An Integrated Approach to Recovering Deleted Files from NAND Flash Data

James Luck & Mark Stokes

***Abstract*—Conventional techniques for recovering deleted files often prove useless in recovering files in general and video files in particular, from downloads of the raw memory data from mobile telephones (containing NAND flash memory). Several factors that are relied upon conventionally do not occur in mobile telephones. This paper presents an approach for recovering deleted files in general and video files in particular from NAND flash data files: starting with rebuilding the FAT partition, through recovering files from lost cluster chains and culminating in a methodology for enhanced extraction of deleted and corrupted video files by using the MPEG-4 meta data. Examples of successful video file extractions are given and the advantages illustrated. The structure of FAT volumes and MPEG-4/3gp video files as implemented on mobile telephones is also described.**

***Index Terms*—MPEG-4, mp4, 3gpp, 3gp, FAT rebuild, corrupted video, forensic digital, data recovery**

I. Introduction

TECHNIQUES for the recovery of deleted files from magnetic media are well established [1], but those for the recovery of deleted files from mobile telephone handsets (hereafter, ”handsets”) are much less so. Many handsets use variants of the FAT file system [2], [3], originally created by Microsoft for the IBM PC, to maintain media files such as pictures and video clips in NAND flash memory. The differences between the implementations on a handset and on a PC make the recovery of deleted files from the handset more difficult. In particular, the starting cluster (SC) in the directory entry may be overwritten upon deletion and there may be multiple versions of sectors with the same Logical Sector Number (LSN). In addition, in NAND flash, file sectors may be deliberately distributed throughout the physical memory and their LSNs may not be continuous. The purpose of this paper is to present an approach for overcoming these difficulties in recovering deleted files from handsets and present a methodology specifically for recovering deleted video files. Video files often occupy hundreds, even thousands, of sectors in memory. If any of the sectors have been erased (reset to 0xFF) or if a single sector is placed in the extracted file erroneously then the recovered video may fail to play at all; this is particularly significant in recovering deleted files. The MPEG-4 video file format separates the media data (video and audio tracks, for example) from the meta data (data that instructs the codec on how to interpret the media data). The meta data contains tables that give the offsets and the type of all the video and audio samples within the media data. If the, *small*, meta data part can be extracted first then it can be used to guide the extraction of the, *much larger*, media data part leading to a vastly improved chance of a successful extraction. Further, as erroneous sectors can be identified and replaced with null sectors (0x00), incomplete video files can still be played on readily available video playback software (e.g. Apple QuickTime 7). We have called this methodology, ”Xtractor”. The three major stages in the approach are: (i) Rebuild the FAT, where appropriate, and extract extant files [1] (Sec. 3), (ii) recover any lost clusters and associated files (Sec. 3.B.6.), (iii) use Xtractor for enhanced video recovery (Sec. 5). Xtractor can also be used independently. The structure of a FAT volume is explained in Sec. 2 and that of an MPEG-4 file in Sec. 4. In this paper hexadecimal numbers are denoted with the prefix ”0x”, binary numbers with ”0b”. All un-specified numbers are decimal with the exception of the data in the example figures, which are hexadecimal or binary, as applicable, with decimal offsets. The binary file of the raw memory data downloaded from the handset memory will be called the Source File. The term ”sector” shall refer to a physical sector in the memory chip or in the Source File. The term ”page” shall refer to the data of a sector in a media file. A sector size of 512 bytes will be used throughout. The offset from the beginning of a file will be termed the ”offset”, whereas ”Page Offset” will be the offset from the beginning of a page. Sectors also have associated meta data that provides information about the sector; this is often called the ”Spare Area” data. Sectors and associated Spare Area have been assigned a notional sequential number, starting from 1, called its Master Index (MI) value. Each sector is thus identified uniquely in the Source File and can be accessed directly from its MI value.

Here we have the typical use of a ”T” for an initial drop letter and ”HIS” in caps to complete the first word. You must have at least 2 lines in the paragraph with the drop letter(should never be an issue)

II. The FAT Volume in Handsets

A brief overview of the File Allocation Table (FAT) file system will be given here along with a description of the difficulties encountered in recovering deleted files from handsets that have FAT-based file systems.

A. The FAT Structure

A map of the FAT volume from a Nokia 6230 handset (software version: 5.24, NAND flash chip: Samsung KEE00E00CM) is shown in Fig. 1. There was no Master Boot Record (MBR) in this example and the first sector was the Volume Boot Record (VBR), occupying 1 sector. As handset memories often consist of only a single volume, the MBR

[1] Files that have not been deleted.

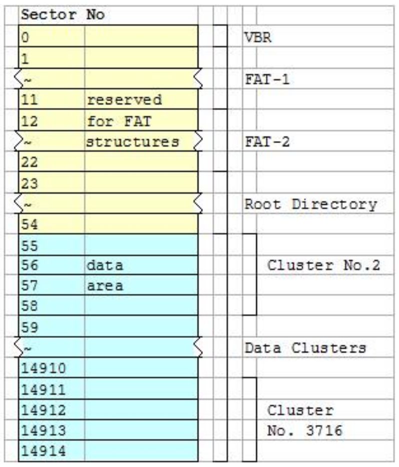
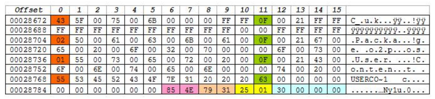


Fig. 1. FAT volume from a Nokia 6230 handset. Predefined sectors (yellow): VBR, FAT-1, FAT-2, root directory; data sectors (blue): 4 sectors per cluster

may or may not exist. The VBR was followed by FAT-1 and FAT-2 (11 sectors each) and then by the root directory (32 sectors). The data area extended from sector numbers 55 to 14914 (cluster numbers 2 to 3716) with the first 460 sectors occupied by sub-directories. In handset memories it is unlikely that the data from an individual file will reside in contiguous clusters.

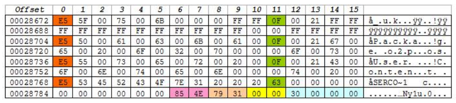
Each file will have a directory entry with the *starting-cluster* (SC) of the file (see Fig. 2) and, separately, there will be a list of the clusters used by the file, the cluster chain . [2] Fig. 2 shows an extant directory entry in FAT format from a handset. The handset supports long filenames (LFN) so there is a DOS block (the last 32 bytes) and a number of 32-byte LFN blocks. The number of LFN blocks that are occupied by a single entry depends on the length of the LFN and is given by the size byte of the terminal block (the initial byte of the top LFN block, in this case 0x43 means three blocks) and the order of the blocks is given by the *ordinate* field (the initial byte of the middle two LFN blocks). In this case the order is: DOS block - ”0x01” - ”0x02” - terminal block. The LFN can be seen to be ”User Content Package o2pos_uk” and the DOS or Short File Name (SFN) can be seen to be ”USERCO˜1”. The *attributes* byte (offset 11 in each block) is used to discriminate between LFN and DOS blocks: LFN blocks have attributes that could not occur for a valid DOS entry. *Attributes* is variable for the DOS block (0x63 here) and is 0x0F for the LFN blocks in this case. The SC (offsets 26, 27 in the DOS block) is 0x0125. Also shown in Fig. 2, from the DOS block are: the *modification-time*, 0x4E85 (offsets 22 & 23); *modification-date*, 0x3179 (offsets 24 & 25) and *file-size*, 0x00000030 (offsets 28-31). These fields follow the standard DOS format [2], [3]]. The *creation-time* & *creation-date* and the *accessed-time* & *accessed-date* were not used by this handset.

[2]The cluster chain is contained within the File Allocation Table, from which the file system gets its name.



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00028672	43	5F	00	75	00	6B	00	00	00	FF	FF	0F	00	21	FF	FF	C_.u.k...ÿÿ..!ÿÿ
00028688	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿ..ÿÿÿÿ
00028704	02	50	00	61	00	63	00	6B	00	61	00	0F	00	21	67	00	.P.a.c.k.a...!g.
00028720	65	00	20	00	6F	00	32	00	70	00	00	00	6F	00	73	00	e. .o.2.p...o.s.
00028736	01	55	00	73	00	65	00	72	00	20	00	0F	00	21	43	00	.U.s.e.r. ...!C.
00028752	6F	00	6E	00	74	00	65	00	6E	00	00	00	74	00	20	00	o.n.t.e.n...t. .
00028768	55	53	45	52	43	4F	7E	31	20	20	20	63	00	00	00	00	USERCO~1 c....
00028784	00	00	00	00	00	00	85	4E	79	31	25	01	30	00	00	00	......…Ny1u.0...

