle
G1: rapidly move the nozzle to the X, Y coordinates with extrusion.
E:the accumulated length of the extruded filament.
G-code file example

```
;FLAVOR:RepRap
;TIME:5895
;Filament used: 3.0761m
;Layer height: 0.1
M190 S50
G28 ;Home
;Prime the extruder
G92 E0
;LAYER_COUNT:37
;LAYER:-9
G1 F3600 E-6.5
M107
G0 F1200 X138.449 Y113.897 Z0.36
G1 F3600 E0
G1 F900 X138.711 Y113.619 E0.04574
G1 X140.644 Y111.81 E0.36274
G1 X140.945 Y111.564 E0.40928
G1 X143.104 Y110.017 E0.72731
G1 X143.411 Y109.826 E0.7706
G1 X145.788 Y108.555 E1.09334
……
```

Fig. 2. The illustration of G-code file format.

III. The Proposed Self-Embedding Watermarking Method for G-code

A. The Watermarking Framework

The proposed self-embedding watermarking method for G-code file includes the watermark embedding, authentication and recovery of the G-code. The framework of the proposed method is diagramed in Fig. 3. During the watermark embedding procedure, the command lines in the G-code file are grouped into code blocks. Then, the code blocks are randomly permuted to obtain a mapping block for each code block. Every code block is divided into two parts. The X and Y coordinates in the first part of code block are used to generate authentication and recovery bits. Afterwards, the authentication bits of one code block is embedded into the multiple least significant bits of the X and Y coordinates in the first part of the block. Simultaneously, the recovery bits for a code block are embedded into the second part of its mapping block.

Once the 3D printer receives the G-code file compromised by potential attacks, it identifies the tampered code blocks by verifying the authentication bits in the first part of each

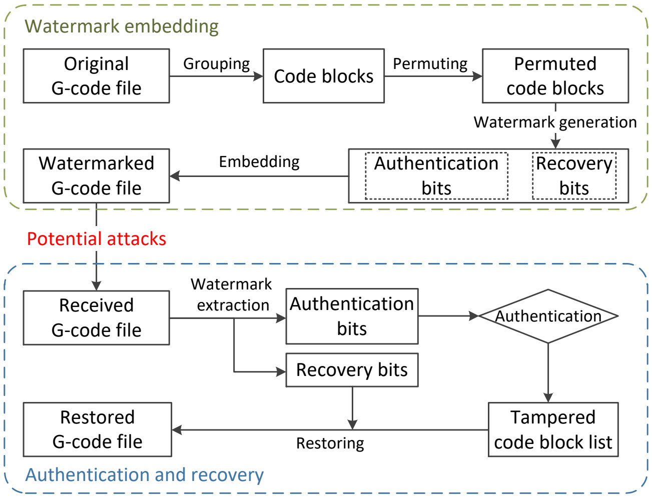


Fig. 3. The diagram of the proposed watermarking framework.

code block. Then, in the recovery stage, if no code block is identified as tampered, the first part of each code block is fully restored based on the recovery bits embedded in its mapping block. However, if any code block is tampered and its mapping block is authentic, the first part of this block can still be fully restored based on the recovery bits embedded in its mapping block. In addition, the second part of this block is restored based on the geometric information from the neighboring blocks. When one code block and its mapping block are both tampered, the recovery of the code block is merely based on the geometric information from the neighboring blocks.

B. Watermark Embedding

The detailed steps of the watermark embedding process for a given G-code file are as follows.

1) Group G0 and G1 command lines into code blocks. Based on the assumption that an attacker has to tamper multiple consecutive lines to make the prints defective, code block is proposed as the basic unit for tamper detection and recovery. Since G0 and G1 command lines contain most of the geometric information used in the printing, we group the G0 and G1 command lines into code blocks and each block contains n lines of code, denoted as $\{C_j | j = 1, 2, ..., \lfloor N/n \rfloor\}$, where N is the number of G0 and G1 command lines in the G-code file. Then, the code blocks are permuted by the function P with the random seed s, resulting in a new sequence of blocks $\{C_{P_s(j)} | j = 1, 2, ..., \lfloor N/n \rfloor\}$. We name the code block $C_{P_s(j+1)}$ as the mapping block of code block $C_{P_s(j)}$. Besides, the mapping block of the last code block in the permuted sequence is the first code block in the sequence. This implies a mapping of the code block set to itself, while avoiding any block mapped to itself, which is the basis of self-embedding watermarking.

