

TC260-PG-20257A

---

# 网络安全标准实践指南

——人工智能生成合成内容标识方法 文件  
元数据隐式标识 视频文件

---

(V1.0-202508)



全国网络安全标准化技术委员会秘书处

2025 年 08 月

本文档可从以下网址获得：

[www.tc260.org.cn/](http://www.tc260.org.cn/)



全国网络安全标准化技术委员会  
National Technical Committee 260 on Cybersecurity of SAC



## 前 言

《网络安全标准实践指南》（以下简称《实践指南》）是全国网络安全标准化技术委员会（以下简称“网安标委”）秘书处组织制定和发布的标准相关技术文件，旨在围绕网络安全法律法规政策、标准、网络安全热点和事件等主题，宣传网络安全相关标准及知识，提供标准化实践指引。

本文件起草单位：北京快手科技有限公司、北京抖音信息服务有限公司、中国电子技术标准化研究院、国家计算机网络应急技术处理协调中心、浙江大学、中央网信办数据与技术保障中心、阿里云计算有限公司、河北省网络安全和信息化技术中心、国家计算机网络应急技术处理协调中心山西分中心、北京世纪好未来教育科技有限公司、腾讯音乐娱乐（深圳）有限公司、北京金山办公软件股份有限公司、中国移动通信集团有限公司。

本文件主要起草人：耿星、苍鹏、张震、贺凯、杜蕾、郝春亮、孙勇、孔甜甜、落红卫、谷晨、王志伟、许晓耕、王丹丹、李一鸣、张立尧、程鹏、董琳、张谦、侯健玮、李茹、李伟峰、张树玲、杨敏、安红云、徐嵩、孙培尧、张秋芬。



## 声 明

本《实践指南》版权属于网安标委秘书处，未经秘书处书面授权，不得以任何方式抄袭、翻译《实践指南》的任何部分。凡转载或引用本《实践指南》的观点、数据，请注明“来源：全国网络安全标准化技术委员会秘书处”。



全国网络安全标准化技术委员会  
National Technical Committee 260 on Cybersecurity of SAC



## 摘 要

为落实《人工智能生成合成内容标识办法》，根据强制性国家标准GB 45438—2025《网络安全技术 人工智能生成合成内容标识方法》的要求，本文件提供了人工智能生成合成的视频文件元数据隐式标识方法，包括针对不同文件格式的具体方案，指导人工智能生成合成内容服务提供者和网络信息内容传播服务提供者开展人工智能生成合成的视频文件元数据隐式标识活动。





## 目 录

1 范围 .....	2
2 规范性引用文件 .....	2
3 术语和定义 .....	2
4 缩略语 .....	3
5 概述 .....	3
5.1 视频文件格式 .....	4
5.2 方法概述 .....	4
6 视频文件的元数据隐式标识方案 .....	4
6.1 原生嵌入方案 .....	4
6.2 XMP 方案 .....	6
附 录 A（资料性） MP4 视频文件的元数据方案参考示例 .....	8
附 录 B（资料性） FLV 视频文件的元数据方案参考示例 .....	9
附 录 C（资料性） MKV 视频文件的元数据方案参考示例 .....	10
附 录 D（资料性） AVI 视频文件的元数据方案参考示例 .....	12
参考文献 .....	15





## 1 范围

本文件给出了人工智能生成合成视频内容的文件元数据隐式标识方法，包括针对不同视频文件格式的具体方案。

本文件适用于指导人工智能生成合成视频内容服务提供者和网络信息视频内容传播服务提供者开展文件元数据隐式标识活动。

## 2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB 45438—2025 网络安全技术 人工智能生成合成内容标识方法

## 3 术语和定义

下列术语和定义适用于本文件。

### 3.1 人工智能生成合成视频内容 **video content generated by artificial intelligence**

利用人工智能技术生成、合成的视频信息。

### 3.2 人工智能生成合成视频内容服务提供者 **artificial intelligence video content generation service provider**

生成合成视频服务提供者



利用人工智能技术（包括通过提供可编程接口等方式）向公众提供生成合成视频服务的组织或个人。

### 3.5 网络信息视频内容传播服务提供者 internet information video content propagation service provider

#### 视频内容传播服务提供者

提供网络信息视频内容传播服务的网络信息服务提供者。

## 4 缩略语

下列缩略语适用于本文件。

AVI: 音频视频交错格式 (Audio Video Interleave)