Fig. 2. Directory entry (Nokia 6230) with ”initial” bytes (orange), *attributes* (green), SC (yellow), *modification-time* (pink), *modification-date* (brown) and *file-size* (blue): Before deletion



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00028672	E5	5F	00	75	00	6B	00	00	00	FF	FF	0F	00	21	FF	FF	å_.u.k...ÿÿ..!ÿÿ
00028688	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	00	00	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿ..ÿÿÿÿ
00028704	E5	50	00	61	00	63	00	6B	00	61	00	0F	00	21	67	00	åP.a.c.k.a...!g.
00028720	65	00	20	00	6F	00	32	00	70	00	00	00	6F	00	73	00	e. .o.2.p...o.s.
00028736	E5	55	00	73	00	65	00	72	00	20	00	0F	00	21	43	00	åU.s.e.r. ...!C.
00028752	6F	00	6E	00	74	00	65	00	6E	00	00	00	74	00	20	00	o.n.t.e.n...t. .
00028768	E5	53	45	52	43	4F	7E	31	20	20	20	63	00	00	00	00	åSERCO~1 c....
00028784	00	00	00	00	00	00	85	4E	79	31	00	00	30	00	00	00	......…Ny1u.0...

Fig. 3. Directory entry (Nokia 6230) with ”initial” bytes (orange), *attributes* (green), SC (yellow), *modification-time* (pink), *modification-date* (brown) and *file-size* (blue): After deletion, with overwritten initial byte and SC

B. Complications Inherent in Handset FAT File Systems

In the FAT file systems implemented on handsets, there are frequently several factors that complicate the extraction of deleted files.

C. Deleted Directory Entries

If the entry in Fig. 2 were to be deleted (as in Fig. 3), the initial byte in every block making up the directory entry would be overwritten with 0xE5. In this 4-block entry the *file-size*, both *ordinates* and the first character of the SFN have all been lost. Because characters of the LFN itself are not overwritten the full LFN can be recovered. In magnetic media it is normal for the SC to remain intact. The SC in this handset has been overwritten with 0x0000 by the handset software severing the link to the cluster chain in the FAT. This precludes recovery by simple un-deletion. The *modification-time*, *modification-date* and *file-size* are un-affected by the file deletion.

Sectors are also marked in the Spare Area as deleted. Immediately after deletion the file structures and the file data still exist in memory, however all are then vulnerable to being erased (reset to 0xFF) and subsequently overwritten by new files. Adding new directory entries during normal usage leads to the creation of multiple versions of sectors containing old directory entries (see Sec. 2.B.4), as these versions have not been deleted by the user they may contain the information lost from the deleted entries described above. Recovery of these sectors, where possible, can therefore be beneficial.

D. Non-Contiguous Physical Sectors

One attempt to circumvent the loss of the SC would be to extract a sequence of contiguous physical sectors starting from an identifiable file header (e.g. a sector containing an ASCII ”ftyp” for a MPEG-4 video). In the handset memory the sectors of a single file are often distributed throughout the physical memory, for the purpose of wear-levelling the memory chip. Also there may be multiple versions of the same logical sector (see Sec. 2.B.4) interleaved with the required sectors. The file, therefore, does not exist in contiguous physical sectors defeating conventional file carving attempts.

E. Non-Continuous LSN Sequences

When a handset has been in use for some time it is very likely to have had several files stored in memory, some files deleted and new files stored, with the process repeated many times. The result can be a number of interleaved sequences of LSNs with the LSNs from one file being interleaved with those of another. An attempt to extract a file by following its sequence of LSNs may result in sectors from other files being included. This could prevent genuine sectors of a jpg file that occurred later in the file from being viewed and may prevent a video file from being played at all.

F. Multiple Versions

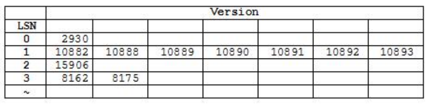
NAND flash memory obtains a speed enhancement by erasing (i.e. returning bits to state ”1”) entire blocks of memory in a single event. Consequently, to change even a single bit from state ”0” to state ”1” the entire ”erase-block” has to be erased resetting every bit in the erase-block to state ”1”. Hence all the bytes in ”Free” sectors in flash memory are characteristically ”0xFF”. During the normal creation of a file, as the data is being written to flash memory, if a sector needs to be modified the whole sector has to be copied to a new sector with the revision [3] . The ”old” sector should then be marked as deleted. There should, therefore, be one ”valid” sector and one or more ”deleted” counterparts, with all versions of a sector having the same LSN. In practice, all of the ”old” sectors might not be deleted, leaving several ”valid” versions. In the recovery of extant files this may be problematic. For files deleted by the user this may actually be useful in identifying the desired sector. Once a file has been deleted all of its sectors are marked as deleted. As many of the ”old” versions are marked as deleted also, sector status alone cannot then be relied upon to identify the desired version. We have observed that when a sector is copied as part of the updating process it is typically copied to a higher memory address, remaining in the same block and often copied by just one or a few physical sectors. The version that occurs at the higher memory address is, thus, the most likely to contain the latest data, as reported by Breeuwsma et al. [4]. These observations led to the creation of ”Version Tables” (see Sections 3.B.1 & 5.B.3) to assist in the selection of a particular version of a logical sector to use in file reconstruction.

III. Rebuilding the FAT Volume

Rebuilding a FAT volume from the Source File involves placing the logical sectors in the position of the physical sectors with the same respective sector number [4]. An example was given in Fig. 1. To rebuild the FAT volume the LSN from the Spare Area is required. The existence of multiple sector versions will have to be accommodated and can be to the advantage of the forensic examiner as earlier versions of the FAT volume may be retrievable.

[3]As sectors have to be written in their entirety the remaining bytes of an incomplete sector will be 0xFF. When the sector is completed the existing data will be copied to a new sector with the remainder of the data.

[4]When the logical sectors have been placed in their correct physical location the LSN and physical sector number are the same.



	Version						
LSN							
0	2930						
1	10882	10888	10889	10890	10891	10892	10893
2	15906						
3	8162	8175					
~							

Fig. 4. VT for FAT reconstruction: MI values for different versions of each logical sector

A. Multiple Versions

Our process consists of six major steps: (i) Build Version Table, (ii) rebuild FAT volume, (iii) analyse VBR, (iv) extract directory, (v) extract extant files, (vi) recover lost chains and lost files. These individual procedures will now be explained:

B. Rebuilding Process

1) Build Version Table: In the Source File, there can exist multiple sectors with the same LSN. To assist with the rebuilding process a table is made of MI values of each sector with the same LSN. This will be called a Version Table (VT), see Fig. 4. The MI values in the table are in the order in which they are located in the Source File: entries to the left are at the lower memory address and those to the right are at the higher memory address. It can be seen that there was only one version of the sector with LSN = 0, seven with LSN = 1 etc.

2) Rebuild FAT Volume: The process of rebuilding a FAT volume consists of selecting, by its MI value, from the VT, a version of a sector for each LSN. If a LSN does not exist in the Source File it is replaced with a null page in the FAT volume to maintain the correct offsets. By choosing different versions of the sector, different versions of the FAT volume can be constructed e.g. that which consisted of sectors which existed at the highest or lowest memory address. Depending on the Spare Area data available for the sector, it may be possible to construct FAT versions that consist of chronologically earliest or latest versions, or are made from deleted or extant sectors. In our implementation a manual choice option has been provided so that individual sectors can be chosen. Full logging of the extraction has been included for forensic reporting.

3) Analyse VBR: We have found that the VBR in a handset closely follows the DOS standard [2], [3]. From the VBR the sector size, the number of FATs, the number of sectors per FAT, the number of sectors per cluster, the size of the VBR, the size of the Root Directory and the total number of sectors can be obtained. From this information the map of the volume shown in Fig. 1 was constructed. Note that the next sector following the root directory is the first sector of the first data cluster; this data cluster is cluster number two, and the sector numbering in the FAT volume starts from zero. The relationship between cluster number in the FAT and the physical cluster is thus determined and can be used to extract files according to the cluster chain.

4) Extract Directory: From the rebuilt FAT volume the directory entries can be read. In the case of extant entries this is straightforward and can be achieved either recursively or, if the extent of the directory entries is pre-discovered, linearly. Note that the sub-directories occupy part of the data area.

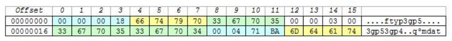
The most interesting directory entries are, usually, the deleted entries. The number of 32-byte blocks that are occupied by a single deleted entry depends on the length of the LFN; in the undeleted case this was given by the *file-size* and the order of the blocks was given by the *ordinate* field (see Fig. 2). It can be seen from Fig. 3 that in the deleted case both the *file-size* and the *ordinate* field have been lost. If there are several deleted LFN entries abutting one another then it will be necessary to discriminate between the separate directory entries. From our experience the order of the LFN blocks and the DOS block is invariant (the ordinate field is, therefore, redundant). Reading upwards; the DOS block is first, followed by the LFN blocks in their correct order. It is thus only necessary to discriminate between LFN and DOS blocks to discriminate between the separate directory entries; this is done via the *attributes* byte. As the definitive ordering has been lost, care must be taken to check the handset under examination and when reporting such results in evidence.