2) Select the watermark embedding domain. Each code block is separated into two parts: Part I) the first command line, denoted as $C^1_{P_s(j)}$; Part II) the other $n-1$ command lines, denoted as $\{C^k_{P_s(j)} | k = 1, ..., n-1\}$, where $j = 1, 2, ..., \lfloor N/n \rfloor$. Meanwhile, X and Y coordinates in $C^1_{P_s(j)}$ are used to generate authentication and the recovery bits. X and Y coordinates in Part II of the block are utilized to carry the recovery bits.

3) Convert X and Y coordinates to binary representations. Given that X and Y coordinates in the G-code file are usually rounded to the precision of three decimal places, we shift all the digits of X and Y coordinates to the left three number places and obtain a decimal integer. Then, convert the decimal integer to a binary integer with the length of p bits, denoted as $x_{i,2} \in \mathbb{Z}_2^p$ and $y_{i,2} \in \mathbb{Z}_2^p$, where $i = 1, 2, ..., N$.

4) Embed the authentication bits into the code block itself. For Part I of $P_s(j)$-th code block, the binary representation of the X coordinate is denoted as $x^1_{P_s(j)} \in \mathbb{Z}_2^p$. The $p-6$ most significant bits of $x^1_{P_s(j)}$ are input to a Fowler-Noll-Vo (FNV) hash function to generate 6 authentication bits, $A^x_{P_s(j)}$. Then the 6 least significant bits of $x^1_{P_s(j)}$ are replaced by $A^x_{P_s(j)}$, obtaining $x'^1_{P_s(j)}$. The same process as described above is also applied to the Y coordinates.

5) Embed the recovery bits into the mapping block. The original bits of $x^1_{P_s(j)}$ and $y^1_{P_s(j)}$ before embedding the authentication bits are defined as the recovery bits for $P_s(j)$-th code block. The $m = \lceil \frac{p}{n-1} \rceil$ least significant bits of the binary X and Y coordinates $\{x^k_{P_s(j+1)}, y^k_{P_s(j+1)} | k = 1, ..., n-1\}$ in Part II of the mapping code block $C^1_{P_s(j+1)}$ are replaced by the recovery bits derived from the code block, $C^1_{P_s(j)}$. For example, when p=20, n=5, the $m = 5$ least significant bits of $\{x^k_{P_s(j+1)} | k = 1, ..., 4\}$ carry the 20 bits of $x^1_{P_s(j)}$. The recovery bits of the last code block in the permuted sequence are embedded into the first code block in the sequence.

6) After the embedding of authentication bits and recovery bits, the binary representations of X and Y coordinates are converted back to the decimal representation with the same precision.

C. Authentication and Recovery of G-code

When the 3D printer receives a G-code file, it checks whether the G-code has been tampered or not. If no tampering is detected, it restores the distortion caused by the embedding of authentication bits based on the recovery bits. If the G-code file is tampered, it can be partially recovered with the help of recovery bits and the geometric information from the neighboring code blocks.

In the authentication procedure, the first three steps are the same as those of the embedding procure. It first groups the command lines into code blocks and permutes them using the same method P_s. After obtaining the binary representations of the X and Y coordinates in each code block, it calculates the authentication bits of each code block, $\{\hat{A}^x_{P_s(j)}, \hat{A}^y_{P_s(j)} | j = 1, 2, ..., \lfloor N/n \rfloor\}$. If $\hat{A}^x_{P_s(j)}$ and $\hat{A}^y_{P_s(j)}$ are identical to the 6 least significant bits of $x^1_{P_s(j)}$ and $y^1_{P_s(j)}$, then the code block $C_{P_s(j)}$ is considered as genuine, otherwise the code block is identified as tampered. If none of the code blocks of the G-code file is tampered, the G-code file is authentic.