AVS: 数字音视频编解码技术标准 (Audio Video coding Standard)

BMFF: 基础媒体文件格式 (Base Media File Format)

FLV: 动画视频 (Flash Video)

MKV: Matroska 多媒体容器 (Matroska Video)

MOV: 影片格式 (Movie Format)

MP4: MPEG-4 容器格式 (Moving Picture Experts Group 4 Part 14)

RDF: 资源描述框架 (Resource Description Framework)

RIFF: 资源交换文件格式 (Resource Interchange File Format)

URI: 统一资源标识符 (Uniform Resource Identifier)

XMP: 可扩展元数据平台 (Extensible metadata platform)

## 5 概述



## 5.1 视频文件格式

视频文件格式一般包括：

- a) 基于 BMFF 规范的视频文件，如 MP4、MOV；
- b) 基于 RIFF 规范的视频文件，如 AVI；
- c) 基于 Matroska 规范的视频文件，如 MKV、WebM；
- d) 基于 FLV 规范的视频文件，如 FLV。

## 5.2 方法概述

针对本文件 5.1 所列各类格式的视频文件，本文件给出了通过原生嵌入实现的元数据隐式标识方案；针对其他格式的视频文件，同时给出了通过 XMP 规范实现的元数据隐式标识方案。

**注：**原生嵌入是指利用特定文件格式预先定义的存储结构与访问接口将元数据写入文件内部预留区域的行为。

## 6 视频文件的元数据隐式标识方案

### 6.1 原生嵌入方案

#### 6.1.1 基于 BMFF 的视频文件

对于 MP4 格式和 MOV 格式的视频文件，具体方案如下：

- a) 存储位置为 moov.udta.meta。
- b) 写入方式为：
  - 1) 在 moov.udta.meta.keys 中写入 key: AIGC；
  - 2) 在 moov.udta.meta.ilst 中写入 value:

`{"Label":"value1","ContentProducer":"value2","ProduceID":"`





value3","ReservedCode1":"value4","ContentPropagator":"value5","PropagateID":"value6","ReservedCode2":"value7"}。

MP4 的实现示例参见附录 A。

### 6.1.2 基于 FLV 的视频文件

对于 FLV 格式的视频文件，具体方案如下：

- a) 存储位置为 script.onMetaData;
- b) 写入方式为：在 script.onMetaData 中写入 GB 45438—2025 附录 E b) 所定义的字符串："AIGC"：  
{"Label":"value1","ContentProducer":"value2","ProduceID":"value3","ReservedCode1":"value4","ContentPropagator":"value5","PropagateID":"value6","ReservedCode2":"value7"}。

FLV 的实现示例参见附录 B。

### 6.1.3 基于 Matroska 的视频文件

对于 MKV 格式和 WebM 格式的视频文件，具体方案如下：

- a) 存储位置为 root.Segment.Tags.Tag.SimpleTag;
- b) 写入方式为：
  - 1) 在 root.Segment.Tags.Tag.SimpleTag.TagName 中写入 AIGC;
  - 2) 在 root.Segment.Tags.Tag.SimpleTag.TagString 中写入：  
{"Label":"value1","ContentProducer":"value2","ProduceID":"value3","ReservedCode1":"value4","ContentPropagator":"value5","PropagateID":"value6","ReservedCode2":"value7"}。