5) Recovering Extant Files: The extant files indexed in the directory have their SCs intact and can easily be extracted by following their cluster chains. Any application that can mount a FAT volume will be able to read the extant files and probably the deleted directory entries also (but not the deleted files).

6) Recovering Lost Chains and Deleted Files: For files that have been deleted, where the SC has been overwritten, the link between the directory entry and the cluster chain will have been lost. The cluster chain, however, might still exist in memory and if it can be recovered the file may still be recovered from it. The File Allocation Table itself is just a numbered list of numbers (irrespective of the number of bytes used per entry, i.e. whether it is FAT 12 or FAT 16). An entry in the FAT refers to both the number of a cluster in the file and the number of the next entry in the list. For example, if entry number 6 is the number 7 then cluster 6 is in the file and entry 7 says what to do next. If entry 7 is another valid number then cluster 7 is in the file and the number denoted by entry 7 continues the sequence until the entry is the "end" value (e.g. 0xFFF for FAT 12, 0xFFFF for FAT 16). This is how the handset normally accesses the files with the first number in the chain being the SC from the directory entry. It is not known, a priori, where every chain starts nor is it known, in reconstruction, whether a particular chain is complete[5] . To ensure that all chains that exist in the FAT, including all lost chains and partial fragments of lost chains, are extracted it is necessary to follow every chain starting from every entry in the FAT. For example if the chain 6-7-8-end exists then 6-7-8-end, 7-8-end & 8-end will be found in the FAT (the extracted chains can then be examined to remove the sub-chains). Note that unused entries in the FAT may effectively reference Cluster 0 and must be terminated.

IV. MPEG-4 & 3GP FILE FORMAT

Information contained within MPEG-4 & 3gp files may be useful to a forensic examiner attempting to reconstruct deleted video files. Also, the structure of the video file forms a vital part of the extraction methodology described in Sec. 5, so a description of the file format, as applicable to handset video, will be given in this section. The 3gp example data are from a Nokia 7600 and the MPEG-4 example data are from a Samsung D600. The file formats of MPEG-4 and 3gp video files, commonly found on handsets, are defined by 3GPP [5] and largely conform to the MPEG-4 standard: ISO/IEC 14496 pt.12 [6] and pt.14 [7]. See also, Pereira & Ebrahimi [8] for a good description. MPEG-4 files usually have the file extension "mp4", and 3gp files, "3gp". MPEG-4 files are made up of a number of units called "atoms". Atoms can contain information or other atoms. Three atoms exist at the uppermost level of the hierarchy: the "file-type" atom, `ftyp`, identifies the file as a 3gp or MPEG-4 video file and serves as a file header; the "media-data" atom, `mdat`, contains the digital encoding of the video and audio content; and the "movie-data" atom, `moov`, contains meta data, which describes the content. Video and audio quanta are called samples[6] . The `moov` atom contains nested sub-atoms; some of these sub-atoms contain tables of the offsets within the file of the video and audio samples. A collection of video or audio samples is called a "chunk". An `mdat` atom may consist of any number of alternating audio and video chunks or runs of several audio or video chunks or any combination thereof.



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	00	00	00	18	66	74	79	70	33	67	70	35	00	00	03	00	....ftyp3gp5....
00000016	33	67	70	35	33	67	70	34	00	04	71	BA	6D	64	61	74	3gp53gp4..q°mdat

Fig. 5. 3gp file header: *size* (blue), `ftyp` and `mdat` markers (yellow), and brands (green)

A. Top-level Atoms

An example of a video header is given in Fig. 5 and will serve to describe some of the salient features of the atoms' structure. An atom has a 4-byte name (chosen to be visible as ASCII text), in this case: "ftyp". Preceding the name is the 4-byte atom size: 0x00000018 = 24 bytes. The `ftyp` atom contains "brands" which identify a specification to which the file conforms [5]. Examples include "3gp5" for 3gp release 5 and "isom" for MPEG-4. The brand, however, is not definitive and the `stsd` atom (Sec. 4.C.2) should be consulted for codec specification. Also shown in Fig. 5 is the header of the `mdat` atom, with `mdat` size: 0x000471BA = 291,258 bytes. An example of a `moov` atom[7] header in given in Fig. 6, with `moov` size: 0x1560 = 5,472 bytes (omitting superfluous leading zeros). The sum of these three atom sizes gives the size of the complete video file: file size = 296,754 bytes.

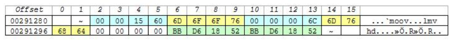
B. Identifiable Headers Within an `mdat` Atom

Individual video and audio samples within an `mdat` atom start with identifiable headers. By maximising the number of

[5]The chain might have been partially erased or part of it might have existed on another sector that has not been included in the specific FAT volume reconstructed.

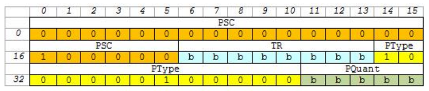
[6]MPEG-4 samples are not instantaneous values of a waveform; rather they are encoded frames of video or audio.

[7]Some video clips have the `moov` atom and mdat atom interchanged compared to this example; the principle of the extraction methodology described in Sec. 5 is unaffected.



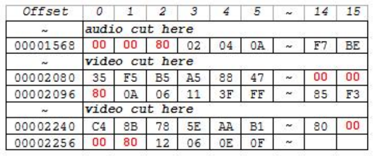
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00291280		~	00	00	15	60	6D	6F	6F	76	00	00	00	6C	6D	76	...`moov...lmv
00291296	68	64	00	00	00	00	BB	D6	18	52	BB	D6	18	52	~		hd....»Ö.R»Ö.R..

Fig. 6. 3gp `moov` atom header: *size* (blue), `moov` and `mvhd` markers (yellow), and time stamps (see Sec. 4.4)(green)



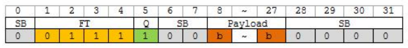
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	PSC															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	PSC						TR								PType	
16	1	0	0	0	0	0	b	b	b	b	b	b	b	b	1	0
	PType											PQuant				
32	0	0	0	0	0	1	0	0	0	0	0	b	b	b	b	b

Fig. 7. Header from H.263 video frame, ”b” is an arbitrary bit



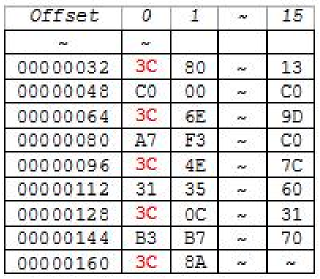
Offset	0	1	2	3	4	5	~	14	15
~	audio cut here								
00001568	00	00	80	02	04	0A	~	F7	BE
~	video cut here								
00002080	35	F5	B5	A5	88	47	~	00	00
00002096	80	0A	06	11	3F	FF	~	85	F3
~	video cut here								
00002240	C4	8B	78	5E	AA	B1	~	80	00
00002256	00	80	12	06	0E	0F	~		

Fig. 8. H.263 video data showing tags: 0x000080 (red font)



0	1	2	3	4	5	6	7	8	~	27	28	29	30	31
SB	FT				Q	SB		Payload			SB			
0	0	1	1	1	1	0	0	b	~	b	0	0	0	0

Fig. 9. AMR audio frame, ”b” is an arbitrary bit



Offset	0	1	~	15
~	~			
00000032	3C	80	~	13
00000048	C0	00	~	C0
00000064	3C	6E	~	9D
00000080	A7	F3	~	C0
00000096	3C	4E	~	7C
00000112	31	35	~	60
00000128	3C	0C	~	31
00000144	B3	B7	~	70
00000160	3C	8A	~	~

Fig. 10. AMR audio data showing tags, 0x3C (red font)

predicted bits in the header the maximum discrimination of video and audio samples in the Source File can be achieved. In each header some bits will be fixed, e.g. start codes, some will be parameters, e.g. frame dimensions, and others will vary from sample to sample. The headers depend upon the video and audio coding schemes and the specific parameters within those schemes. Several different schemes are available for use in MPEG-4 and 3gp video [5]. 3gp with ITU H.263 video [9] and AMR (Adaptive Multi-Rate) audio [5] is the most common format recorded by handsets. Some manufacturers, notably Samsung, predominantly use mp4 with MPEG-4 video [11] and AAC (Advanced Audio Coding) audio [12].