On one hand, if the G-code file is identified as authentic, Part I of each code block is restored by the corresponding recovery bits embedded in its mapping block. Namely, the original bits of $x^1_{P_s(j)}$ are obtained from the combination of the m least significant bits of $\{x^k_{P_s(j+1)}|k=1,...,n-1\}$.

On the other hand, if the G-code file is identified as tampered, the algorithm can partially recover the tampered lines. Two cases may emerge, according to the authentication results of all code blocks:

1) The code block $C_{P_s(j)}$ is tampered and its mapping block $C_{P_s(j+1)}$ is authentic. If so, Part I of $C_{P_s(j)}$ is fully recovered based on the recovery bits embedded in $C_{P_s(j+1)}$. Part II of $C_{P_s(j)}$ is estimated based on the geometric information presented in Part I of $C_{P_s(j)}$ and its subsequent code block in the original sequence, $C_{P_s(j)+1}$. The fully recovered X coordinates in Part I of $C_{P_s(j)}$ and $C_{P_s(j)+1}$ in decimal representation are denoted as $\{\hat{x}^1_{P_s(j)}\}$ and $\{\hat{x}^1_{P_s(j)+1}\}$. Then the X coordinates in Part II of $C_{P_s(j)}$ are calculated as,

$$\hat{x}^k_{P_s(j)} = \hat{x}^1_{P_s(j)} + k \times \frac{\hat{x}^1_{P_s(j)+1} - \hat{x}^1_{P_s(j)}}{n}, \qquad (1)$$

where $k = 1, 2, ..., n-1$.

2) Both $C_{P_s(j)}$ and $C_{P_s(j+1)}$ are identified as tampered. In this case, it is impossible to fully recover Part I or Part II of code block $C_{P_s(j)}$. Nevertheless, they can be estimated based on the geometric information from the code blocks before and after $C_{P_s(j)}$ in the original G-code file, namely $C_{P_s(j)-1}$ and $C_{P_s(j)+1}$. If X coordinates in Part I of $C_{P_s(j)-1}$ and $C_{P_s(j)+1}$ are fully recovered as $\hat{x}^1_{P_s(j)-1}$ and $\hat{x}^1_{P_s(j)+1}$, the X coordinates in $C_{P_s(j)}$ are estimated as

$$\hat{x}^k_{P_s(j)} = \frac{\hat{x}^1_{P_s(j)+1} - \hat{x}^1_{P_s(j)-1}}{2} + (k-1) \times \frac{\hat{x}^1_{P_s(j)+1} - \hat{x}^1_{P_s(j)-1}}{2n}, \qquad (2)$$

where $k = 1, 2, ..., n$.

The estimation of the Y coordinates in $C_{P_s(j)}$ is similar in all the cases mentioned above. This estimation approach is based on the observation that the X and Y coordinates in the neighboring command lines have strong linear correlations.

IV. Experimental Results

In this section we test the performance of the proposed watermarking method. We perform experiments testing the invisibility of the watermark, test the tamper detection accuracy and perform G-code file recovery.

Above all, we propose a novel measure named Averaged Relative Difference (ARD) for the aim of properly assessing the difference between two G-code files, denoted as $\mathcal{G}$ and $\mathcal{G}'$, which is calculated as,

$$ARD(\mathcal{G}, \mathcal{G}') = \frac{\frac{1}{N}\sum_{i=1}^{i=N}\sqrt{(x_i - x_i')^2 + (y_i - y_i')^2}}{\min\{IQR(\mathbf{x}), IQR(\mathbf{y})\}}, \qquad (3)$$

where x_i and y_i are the X and Y coordinates in the i-th command line of G-code file, $\mathcal{G}$, while x_i' and y_i' are those of $\mathcal{G}'$. $\mathbf{x}$ and $\mathbf{y}$ are the sets of the X and Y coordinates in $\mathcal{G}$. $IQR(\cdot)$ represents the interquartile range of the data.