ue5","PropagateID":"value6","ReservedCode2":"value7"}。

AVS 编码的视频采用 MKV 容器，实现示例参见附录 C。

#### 6.1.4 基于 RIFF 的视频文件

对于 AVI 格式的视频文件，具体方案如下：

- a) 存储位置为 RIFF 的 LIST/INFO 块(chunk);
- b) 写入方式为：
  - 1) 在 LIST/INFO 块的块数据中，创建自定义子块;
  - 2) 在 1) 创建的自定义块中，写入 ID 为 AIGC，Value 为  
{"Label":"value1","ContentProducer":"value2","ProduceID":"value3","ReservedCode1":"value4","ContentPropagator":"value5","PropagateID":"value6","ReservedCode2":"value7"}。

AVI 的实现示例参见附录 D。

#### 6.2 XMP 方案

对于其他视频格式，具体 XMP 方案如下：

- a) RDF 中新增自定义命名空间 TC260，URI 为 <http://www.tc260.org.cn/ns/AIGC/1.0/>。该命名空间已有其他内容时，应按属性更新，不应整体覆盖;
- b) 在 TC260 空间下 AIGC 键值中填入 GB 45438-2025 附录 E 规定字符串，且按照下文实现序列化：

<TC260:AIGC>{"Label":"value1","ContentProducer":"value2","



ProduceID":"value3","ReservedCode1":"value4","ContentPropagator":"value5","PropagateID":"value6","ReservedCode2":"value7"}</TC260:AI  
GC>。





## 附录 A

### (资料性)

## MP4 视频文件的元数据方案参考示例

以使用 ffmpeg 工具为例，写入示例如下：

```
ffmpeg -i input.mp4 \
-metadata
AIGC="{\"Label\":\"value1\",\"ContentProducer\":\"value2\",\"ProduceID\":\"value3\",\"ReservedCode1\":\"value4\",
\"ContentPropagator\":\"value5\",\"PropagateID\":\"value6\",\"ReservedCode2\":\"value7\"}" \
-movflags use_metadata_tags \
-c copy example.mp4
```



读取示例如下：

```
ffmpeg -i example.mp4
```

```
ffmpeg version 7.1.1 Copyright (c) 2000-2025 the FFmpeg developers
  built with Apple clang version 15.0.0 (clang-1500.3.9.4)
  configuration: --prefix=/opt/homebrew/Cellar/ffmpeg/7.1.1-with-options_1 --enable-shared --cc=clang --host-cflags= --host-ldflags=
--enable-gpl --enable-libaom --enable-libdav1d --enable-libharfbuzz --enable-libmp3lame --enable-libopus --enable-libsnap --enable-
e-libtheora --enable-libvorbis --enable-libvpx --enable-libx264 --enable-libx265 --enable-libfontconfig --enable-libfreetype --enabl
e-frei0r --enable-libass --enable-demuxer=dash --enable-neon --enable-openc1 --enable-audiotoolbox --enable-videotoolbox --disabl-h
tmlpages --enable-libfdk-aac --enable-nonfree
  libavutil      59. 39.100 / 59. 39.100
  libavcodec     61. 19.101 / 61. 19.101
  libavformat    61.  7.100 / 61.  7.100
  libavdevice    61.  3.100 / 61.  3.100
  libavfilter    10.  4.100 / 10.  4.100
  libswscale     8.  3.100 / 8.  3.100
  libswresample  5.  3.100 / 5.  3.100
  libpostproc   58.  3.100 / 58.  3.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'example.mp4':
  Metadata:
    minor_version      : 512
    major_brand        : isom
    compatible_brands   : isomiso2avc1mp41
    AIGC                : {\"Label\":\"value1\",\"ContentProducer\":\"value2\",\"ProduceID\":\"value3\",\"ReservedCode1\":\"value4\",
    :\"value5\",\"ContentPropagator\":\"value5\",\"PropagateID\":\"value6\",\"ReservedCode2\":\"value7\"}
  encoder             : Lavf61.7.100
  Duration: 00:00:03.04, start: 0.000000, bitrate: 356 kb/s
  Stream #0:0[0x1](und): Video: h264 (High) (avc1 / 0x31637661), yuv420p(progressive), 1280x720 [SAR 1:1 DAR 16:9], 208 kb/s, 60 fps
, 60 tbr, 90k tbn (default)
    Metadata:
      handler_name      : VideoHandler
      vendor_id         : [0][0][0][0]
  Stream #0:1[0x2](und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 127 kb/s (default)
    Metadata:
      handler_name      : SoundHandler
      vendor_id         : [0][0][0][0]
```