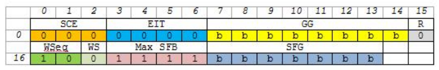
1) H.263 Video: The frame is byte aligned and has the following fields [9], see Fig. 7 : *Picture Start Code* (PSC), 22 bits; *Temporal Reference* (TR), 8 bits; *Picture Type* (PType), 13 bits; *PQuant*, 5 bits. The PSC is a 22-bit fixed code consisting entirely of 0s except for one 1 at bit offset 16.

The TR is in effect a modulo-256 index of the stored frames, however as not all frames are stored this index can typically increment by 2 or 4 between adjacent stored frames. In addition some frames may not be included resulting in an incomplete TR sequence. This makes it difficult to use the TR to identify frames from a deleted file as it is not known a priori what the exact TR sequence should be. PType is a collection of bits and flags[8] , most of which remain constant from frame to frame; the most important to this work are as follows. Bits at offsets 30 and 31 are always ”1” and ”0” respectively to prevent code emulation. Bits at offsets 32-34 and 39-42 have been observed to be invariant and equal to 0. Bits at offsets 35-37 are the *Source-Format*: 0b001 means sub-QCIF (128 96 pixels) this can be predicted from the `s263` atom in Sec. 4.C.2. The bit at offset 38 is the *Picture-Coding-Type*: 0 for an I-frame (intra-coded) and 1 for a P-frame (inter-coded). The first sample would be expected to be an I-frame and subsequent samples P-frames (see Sec. 4.E). *PQuant* (offsets 43-47) indicates the quantizer to be used [9]; this is likely to change from frame to frame. Thus the number of predictable bits in the frame header is 34 (35 including *Picture-Coding-Type*). The two most significant bits of the TR (offsets 22, 23) are the two least significant bits of the least significant byte holding the PSC. This causes the video frames to appear to have one of the four characteristic tags 0x000080 - 83. Fig. 8 shows 0x000080 tags in a video file at offsets: 1,568, 2,094, and 2,255.

2) AMR Audio: The AMR codec can be configured to operate in a number of different modes, which determines the bit rate and the number of bits in an AMR frame and hence the number of bits in an audio sample. The structure of an AMR frame [10] is shown in Fig. 9. The frame is byte aligned and has the following fields: *frame-type* (FT), 4 bits; *Frame-Quality* (Q), 1 bit; audio payload data. Stuffing bits (SB) are inserted in the header and at the end of the frame to achieve an integer number of bytes per frame. FT gives the mode, 0b0111 = 7, corresponding to 12.2kbit/s with 244 audio bits per frame [13]. The mode is given independently by the *mode-set* parameter in the damr atom (Sec. 4.C.2). Four stuffing bits are required at the end of the frame to maintain byte-alignment; making the frame size 32 bytes. We have seen that Q = 1 for video shot and stored on the same handset. Thus there are 8 bits that can be predicted and the characteristic 0x3C tag can be seen in Fig. 10. Eight bits are generally insufficient to form a reliable tag; however, for non-initial frames in a chunk the tag can be augmented by the four stuffing bits (0b0000) preceding the AMR header and the regular frame pitch can be exploited.

The `damr` atom (Sec. 4.C.2) gives the number of frames per sample as four. The size of an audio sample is, therefore, 128 bytes and audio samples can be seen to be at offsets: 32, 160, etc. in Fig. 10.

[8]A typical example of PType parameters encountered on a handset is given here.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	SCE			EIT				GG								R
0	0	0	0	0	0	0	0	b	b	b	b	b	b	b	b	0
	WSeq		WS	Max SFB				SFG								
16	1	0	0	1	1	1	1	b	b	b	b	b	b	b		

Fig. 11. Header from AAC audio frame

3) MPEG-4 Video: The tag for an MPEG-4 Video Object Plane (VOP) [9] consists of the *Start-Code-Prefix*: 0x000001, followed by the VOP-Start-Code = 0xB6 [11], after van der Knijff [14]. The next 2 bits give the *VOP-Coding-Type*: 0b00 for an 'intra-coded' or I-VOP, 0b01 for a 'forward predicted' or P-VOP, 0b10 for a 'bi-directionally coded' or B-VOP, 0b11 for a 'Sprite' [11], [14]. In practice only I-VOPs and P-VOPs (see Sec.4.E) are encountered in mobile phone video. At 32 bits 0x000001B6 forms a reliable header and if a distinction is made between I-VOPs and P-VOPs 34 bits are available.

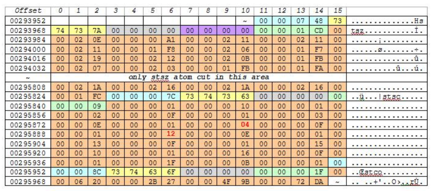
4) AAC Audio: Stored samples consist of raw AAC data streams. A raw data stream consists of a number of syntactic elements [12]. For example, a *Single-Channel-Element* (SCE, code 0b000) is a mono stream and a *Channel-Pair-Element* (CPE, code 0b001) is a two-channel stream. A *Fill* Element (FIL, code 0b110) is used to pad-out a data block. In a handset the raw data stream is most likely to be either mono or stereo and therefore start with an SCE or CPE respectively. We shall give an SCE example here. The SCE has an internal structure that defines parameters and contains the audio data. The structure of a frame header [12] is shown in Fig. 11. The frame is byte aligned and has the following fields: *Element-code*, 3 bits; *Element Instance Tag* (EIT), 4 bits; *Global-Gain* (GG), 8 bits; Reserved (R), 1 bit = 0; *Window-Sequence* (WSeq), 2 bits; *Window-Shape* (WS), 1 bit; *Noof-Scale-Factor-Bands* (Max-SFB), 4 or 6 bits; *Scale-Factor-Grouping* (SFG), 7 bits. This is followed by the audio payload data. The EIT demarks specific instances of a syntactic element.

The most reliable bits for forming tags are: SCE = 0b000, for the first SCE in the sample EIT will be 0b0000, and R=0 making 8 bits. From observation: WSeq = 0b10; WS = 0; Max-SFB is 4 or 6 bits depending on the value of WSeq, in this case Max-SFB is 4 bits (0b1111); SFG = 0b0101101 making 22 reliable bits in all. GG can vary from one sample to another within a video and from video to video. The AAC raw data streams are likely to also contain a number of syntactic FIL elements. The string of FIL elements begins with the identification 0b110 and a number of FIL elements follow. The FIL element is the bit string 0b1001 0110. This bit pattern is another reliable indication of the presence of AAC data. As the FIL element is not purposely byte-aligned the bytes: 0x9696... or 0x2D2D..., etc. can be observed at the end of an audio sample.

C. `moov` Atom Sub-Atoms

1) Sample Tables: Within the `moov` atom there are three atoms that enumerate the offsets of the video and audio samples: the ”sample-size table”, `stsz`; the ”sample-to-chunk

[9]A Video Object Plane is a plane on which video objects are displayed. On handsets a frame fills the entire VOP, so ”VOP” and ”frame” are synonymous.



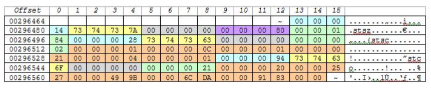
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00293952											~	00	00	07	48	73	..............Hs
00293968	74	73	7A	00	00	00	00	00	00	00	00	00	00	01	CD	00	tsz...........Í.
00293984	00	02	0E	00	00	00	A1	00	00	02	11	00	00	02	11	00	......¡.........
00294000	00	02	11	00	00	01	F8	00	00	02	06	00	00	01	F7	00	......ø.......÷.
00294016	00	02	19	00	00	02	12	00	00	02	0B	00	00	01	FB	00	..............û.
00294032	00	02	07	00	00	02	03	00	00	01	FB	00	00	01	FA	00	..........û...ú.
~	only stsz atom cut in this area																
00295808	00	02	1A	00	00	02	16	00	00	02	1A	00	00	02	16	00	....
00295824	00	01	FC	00	00	00	7C	73	74	73	63	00	00	00	00	00	..ü...|stsc.....
00295840	00	00	09	00	00	00	01	00	00	00	10	00	00	00	01	00	....
00295856	00	00	02	00	00	00	0F	00	00	00	01	00	00	00	03	00	....
00295872	00	00	0E	00	00	00	01	00	00	00	04	00	00	00	0F	00	....
00295888	00	00	01	00	00	00	12	00	00	00	0E	00	00	00	01	00	....
00295904	00	00	13	00	00	00	0F	00	00	00	01	00	00	00	15	00	....
00295920	00	00	10	00	00	00	01	00	00	00	16	00	00	00	0F	00	....
00295936	00	00	01	00	00	00	1F	00	00	00	0B	00	00	00	01	00	....
00295952	00	00	8C	73	74	63	6F	00	00	00	00	00	00	00	1F	00	..Œstco.........
00295968	00	06	20	00	00	2B	27	00	00	4F	9B	00	00	72	DA	~	.. ..+'..O›..rÚ.