This measurement reflects the average Euclidean distances between the geometric information in two G-code files while considering the whole size of the object. ARD is used for evaluating the distortion in the G-code file caused by the watermark embedding when $\mathcal{G}$ and $\mathcal{G}'$ represent the original and watermarked G-code files. We also consider ARD between the original and tampered G-code files. We also evaluate the recovery ratio using ARD when $\mathcal{G}$ and $\mathcal{G}'$ represent the original and restored G-code files, respectively. It should be noted that this measurement assumes that the two G-code files have the same number of command lines.

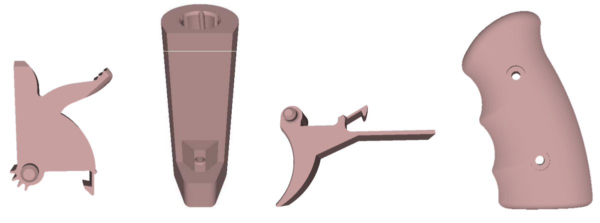

(a) Hammer (b) Barrel (c) Trigger (d) Grip

Fig. 4. The shapes of the 3D prints corresponding to the four G-code files used in the experiments.

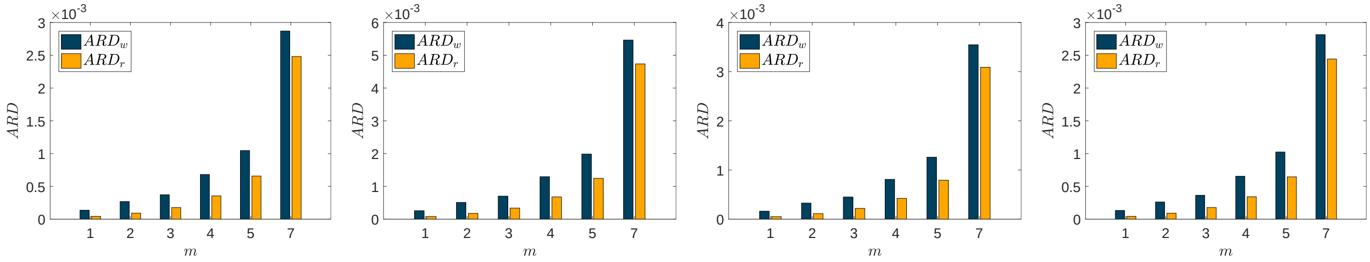
In the experiments, four G-code files for different 3D prints are used as the original G-code files. The shape of the four 3D prints, which are parts of a revolver model, are shown in Fig. 4. The G-code files are generated by Cura version 4.10.0, with some printing parameters, infill density 20%, layer height 0.16 mm, print speed 60 mm/s.

A. Invisibility Analysis

When analyzing the invisibility of the watermarking method, we apply the proposed self-embedding watermarking method on the four G-code files corresponding to the shapes from Fig. 4, setting the embedding parameters at different values to investigate their influence on the watermark embedding distortion.

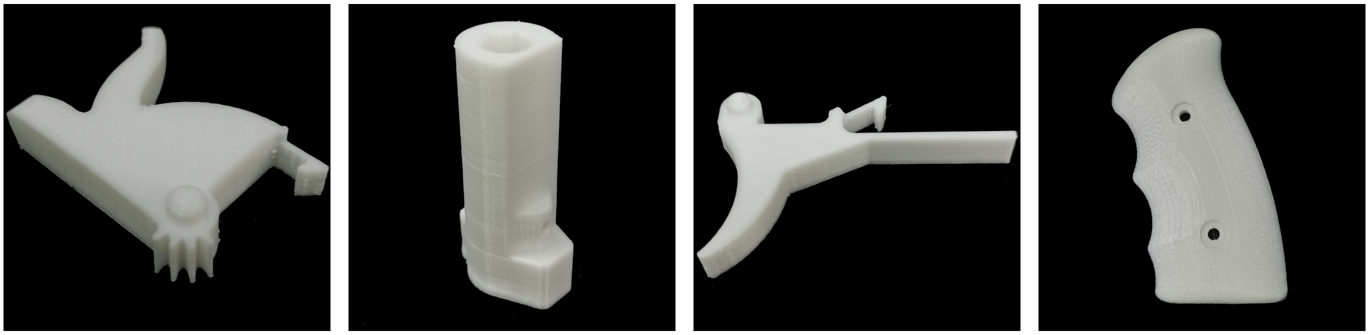
Theoretically analyzing the watermarking distortion, if the G-code file is authentic, the distortion caused by the embedding of authentication bits is reversible, because Part I of each code block can be fully recovered based on the recovery bits embedded in its mapping block. In consequence, the only distortion is caused by the embedding of the recovery bits in Part II of each block. In fact, these embedding modifications are limited to the $m = \lceil \frac{p}{n-1} \rceil$ least significant bits of the binary representation of X and Y coordinates, where n is the number of command lines in each code block and p is the length of the binary representation of X and Y coordinates. We set the embedding parameters $(p, n) \in \{(20, 21), (20, 11), (21, 8), (20, 6), (20, 5), (21, 4)\}$, so that $m \in \{1, 2, 3, 4, 5, 7\}$, when applying the proposed watermarking method on the four original G-code files.