## 附录 B

### (资料性)

#### FLV 视频文件的元数据方案参考示例

以使用 ffmpeg 工具为例，写入示例如下：

```
ffmpeg -i input.flv \
-metadata
AIGC="{\"Label\":\"value1\",\"ContentProducer\":\"value2\",\"ProduceID\":\"value3\",\"ReservedCode1\":\"value4\",
\"ContentPropagator\":\"value5\",\"PropagateID\":\"value6\",\"ReservedCode2\":\"value7\"}"\
-c copy example.flv
```



读取示例如下：

```
ffmpeg -i example.flv
```



## 附录 C

(资料性)

### MKV 视频文件的元数据方案参考示例

以使用 ffmpeg 工具为例，写入示例如下：

```
ffmpeg -i input.mkv \
-metadata
AIGC="{\"Label\":\"value1\",\"ContentProducer\":\"value2\",\"ProduceID\":\"value3\",\"ReservedCode1\":\"value4\",
\"ContentPropagator\":\"value5\",\"PropagateID\":\"value6\",\"ReservedCode2\":\"value7\"}" \
-movflags use_metadata_tags \
-c copy example.mkv
```



读取示例如下：

```
ffmpeg -i example.mkv
```







```
PS D:\ffmpeg-V5.X-avs23-20220222_davs2-10bit-Win10-X64-EXE> .\ffmpeg.exe -i
ffmpeg version N-105467-g4a360c73fe-ffmpeg-windows-build-helpers Copyright (c) 2000-2022 the FFmpeg developers
built with gcc 10.2.0 (GCC)
configuration: --pkg-config=pkg-config --pkg-config-flags=--static --extra-version=ffmpeg-windows-build-helpers --enab
le-version3 --disable-debug --disable-w32threads --arch=x86_64 --target-os=mingw32 --cross-prefix=/home/hgfk/ffmpeg-wind
ows-build-helpers/sandbox/cross_compilers/mingw-w64-x86_64/bin/x86_64-w64-mingw32- --enable-libcaca --enable-gray --enab
le-libtesseract --enable-fontconfig --enable-gmp --enable-gnutls --enable-libass --enable-libbluray --enable-libbs2b --e
nable-libflite --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libilbc --enable-libmod
plug --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libopus --enable-libsnappy --ena
ble-libsoxr --enable-libspeex --enable-libtheora --enable-libtwolame --enable-libvo-amrwbenc --enable-libvorbis --enable
-libwebp --enable-libzimg --enable-libzvtbi --enable-libmysofa --enable-libopenjpeg --enable-libopenh264 --enable-liblens
fun --enable-librt --enable-libxml2 --enable-opengl --enable-libdav1d --enable-cuda-llvm --enable-libsrt --enable-lib
baom --enable-libvpx --enable-nvenc --enable-nvdec --extra-libs=lharfbuzz --extra-libs=lm --extra-libs=lthread --ext
ra-cflags=DLIBTWOLAME_STATIC --extra-cflags=DMODPLUG_STATIC --extra-cflags=DCACA_STATIC --enable-amf --enable-libmfx
--enable-gpl --enable-frei0r --enable-librubberband --enable-libvidstab --enable-libx264 --enable-libx265 --enable-avisy
nth --enable-libaribb24 --enable-libxvid --enable-libdav2 --enable-libuavs3d --enable-libxavs2 --enable-libuavs3e --ena
ble-libxavs --extra-cflags='-mtune=generic' --extra-cflags=-O3 --enable-shared --disable-static --prefix=/home/hgfk/ffmp
eg-windows-build-helpers/sandbox/win64/ffmpeg_git_shared
libavutil      57. 19.100 / 57. 19.100
libavcodec     59. 20.100 / 59. 20.100
libavformat    59. 17.101 / 59. 17.101
libavdevice    59.  5.100 / 59.  5.100
libavfilter    8. 26.101 /  8. 26.101
libswscale     6.  5.100 /  6.  5.100
libswresample  4.  4.100 /  4.  4.100
libpostproc   56.  4.100 / 56.  4.100