Fig. 12. Portion of tables `stsz`, `stsc` and `stco` for video track: *size* (blue), *name* (yellow), *version* (grey), *sample-size* (purple), *entry-count* (green), table entries (brown). 'chunk-number jump', see text, (red font)

table”, `stsc`; and the ”chunk-offset table”, `stco`. The video and audio tracks have one set each. `stsz` contains the size of each sample in the audio and video tracks respectively in the file. `stsc` enumerates how many samples are in each chunk. `stco` gives the offset of each chunk from the beginning of the file. Figs. 12 and 13 provide examples of these atoms for video and audio respectively. All the field sizes in these atoms are four bytes each. The fields in `stsz` are: *atom-sizenameversionsample-sizeentry-countsize*. Where ”” indicates a field and ”” indicates that the field or group of fields is repeated to the end of the table. If *sample-size* = 0 then the table consists of *entry-count* entries of the respective *size*; whereas if *sample-size* $\neq$ 0, the table is compact and translates as *entry-count* entries each of *sample-size*. The fields in stsc are: *atom-sizenameversionentry-countchunk-numbersamples-per-chunksample-description-index*. The *sample-description-index* enables different chunks to be played with different codec parameters, and can be disregarded here. The fields in `stco` are: *atom-sizenameversionentry-countchunk-offset*.

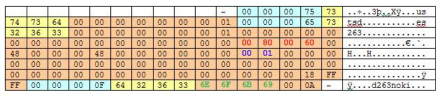
a) Video Sample Tables: The three tables that enumerate the offsets of the video samples are shown in Fig. 12. From the `stsz` video atom it can be seen that the table consists of 0x01CD = 461 entries and the video samples are of sizes (in bytes): 0x020E = 526, 0xA1 = 161, 0x0211 = 529 and so on to the end of the table. The total size of the video track is the sum of all of the sample sizes in the `stsz` video atom. From the `stsc` video atom it can be seen that the table consists of 0x09 = 9 entries. The three fields in the first entry are: *chunk-number* = 0x01 = 1, *samples- per-chunk* = 0x10 = 16, *sample-description-index* = 0x01 = 1. This table is compact and it can be seen that *chunk-number* jumps from 4 to 12 in the table (red font in Fig 12). This is a compact way of representing the fact that *chunk-numbers* 5-11 are the same as *chunk-number* 4. Only chunks that are different to the previous chunk are entered in the table; thus chunk 12 is different to chunk 11. From the `stco` video atom it can be seen that the table consists of 0x1F = 31 entries and that the offsets of the first three video chunks are: 0x0620 = 1,568, 0x2B27 = 11,047, 0x4F9B = 20,379. The offset of the first video chunk is also the offset of the first video sample in Fig. 8. From the `stsc` and `stsz` atoms, the first video chunk contains 16 samples, of which the sum of the sizes is 7,943 bytes.

b) Audio Sample Tables: The three tables that enumerate the offsets of the audio samples are shown in Fig. 13. The tables have been compacted where possible. This is frequently the case with the audio track as many of the audio samples and



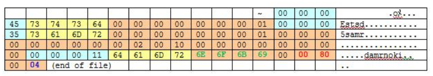
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00296464													~	00	00	00	........„....ì...
00296480	14	73	74	73	7A	00	00	00	00	00	00	00	80	00	00	01	.stsz.......€...
00296496	84	00	00	00	28	73	74	73	63	00	00	00	00	00	00	00	„...(stsc.......
00296512	02	00	00	00	01	00	00	00	0C	00	00	00	01	00	00	00	....
00296528	21	00	00	00	04	00	00	00	01	00	00	00	94	73	74	63	!...........”stc
00296544	6F	00	00	00	00	00	00	00	21	00	00	00	20	00	00	25	o.......!... ..%
00296560	27	00	00	49	9B	00	00	6C	DA	00	00	91	83	00	00	~	'..I›..lÚ..'f..¶

Fig. 13. Portion of tables `stsz`, `stsc` and `stco` for audio track: *size* (blue), *name* (yellow), *version* (grey), *sample-size* (purple), *entry-count* (green) and table entries (brown)



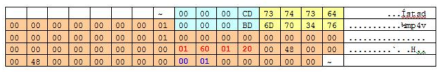
										~	00	00	00	75	73	..÷..3p..Xÿ...us
74	73	64	00	00	00	00	00	00	00	01	00	00	00	65	73	tsd...........es
32	36	33	00	00	00	00	00	00	00	01	00	00	00	00	00	263.............
00	00	00	00	00	00	00	00	00	00	00	00	80	00	60	00	............€.`.
48	00	00	00	48	00	00	00	00	00	00	00	01	00	00	00	H...H...........
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00	00	00	00	00	00	00	00	00	00	00	00	00	00	18	FF	...............ÿ
FF	00	00	00	0F	64	32	36	33	6E	6F	6B	69	00	0A	~	ÿ....d263noki...

Fig. 14. `stsd`, `s263` and `d263` atoms for H.263 video: *size* (blue), *name* (yellow), data-entries (brown): *width & height* (red font), *frame-count* (blue font), *vendor* (green font)



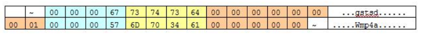
												~	00	00	00	.o}...
45	73	74	73	64	00	00	00	00	00	00	00	01	00	00	00	Estsd...........
35	73	61	6D	72	00	00	00	00	00	00	00	01	00	00	00	5samr...........
00	00	00	00	00	00	02	00	10	00	00	00	00	00	00	00	....
00	00	00	00	11	64	61	6D	72	6E	6F	6B	69	00	00	80	.....damrnoki...
00	04	(end of file)														..

Fig. 15. `stsd`, `samr` and `damr` atoms for AMR audio: *size* (blue), *name* (yellow), data entries (brown): *vendor* (green font), *mode-set* (red font), *frame-count* (blue font)



							~	00	00	00	CD	73	74	73	64	...Ístsd
00	00	00	00	00	00	00	01	00	00	00	BD	6D	70	34	76	...........½mp4v
00	00	00	00	00	00	00	01	00	00	00	00	00	00	00	00	....
00	00	00	00	00	00	00	00	01	60	01	20	00	48	00	00	.........`. .H..
00	48	00	00	00	00	00	00	00	01	00	00	00	00	00	~	

Fig. 16. `stsd` and mp4v atoms for MPEG-4 video: *atom-size* (blue), *name* (yellow), data entries (brown): *width & height* (red font), *frame-count* (blue font)



	~	00	00	00	67	73	74	73	64	00	00	00	00	00	00	...gstsd......
00	01	00	00	00	57	6D	70	34	61	00	00	00	00	00	~	.....Wmp4a......

Fig. 17. `stsd` and mp4a atoms for AAC audio: *atom-size* (blue), *name* (yellow), data entries (brown)

chunks will be the same size. The `stsz` audio atom is compact and is read as 0x0184 = 388 samples each of size 0x80 = 128 bytes. The total size of the audio track is the sum of all of the sample sizes in the `stsz` audio atom. The `stsc` audio atom has only two entries. First entry: *chunk-number* = 0x01 = 1, *samples-per-chunk* = 0x0C = 12, *sample-description-index* = 0x01 = 1. Second Entry: *chunk-number* = 0x21 = 33, *samples-per-chunk* = 0x04 = 4, *sample-description-index* = 0x01 = 1. The last audio chunk (No. 33) thus has only 4 samples all the other 32 chunks have 12 samples each. From the `stco` audio atom the table consists of 0x21 = 33 entries. The offsets of the first three audio chunks are: 0x20 = 32, 0x2527 = 9,511, 0x499B = 18,843. The offset of the first audio chunk is also the offset of the first audio sample in Fig. 10 and the first audio chunk is the very first chunk in the file. From the `stsc` and `stsz` atoms, the first audio chunk contains 12 samples, each of size 128 bytes. The size of the first audio chunk is, therefore, 1,536 bytes.

2) Sample Description Atom (`stsd`): Detailed information on the codec and coding parameters are given by the `stsd` atom [6], [7]. The `stsd` atom contains the "coding-name" atom which, in turn, contains an "extension" atom specific to the codec used. Some of the parameters contained within the `stsd` atom define parameters in the frame headers described in Sec. 4.B. Coding-name and extension atoms commonly encountered on handsets will now be described.

a) `s263` Atom (Video): In Fig. 14 the coding-name atom can be seen to be, "s263" and the extension atom, "d263". "s263" defines the use of H.263 video [5]. The *width* and *height* of the video frame in pixels are 0x0080 = 128, 0x0060 = 96; this is sub-QCIF, predicting the source format in the video header of Sec. 4.B.1. *Frame-count* gives the number of frames per sample: 0x0001 means that each sample-offset given by the `stco` and `stsz` atoms corresponds to a single frame of video. The *vendor* is listed as "noki" (0x6E6F6B69).