The ARD between the original G-code file and the watermarked one is denoted as ARD_w and that of the original and the restored ones is denoted as ARD_r. The ARD_w and ARD_r of the four testing G-code files are shown in Fig. 5. It is shown that the distortion caused by the watermark embedding is very limited when $m \leqslant 5$. Meanwhile, it is shown that ARD_r is



(a) Hammer (b) Barrel (c) Trigger (d) Grip

Fig. 5. The ARD between the four original G-code files corresponding to the shapes shown in Fig. 4 and their watermarked ones, or restored ones, considering various embedding settings. ARD_w represents the ARD between the original G-code file and the watermarked one. ARD_r the represents that of the original and restored ones.



(a) Hammer (b) Barrel (c) Trigger (d) Grip

Fig. 6. The 3D prints obtained using the restored G-code files after watermark embedding.

less than ARD_w in every case which verifies that the recovery of Part I of each code block can further reduce the distortion.

In addition, the four 3D prints shown in Fig. 6 are created using the restored G-code file, after it had been watermarked, when $m = 4$. The model of the 3D printer is JGAURORA A8L whose precision can be as high as 0.05mm. It can be observed from Fig. 6 that the embedding distortion of the G-code files has very limited influence on the resulting 3D prints.

B. Tamper Detection

In order to test the tamper detection accuracy, we simulate the malicious tampering by shifting the X and Y coordinates in multiple command lines of the watermarked G-code file. We use the watermarked G-code file of the "Hammer" in Fig. 4(a) in the tamper detection experiment. The tamper ratio, T, is the parameter controlling the ratio of tampered command lines compared to the total command lines, which is set to $\{0.01, 0.02, 0.03, 0.04, 0.05\}$. The tampered command lines is a group of successive command lines randomly selected from the G-code file. The shifts of X and Y coordinates are fixed at $10\% \times IQR(\mathbf{x})$ and $10\% \times IQR(\mathbf{y})$ respectively, where $\mathbf{x}$ and $\mathbf{y}$ are the sets of the X and Y coordinates in the watermarked G-code file. The embedding parameters are set as $(p, n) \in \{(20, 11), (21, 8), (20, 6), (20, 5), (21, 4)\}$. The tamper detection error rate is the sum of the miss detection errors and false alarm errors compared to the total number of command lines.

The tamper detection results of the G-code file under different tampering levels are shown in Fig. 7. It can be observed from Fig. 7 that the detection error rate is lower than 6×10^{-4} and it even becomes much lower when n decreases. This is because the tampered part is detected block by block when using the proposed method, so a smaller code block size can benefit the detection accuracy. It is indicated in Fig. 7 that when $n \leqslant 6$, the proposed method achieves relatively stable tamper detection performance.