libuavs3d(10): 1.1.71_release, e58e199ccc50163c20df114db7e3950295c3e2ed
Input #0, matroska,webm, from '.\example.mkv':
  Metadata:
    AIGC      : {Label:value1,ContentProducer:value2,ProduceID:value3,ReservedCode1:value4,ContentPropagator:value
5,PropagateID:value6,ReservedCode2:value7}
    MINOR_VERSION : 1
    COMPATIBLE_BRANDS : iso4hvc1iso6
    MAJOR_BRAND : iso4
    ENCODER : Lavf59.17.101
  Duration: 00:00:04.02, start: 0.000000, bitrate: 578 kb/s
  Stream #0:0: Video: av3, yuv420p10le, 640x360, SAR 1:1 DAR 16:9, 29.97 fps, 29.97 tbr, 1k tbn (default)
    Metadata:
      HANDLER_NAME : hevc@GPAC0.5.2-DEV-rev565-g71748d7-ab-suite
      ENCODER : Lavc58.101.100 libuavs3e
      DURATION : 00:00:04.004000000
  Stream #0:1: Audio: aac (LC), 44100 Hz, stereo, fltp (default)
    Metadata:
      DURATION : 00:00:04.017000000
```



## 附录 D

### (资料性)

## AVI 视频文件的元数据方案参考示例

### D.1 代码开发参考示例

以下伪代码示例是基于 AVI 的视频文件元数据属性读写, 也给出了通过使用 `exiftool` 工具实现 AVI 视频文件元数据读取的示例。

```
function createInfoListChunk(fieldId, metaStr):
    # 将元数据字符串编码为 ASCII 并加上 '\0' 结尾
    content = metaStr + '\0'
    # RIFF 要求字段内容为偶数字节, 对齐填充
    if length(content) is odd:
        content += '\0'
    # 构造字段块: 字段 ID (4 字节) + 长度 (4 字节) + 内容 (变长)
    fieldChunk = pack(fieldId, length(content), content)
    # 构造 INFO 内容块: 'INFO' 标识 + 字段块
    listContent = 'INFO' + fieldChunk
    # 最终构造 LIST 区块: 'LIST' + 长度 + listContent
    return 'LIST' + length(listContent) + listContent

function updateRiffSize(filePath):
    open file in binary read/write mode
    move to end of file
    totalSize = file length
    riffSize = totalSize - 8 # RIFF header 的 size 字段不包含前 8 字节
    seek to byte offset 4
    write riffSize as 4-byte little endian

function insertInfoChunk(inputPath, outputPath, metaStr):
    # 读取原始 WAV 文件
    originalData = read input file
    # 构造新的 INFO 区块, 写入字段和数据
    infoChunk = createInfoListChunk('AIGC', metaStr)
    # 将原始数据和 infoChunk 写入新文件
    write originalData + infoChunk to output file
    # 更新 RIFF 块中的 size 字段
    updateRiffSize(outputPath)

function readInfoField(filePath):
    data = read all file bytes
    offset = 12 # 跳过 RIFF header ('RIFF' + size + 'WAVE')
    while offset < file size:
        read chunkId (4 bytes), chunkSize (4 bytes)
```





```
if chunkId == 'LIST' and content starts with 'INFO':
    subOffset = 4 # 跳过 'INFO'
    while subOffset < chunkSize:
        read fieldId (4 bytes), fieldSize (4 bytes)
        read fieldData of length fieldSize
        if fieldId == 'AIGC':
            return fieldData with trailing '\0' stripped and decoded as ASCII
        # 子字段也需要字节对齐 (偶数)
        if fieldSize is odd:
            subOffset += 1
        move subOffset to next field
    move offset to next chunk (8 + chunkSize)
    if chunkSize is odd:
        offset += 1 # 对齐 RIFF 主块
return " # 没找到目标字段

main:
# 从命令行获取输入文件、输出文件、元数据内容
parse inputPath, outputPath, metaStr from command line args
# 插入 metadata 块并保存到新文件
insertInfoChunk(inputPath, outputPath, metaStr)
# 读取并验证写入的字段值
print readInfoField(outputPath)
```