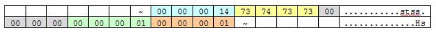
b) `samr` Atom (Audio): In Fig. 15 the coding-name atom can be seen to be "samr" and the extension atom "damr". "samr" defines the use of narrow-band AMR [5]. The *vendor* is listed as "noki", *mode-set* = 0x0080, *frame-count* = 0x04. Each bit in the *mode-set* byte sets one active codec mode; bit zero sets mode 0 etc. [5]. Hence 0x0080 sets mode 7; this agrees with the FT parameter in the AMR header of Sec. 4.B.2. *frame-count* is the number of AMR frames that constitute one MPEG-4 audio sample; this is four as observed in Sec. 4.B.2 and Fig. 10.

c) `mp4v` Atom (video): In Fig. 16 the coding-name atom can be seen to be "mp4v"; this defines the use of the MPEG-4 video codec [5]. The extension atom was "esds" (not shown). Parameters within the mp4v atom include: *width* = 0x0160 = 352 & *height* = 0x0120 = 288 pixels; *frame-count* = 0x0001. *frame-count* is the number of video frames that constitute one MPEG-4 video sample, this agrees with the observed stored data file (not shown).

d) `mp4a` Atom (Audio): From Fig. 17 the coding-name atom can be seen to be "mp4a" this defines the use of the AAC codec [5]. The extension atom was "esds" (not shown).

D. Embedded Time Stamp

The MPEG-4 standard has provision for several time stamps to be embedded in the video file [6]. There are several creation times and the same number of modification times enabling different components of the overall presentation to have different time stamps (e.g. the video and audio tracks). The provision is usually carried over to 3gp video encountered on handsets (Motorola and Samsung handsets are notable exceptions). 3gp has only one video and one audio track [5] and this gives rise to five creation and five modification time stamps (all ten time stamps often are the same value). These time stamps are located within the movie header (`mvhd`), track header (`tkhd`) and media header (`mdhd`) atoms. Each atom has a 4-byte *name* preceded by a 4-byte *size*. The time stamp is a 4-byte integer representing the number of seconds elapsed since 00:00:00 01.01.1904. The offsets of the time stamps from the start of the respective atom are: *creation-time*, 12 bytes and *modification-time*, 16 bytes. Fig. 6 shows a pair of time stamps: 0xBBD61852 which translate to: 04:38:42 11/11/2003.



						~	00	00	00	14	73	74	73	73	00	...........stss.
00	00	00	00	00	00	01	00	00	00	01	~					..............Hs

Fig. 18. `stss` atom: *size* (blue), *name* (yellow), *version* (grey), *entry-count* (green) and *sync-sample-number* (brown)

E. Independent Samples

The majority of the video samples in an MPEG-4/3gp video cannot be viewed independently as they are merely instructions to derive the current picture from an earlier picture [8]. They can, therefore, only be decoded as part of a sequence of samples. The sequence starts with an intra-coded sample or I-frame, which can be decoded and displayed without the prior decoding of any other samples. The numbers of the samples that can be decoded as I-frames are stored as a table in the `stss` atom (see Fig. 18). The format of `stss` is: *atom-sizenameversionentry-countsync-sample-number*. *sync-sample-number* is the number of the sample that can act as a synchronization point i.e. the I-frame. A single I-frame, sample number 1, is listed in this example. Even if only the I-frame is recovered from the `mdat` atom, it still can be viewed. Video clips shot on handsets typically have only one I-frame [10]: the first video sample[11] . The I-frame may be considerably larger than the other samples; typically, ˜ 1 to 4 times the size of a normal sample.

V. Enhanced Video File Recovery with Xtractor

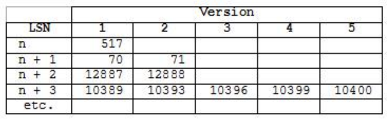
The information contained within the `moov` atom can be used to guide the extraction of the `mdat` atom, increasing the chance of recovering either a complete video clip or a playable partial video clip. The `moov` atom contains the offsets of the video and audio samples within the `mdat` atom and these can be used to test whether a sector in the Source File could be the corresponding page in the video file. A sector under scrutiny will be called a ”Candidate”. Testing Candidate sectors is the essence of the Xtractor technique. It has been found to be particularly valuable when the desired version of a sector cannot be determined from the Spare Area data, when files have become interleaved through LSN reuse or when part of the video file has been overwritten.

A. Overview of Methodology

The extraction sequence consists of the following seven steps: (i) Build an index of the sectors contained within the Source File, (ii) locate the first `moov` atom header in the Source File, (iii) extract the `moov` atom by following the sequence of LSNs, (iv) analyse the `moov` atom to compute the offsets of the video & audio samples, (v) calculate, from the `moov` atom, the size of the `mdat` atom and locate an `mdat` atom of the correct size, (vi) extract the `mdat` atom by following the LSN sequence and testing the Candidate sectors, (vii) append the `moov` atom to the mdat atom to obtain the complete file. Then locate and repeat with the next `moov` atom until no more `moov` atoms are found. A detailed description of the extraction procedure follows.

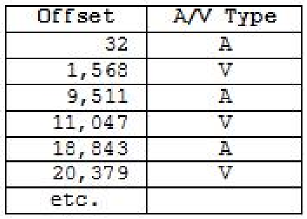
[10]Videos shot on Samsung handsets typically have multiple I-frames.

[11]Pausing a video during recoding will cause the generation of another I-frame.



	Version				
LSN	1	2	3	4	5
n	517				
n + 1	70	71			
n + 2	12887	12888			
n + 3	10389	10393	10396	10399	10400
etc.					

Fig. 19. VT for sectors corresponding to a sequence of LSNs



Offset	A/V Type
32	A
1,568	V
9,511	A
11,047	V
18,843	A
20,379	V
etc.	

Fig. 20. Chunk Offsets and A/V type

B. Extraction Procedure

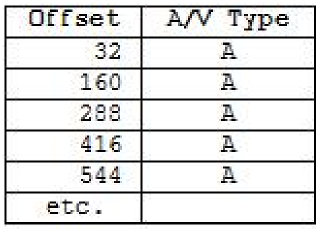
1) Building the Index: Sectors containing `ftyp`, `mdat` and `moov` atom headers, the mdat size, LSN, status etc. are indexed against their MI value. Spare Area data will vary from device to device, for examples see Breeuwsma et al. [4].

2) `moov` Atom Extraction: To extract a `moov` atom the sequence of LSNs is followed, starting from the LSN of the `moov` atom header. Where there are multiple sectors with the same LSN a VT is created as described in Sec. 5.B.3. One MI value from each row is then selected and the corresponding sector extracted. The number of sectors occupied by the `moov` atom can be calculated from its size and used to terminate its extraction. It is important in the subsequent extraction of the `mdat` atom that the `moov` atom is intact. The integrity of a `moov` atom can be checked by following the hierarchical atom structure of the moov atom. It is extremely unlikely that a `moov` atom whose structure was entirely self-consistent would not be genuine.

3) The Version Table: In the extraction of an atom, either `mdat` or `moov`, a VT similar to that of Fig. 4 is constructed, as shown in Fig. 19. Each row in the table contains the MI values of the sectors with the same LSN, which correspond to a single page in the atom being extracted. The required LSNs are searched for in the direction of increasing memory address. The rows, therefore, should be in chronological order from left to right (Sec. 2.B.4). To extract the sectors for a single version of an atom one MI value from each row is selected. Selection criteria can include physical memory location (e.g. highest memory address), sector status etc. or combinations thereof. As with the VT for the FAT volume rebuild in Sec. 3.B.1 the manual choice option allows individual sectors to be chosen.

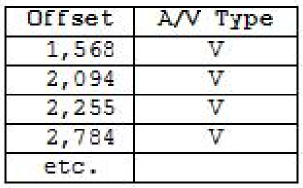
4) Computing the Sample Offsets: For the example video file in Sec. 4: Fig. 20 shows the chunks from the `stco` video and audio atoms placed in order of their respective offsets, along with their Audio/Video (A/V) type.

Within each chunk the offsets of the individual samples can



Offset	A/V Type
32	A
160	A
288	A
416	A
544	A
etc.	

Fig. 21. Sample Offsets, Audio



Offset	A/V Type
1,568	V
2,094	V
2,255	V
2,784	V
etc.	