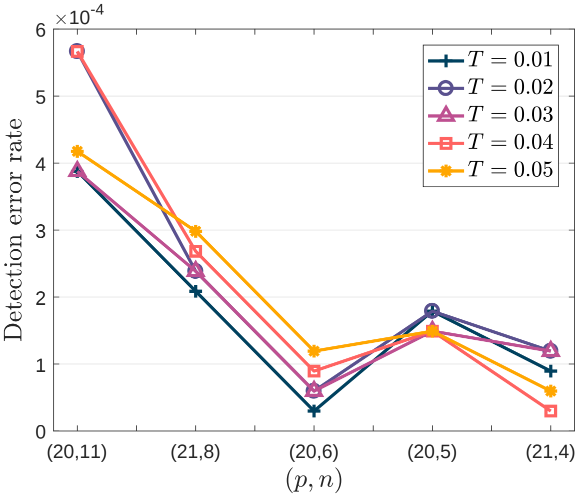
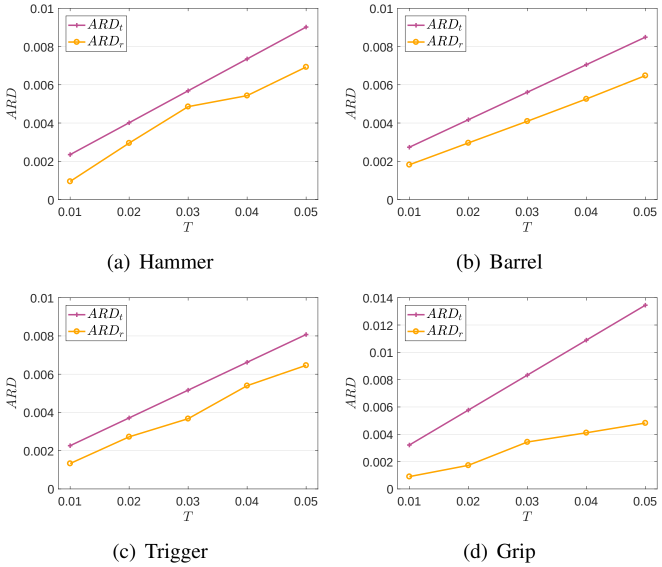


Fig. 7. Tamper detection error of the G-code files, watermarked by the proposed method with different settings when assuming various tamper ratios. T is the tamper ratio and (p, n) are the embedding parameters.

C. G-code Recovery

When testing the recovery ability of the proposed method, we utilize the four G-code files corresponding to the objects shown in Fig. 4 as the original files. The watermark embedding parameters are set to $(p, n) = (20, 6)$ which shows stable performance in tamper detection. The tampering process is the same as before with tamper ratio $T \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$. We calculate the ARD between the original G-code file and the tampered one, denoted as ARD_t, and that of the original and the restored ones, ARD_r.

The recovery results of the tampered G-code file, indicated by ARD_t and ARD_r, are shown in Fig. 8. It can be observed from Fig. 8 that the proposed method can significantly reduce the ARD caused by the tampering.



(a) Hammer

(b) Barrel

(c) Trigger

(d) Grip

Fig. 8. The ARD between the four original G-code files representing the objects shown in Fig. 4 and their tampered ones, or restored ones, considering various tamper ratios. ARD_t represents the ARD between the original G-code file and the tampered one. ARD_r represents that of the original and the restored ones.

V. Conclusion

With a focus on authenticating the G-code file used for 3D printing, a self-embedding watermarking method for G-code file is proposed. To the best of our knowledge, this is the first watermarking method proposed for the authentication of G-code file. The proposed method groups the G-code lines into code blocks and obtains a random mapping relationship for each code block. The authentication and recovery bits are embedded in the code blocks and their mapping blocks, respectively. The tampered parts are detected using the authentication bits in each code block. Meanwhile, the G-code files are restored based on the recovery bits as well as the geometric information of the neighboring code blocks. Experimental results indicate that the proposed method can effectively detect the tampered parts and restore the G-code file to a large extent, while limiting the distortion caused by the watermarking. However, the proposed method still has some limitations. For example, it cannot detect attacks that remove the command lines from the G-code file. A more comprehensive tamper detection and recovery ability is needed for improving G-code watermarking in the future.

References

[1] U. M. Dilberoglu, B. Gharehpapagh, U. Yaman, and M. Dolen, “The role of additive manufacturing in the era of industry 4.0,” *Procedia Manufacturing*, vol. 11, pp. 545–554, 2017.