以使用 `exiftool` 工具为例，读取示例如下：

```
exiftool -v2 ./output.avi | grep -A10 AIGC
```

## D.2 工具开发参考示例

以使用 `ffmpeg` 工具为例，在 `riff.c` 文件以及 `riffenc.c` 从文件中分别新增一行代码，以加粗字体表示，示例如下：

```
diff --git a/libavformat/riff.c b/libavformat/riff.c
index 017a0658ef..82527f0d8f 100644
--- a/libavformat/riff.c
+++ b/libavformat/riff.c
@@ -644,6 +644,7 @@ const AVMetadataConv ff_riff_info_conv[] = {
     { "ISFT", "encoder" },
     { "ISMP", "timecode" },
     { "ITCH", "encoded_by" },
+    { "AIGC", "AIGC" },
     { 0 },
 };

diff --git a/libavformat/riffenc.c b/libavformat/riffenc.c
index 59c9932c36..ec841d5193 100644
--- a/libavformat/riffenc.c
+++ b/libavformat/riffenc.c
@@ -319,6 +319,7 @@ static const char riff_tags[][5] = {
     "IAS8", "IAS9", "ICMS", "ICMT", "ICOP", "ICRD", "ICRP", "IDIM", "IDPI",
     "IENG", "IGNR", "IKEY", "ILGT", "ILNG", "IMED", "INAM", "IPLT", "IPRD",
```



```
"IPRT", "ITRK", "ISBJ", "ISFT", "ISHP", "ISMP", "ISRC", "ISRF", "ITCH",  
+ "AIGC",  
  { 0 }  
};
```

写入元数据示例如下：

```
ffmpeg -i input.avi \  
-metadata  
AIGC='{"Label":"value1","ContentProducer":"value2","ProduceID":"value3","ReservedCode1":"  
value4","ContentPropagator":"value5","PropagateID":"value6","ReservedCode2":"value7"}'\  
-c copy example.avi
```

读取示例如下：

```
ffmpeg -i example.avi
```





## 参考文献

- [1] GB/T 33475.3—2018 信息技术 高效多媒体编码 第3部分：  
音频
- [2] ISO/IEC 14496-12 : Coding of audio visual objects Part 12 ISO  
base media file format
- [3] ISO IEC 14496-14: coding of audio visual objects Part 14 mp4  
file format
- [4] ISO 16684-1:2011 XMP specification Part 1: data model ,  
serialization, and core properties
- [5] ISO 16684-2:2014 XMP Part 2: Description of XMP schemas  
using RELAX NG
- [6] ISO 16684-3:2021 XMP specification Part 3: JSON-LD  
serialization of XMP
- [7] ISO 16684-4:2024 XMP specification Part 4: Use of XMP for  
semantic units