Fig. 22. Sample Offsets, Video

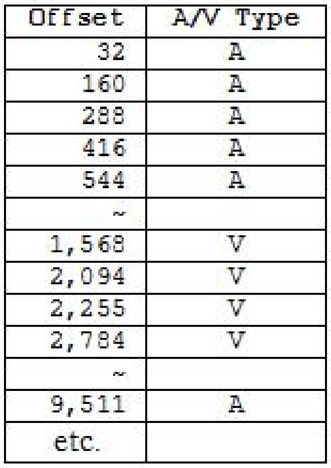
be computed by adding the sample sizes (from the respective `stsz` atom) to the chunk offset. The offset of the first audio chunk is 32. The sample offsets for the first audio chunk are given in Fig. 21. The first two of these sample offsets can be confirmed in Fig. 10.

The offset of the first video chunk is 1,568. The sample offsets for the first video chunk are given in Fig. 22. The first three of these sample offsets can be confirmed in Fig. 8.

Combining Figures 20, 21 & 22 the sample offsets can be placed in order, as shown in Fig. 23. The offset and A/V type for every sample in the video file to be extracted has been determined from its `moov` atom alone (the video codec performs this computation on playing a video). Essential to the Xtractor methodology are the computation of the page in the extracted file that contains the sample and the computation of the offset of the sample from the beginning of that page (the Page Offset). This ”Page and Offset Table” can be derived from Fig. 23 and is shown in Fig. 24.

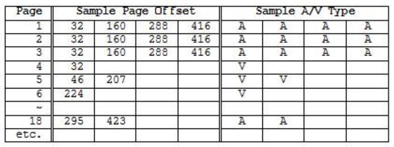
5) Locating the Corresponding `mdat` and `ftyp` Atom Headers: The size of the `mdat` atom as determined by the `moov` atom can be calculated by adding the sizes of the video and audio tracks (Sec. 4.C.1). In the original video clip this would be equal to the stated `mdat` size shown in Fig. 5. By comparing the `mdat` size calculated from the `moov` atom to those in the respective `mdat` headers in the Source File, the corresponding `mdat` header and adjacent `ftyp` header can be found. As the `mdat` size is essentially a random number (between say ˜5k and ˜1M), the probability that two unrelated video clips will have exactly the same `mdat` size is very small.

6) Enhanced `mdat` Atom Extraction: For each matching `mdat` header in the Source File a VT is constructed (Sec. 5.B.3). In the `mdat` case, however, sectors with the correct LSN are not accepted automatically for inclusion in the VT; instead Candidate sectors are checked against the Page and



Offset	A/V Type
32	A
160	A
288	A
416	A
544	A
~	
1,568	V
2,094	V
2,255	V
2,784	V
~	
9,511	A
etc.	

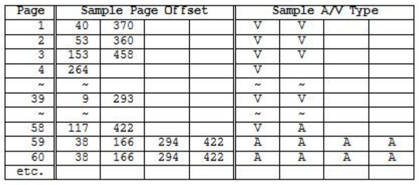
Fig. 23. Computed Sample Offset Figure, from Figures 20, 21 & 22



Page	Sample Page Offset				Sample A/V Type			
1	32	160	288	416	A	A	A	A
2	32	160	288	416	A	A	A	A
3	32	160	288	416	A	A	A	A
4	32				V			
5	46	207			V	V		
6	224				V			
~								
18	295	423			A	A		
etc.								

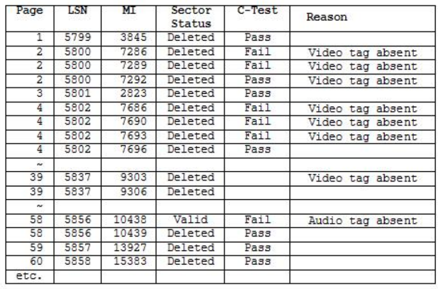
Fig. 24. Computed Page and Offset Figure, from Fig. 23 , pertaining to mdat atom of Fig. 7

Offset Table (Fig. 24) for the presence and location of the required video or audio tags. This will be called the ”Candidate Test” (C-Test). The first page to locate is Page 1, containing the `ftyp` atom and `mdat` header. The next page to locate is Page 2. The first Candidate sector for Page 2 is examined for the required audio tags at Page Offsets 32, 160, 288 and 416 in this example. If they don't match the sector *cannot* be Page 2 of the original video file and another version of the sector (i.e. with the same LSN) has to be sought. Subsequent Candidate sectors for Page 2 are then tested until one passes and is included in the VT. In every search for Candidates the entire memory is searched; continuing after a matching Candidate is found, ensuring that every matching sector version is included in the VT. When the MI values of all the versions of the sector have been recorded in the VT, the LSN is then incremented and the remainder of the pages tested in the same way. Thus sectors are offered on the basis of their LSN and preferentially placed in order of physical memory location and LSN but the Candidate Test can veto any sector. If a successful Candidate with the desired LSN is not found in the entire Source File then the LSN is repeatedly incremented and the memory searched until all matching sectors are located or the (pre-discovered) highest LSN is reached. The number of pages occupied by an individual `mdat` atom can be calculated from its size and used to terminate the extraction of the `mdat` atom. If the highest LSN in the Source File was reached before all the pages were located then the search would be terminated. The remainder of the `mdat` atom in the extracted file would be padded with nulls to preserve the offset of the `moov` atom (the recovered portion



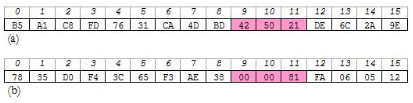
Page	Sample Page Offset				Sample A/V Type			
1	40	370			V	V		
2	53	360			V	V		
3	153	458			V	V		
4	264				V			
~	~				~	~		
39	9	293			V	V		
~	~				~	~		
58	117	422			V	A		
59	38	166	294	422	A	A	A	A
60	38	166	294	422	A	A	A	A
etc.								

Fig. 25. Page and Offset Table computed from `moov` atom (used in subsequent extraction of mdat atom with initial MI = 3845)



Page	LSN	MI	Sector Status	C-Test	Reason
1	5799	3845	Deleted	Pass	
2	5800	7286	Deleted	Fail	Video tag absent
2	5800	7289	Deleted	Fail	Video tag absent
2	5800	7292	Deleted	Pass	Video tag absent
3	5801	2823	Deleted	Pass	
4	5802	7686	Deleted	Fail	Video tag absent
4	5802	7690	Deleted	Fail	Video tag absent
4	5802	7693	Deleted	Fail	Video tag absent
4	5802	7696	Deleted	Pass	
~					
39	5837	9303	Deleted		Video tag absent
39	5837	9306	Deleted		
~					
58	5856	10438	Valid	Fail	Audio tag absent
58	5856	10439	Deleted	Pass	
59	5857	13927	Deleted	Pass	
60	5858	15383	Deleted	Pass	
etc.					

Fig. 26. C-Test results for extraction of `mdat` atom from deleted video (initial MI = 3845)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B5	A1	C8	FD	76	31	CA	4D	BD	42	50	21	DE	6C	2A	9E

(a)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
78	35	D0	F4	3C	65	F3	AE	38	00	00	81	FA	06	05	12

(b)

Fig. 27. First 16 bytes of sector with LSN = 5837: MI = 9303, expected video tag absent (a); MI = 9306, expected video tag present (b)

of the video file would still be expected to play). Extraction of an individual `mdat` atom is completed by selecting one MI value from each row and extracting the corresponding sector. As with the `moov` atom, selection criteria can be applied to the MI values for the `mdat` atom. After a `moov/mdat` atom pair has been successfully extracted, the `moov` atom is appended to the `mdat` atom to create the complete video file.

VI. Results and Discussion

A. FAT (Nokia 6230)

A map of a FAT volume, from a Nokia 6230, rebuilt according to the method of Sec. 3 was shown in Fig. 1. An examination of the recovered directory structure reveled that there were three deleted entries for videos at a time of interest. The SCs had been overwritten in the directory and thus the sectors containing any video footage and their cluster chains that might have still existed were dissociated from the directory. A search for lost cluster chains reveled one that led to the recovery of a deleted video. The size of the recovered file was exactly the same as the file size in one of the directory entries and the time stamp embedded in the file was a few seconds earlier than the time stamp in the directory entry. We have observed (on the Nokia 6230, for example) that the embedded time-stamp (Sec. 4.D) is created when recording begins and the directory time stamp is created when recording ends. It was therefore concluded that the recovered video was most likely to be the one listed in the directory. As the definitive SC link has been lost, caution must be exercised in reporting such results evidentially; the time-stamp embedded in the video, however, remains unambiguous.

B. Xtractor

1) Nokia 6230 Example: The Xtractor method was used on the Source File from Sec. 6.A. The handset had H.263 video and AMR audio (see Sections 4.B & 4.C.2). The same deleted video recovered from the lost cluster chain was recovered again and another deleted video was recovered also. A size and time-stamp comparison led to the conclusion that it was most likely one of the other videos listed in the directory. Details of the extraction of the second video follow. From the `moov` atom the Page and Offset Table shown in Fig. 25 was computed.