[2] L. D. Sturm, C. B. Williams, J. A. Camelio, J. White, and R. Parker, “Cyber-physical vulnerabilities in additive manufacturing systems: A case study attack on the. stl file with human subjects,” *Journal of Manufacturing Systems*, vol. 44, pp. 154–164, 2017.

[3] R. Langner, “Stuxnet: Dissecting a cyberwarfare weapon,” *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[4] Z. Li and A. G. Bors, “Steganalysis of 3D objects using statistics of local feature sets,” *Information Sciences*, vol. 415-416, pp. 85–99, 2017.

[5] Y. Yang, R. Pintus, H. Rushmeier, and I. Ivrissimtzis, “A 3D steganalytic algorithm and steganalysis-resistant watermarking,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 2, pp. 1002–1013, Feb 2017.

[6] Z. Li, S. Beugnon, W. Puech, and A. G. Bors, “Rethinking the high capacity 3D steganography: Increasing its resistance to steganalysis,” in *Proceedings of 2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 510–414.

[7] X. Zhang, Q. Wang, and I. Ivrissimtzis, “Single image watermark retrieval from 3D printed surfaces via convolutional neural networks.” in *Proceedings of 2018 Computer Graphics & Visual Computing (CGVC)*. Eurographics Association, January 2018, pp. 117–120.

[8] Z. Li, D. Gong, F. Liu, and A. G. Bors, “3D steganalysis using the extended local feature set,” in *Proceedings of 2018 IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 1683–1687.

[9] S. Beugnon, W. Puech, and J.-P. Pedeboy, “Format-compliant selective secret 3-D object sharing scheme,” *IEEE Transactions on Multimedia*, vol. 21, no. 9, pp. 2171–2183, 2019.

[10] Z. Li and A. G. Bors, “Selection of robust and relevant features for 3-D steganalysis,” *IEEE Transactions on Cybernetics*, vol. 50, no. 5, pp. 1989–2001, 2020.

[11] ——, “Steganalysis of meshes based on 3D wavelet multiresolution analysis,” *Information Sciences*, vol. 522, pp. 164–179, 2020.

[12] Y. Gao, B. Li, W. Wang, W. Xu, C. Zhou, and Z. Jin, “Watching and safeguarding your 3D printer: Online process monitoring against cyber-physical attacks,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, pp. 1–27, 2018.

[13] M. H. Rais, Y. Li, and I. Ahmed, “Spatiotemporal G-code modeling for secure FDM-based 3D printing,” in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, 2021, pp. 177–186.

[14] X. Zhang, S. Wang, Z. Qian, and G. Feng, “Reference sharing mechanism for watermark self-embedding,” *IEEE Transactions on Image Processing*, vol. 20, no. 2, pp. 485–495, 2011.

[15] P. Korus and A. Dziech, “Efficient method for content reconstruction with self-embedding,” *IEEE Transactions on Image Processing*, vol. 22, no. 3, pp. 1134–1147, 2013.

[16] M. Fallahpour, S. Shirmohammadi, M. Semsarzadeh, and J. Zhao, “Tampering detection in compressed digital video using watermarking,” *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 5, pp. 1057–1072, 2014.

[17] C. Qin, P. Ji, C.-C. Chang, J. Dong, and X. Sun, “Non-uniform watermark sharing based on optimal iterative BTC for image tampering recovery,” *IEEE MultiMedia*, vol. 25, no. 3, pp. 36–48, 2018.

[18] F. Peng, Z.-X. Lin, X. Zhang, and M. Long, “A semi-fragile reversible watermarking for authenticating 2D engineering graphics based on improved region nesting,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 1, pp. 411–424, 2021.

[19] C.-M. Chou and D.-C. Tseng, “Affine-transformation-invariant public fragile watermarking for 3D model authentication,” *IEEE Computer Graphics and Applications*, vol. 29, no. 2, pp. 72–79, 2009.

[20] F. Peng, B. Long, and M. Longa, “A general region nesting based semi-fragile reversible watermarking for authenticating 3D mesh models,” *IEEE Transactions on Circuits and Systems for Video Technology*, 2021, DOI:10.1109/TCSVT.2021.3052468.