From Fig. 25 it can be seen that the first chunk is from the video track and that Page 1 has two video tags at Page Offsets of 40 and 370. Page 2 has two video tags at 53 and 360, and so on for the other pages. The first video sample was the I-frame (Sec. 4.E), of size 330 bytes [12]. Page 58 has one video tag and one audio tag and is the start of the first audio chunk. Pages 59 and 60 have four audio tags each at Page Offsets of 38, 166, 294 and 422. The `mdat` size, calculated from the `moov` atom, was 40,326 bytes. Only one `mdat` atom with that size was found: at MI = 3845. This sector forms the first page of the `mdat` and the extraction proceeded from its LSN (5799), following the sequence of LSNs and testing each Candidate page. The results from the C-Test are shown in Fig. 26.

For Page 2, LSN = 5800 was searched for. The first Candidate, found (at MI = 7286), failed because the expected video sample tag for this page was absent. The Candidate sector, therefore, *could not* have been part of the original video file and was rejected. The search continued for another Candidate for Page 2, i.e. the next sector with LSN = 5800; this was found at MI = 7289. This sector also failed and the search continued. A third Candidate for Page 2 was found at MI = 7292 this sector passed and its MI value was written to the VT for Page 2. No more Candidates were found for Page 2. The remaining pages in the file were searched for a similar manner. In extracting the file, 112 Candidate pages were rejected. The resulting video clip contained a 78-page `mdat` atom and could be played normally on QuickTime 7. For Page 39, a video tag with a Page Offset of 9 (see Fig. 25) in a sector with LSN = 5837 (see Fig. 26) was required. The first 16 bytes of each of the two Candidate sectors are shown in Fig. 27. The required video tag is clearly missing in (a) and present in (b). The distribution of the pages of the video file throughout the memory space is apparent from the spread of their MI values in Fig. 26.

It can be seen from Fig. 26 that where there are multiple candidates, the sector which ultimately proves to be the correct sector can be found at the higher memory address. We have found that this is likely to be the case; but not always, as shown

[12]This is very small for an I-frame and is because the start of this video was shot in the dark.

Page	Sample Page Offset	Sample A/V Type
~		
4	32	V
5	*(no tags)*	*(no tags)*
6	*(no tags)*	*(no tags)*
7	452	V
etc.		

Fig. 28. Page and Offset Table for a video clip with multi-page I-frame

in Sec. 6.B.5. Statistically, therefore, the physical location of the sector can be a useful discriminator. For this deleted video file, Fig. 26 shows that there existed two sectors (candidates for Page 58), one with deleted status that proved to be the correct sector and another sector with valid status that proved to be erroneous. The sector status can therefore be a useful sector discriminator. It can be seen from Fig. 25 that the audio tag Page Offsets have a regular structure, this is because the audio samples have a fixed size. Such regular structures will occur within a video file and similarities may exist between the sectors of different video files[13] . Such pages cannot be discriminated by Candidate Testing, and may reduce the effectiveness of the method. However, only similarly tagged sectors that also have the same LSN will be affected and these will lead to additional entries in the VT. Subsequent discrimination may still be achieved by considering physical memory address and sector status. All the remaining results in this section are from Nokia 7600 handsets (sw. ver. 04.05), with Samsung KEE00D00CM NAND Flash chips. All resulting videos were playable with Apple QuickTime 7.

2) Effect of Large I-frame: Fig. 28 shows the Page and Offset Table for a video file with an I-frame (Sec. 4.E.) that started on Page 4 at Page Offset 32. The size of the I-frame was 1,956 bytes so it continues on Pages 5, 6 and 7. Consequently Pages 5 and 6 contain no tags at all. This complete absence of any tags can also be tested.

3) Interleaved Sector Example: Fig. 29 shows the C-Test results from the extraction of a video that had its sectors interleaved with those of another file or files (Sec. 2.B.3). The extraction proceeded for Pages 1 to 44 as described in Sec. 6.B.1. Page 44 had LSN = 6574, thus Page 45 was expected to have LSN = 6575. A successful candidate was not found in the file with this LSN so the next incremental LSN was searched for repeatedly until LSN = 6931 was found; this page passed and was included in the VT. Thus 355 sectors were skipped before the file was picked up again. The resulting video had an `mdat` size of 1,796,120 bytes and could be played normally.

4) Partial File Example: Fig. 30 is from an extraction of a deleted video that had been partially overwritten. From Page 1 to Page 111 (with LSN = 24889) the extraction was successful, but no matching sectors for subsequent pages were found. The end of the `mdat` atom was, therefore, padded with nulls to preserve the offset of the `moov` atom. The recovered portion of the video could be played normally.

[13]Particularly for videos shot on the same handset that have an audio chunk first; these will have the same `ftyp` atom size placing the audio samples at the same offsets. Subsequent video chunks are unaffected.

Page	LSN	MI	Sector Status	C-Test	Reason
1	6531	48036	Valid	Pass	
2	6532	36164	Valid	Pass	
44	6574	60293	Valid	Pass	
45	6575		(Skip)		LSN not found
~	~				
45	6930		(Skip)		LSN not found
45	6931	17701	Valid	Pass	
etc.					

Fig. 29. Results from C-Test for extant `mdat` atom with interleaved sectors

Page	LSN	MI	Sector Status	C-Test	Reason
~					
111	24889	17453	Deleted	Pass	
112	24890		(Skip)		LSN not found
~					
112	48239		(Skip)		LSN not found
	Limit reached				

Fig. 30. Results from C-Test for deleted video with partially overwritten `mdat` atom

Page	LSN	MI	Sector Status	C-Test	Reason
2	24636	44908	Deleted	Pass	
2	24636	44920	Valid	Fail	Audio tag absent

Fig. 31. Two sectors with same LSN, correct sector at the lower memory address

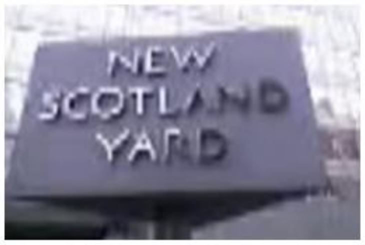


Fig. 32. Still made from I-frame taken from handset video

5) Sector Order: Fig. 31 is from a C-Test for a deleted video. Both Candidate sectors were from the same memory block. The sector that proved to be the correct sector was to be found at the lower memory address (lower MI value) contrary to the observation in Sec. 6.B.1.

C. Single I-Frame Example

The Xtractor method is capable of recovering video files that have been partially overwritten. In cases where the `mdat` atom has been severely corrupted and only the I-frame has been recovered it may still be viewable. An example of a still made from the single I-frame is shown in Fig. 32. The video file containing this I-frame consisted of the `ftyp` atom, the `moov` atom and the `mdat` atom containing one sample: the I-frame. The rest of the `mdat` atom was padded with nulls. The video could be played normally (albeit a still frame) on QuickTime 7. The size of the sample making up the I-frame was 1,238 bytes and would, therefore, fit into a single typical cluster (2k bytes).

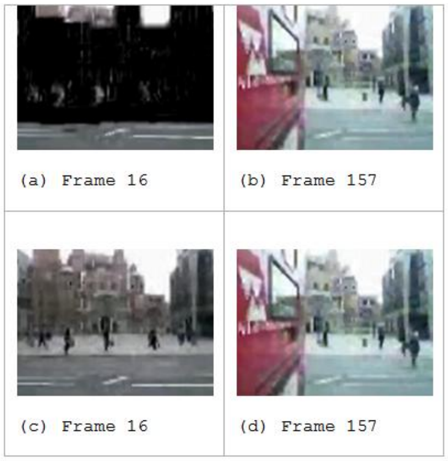


Fig. 33. Stills made from video: with I-frame removed (a) & (b), with I-frame intact (c) & (d)

D. Missing I-Frame Example

As 3gp video normally has only a single I-frame it might be expected that if the I-frame was missing or corrupted then the remainder of the extracted segment could not be viewed. However, if the `stss` atom is edited such that its synchronization point is the number of the first video sample recovered [14] then the video segment can be expected to play. Fig. 33 shows stills from a video that has had its I-frame replaced with nulls and the synchronization point set to sample number 2. A bus was allowed to pass the field of view, refreshing the scene [15] (panning the camera has the same effect). Fig. 33 (a) is an early frame and is severely corrupted; Fig. 33 (b) is a frame from after a bus has passed, showing apparent full recovery. The original video is shown in Fig 33 (c) and (d) for comparison.

E. Extraction Probabilities

The `moov` atom of Fig. 6 occupied 11 sectors and the `mdat` atom of Fig. 5 occupied 569 sectors. In a well-used handset, the probability of all 580 sectors being in a continuous sequence of LSNs would be quite low. As the size of the `moov` atom is less than 1/52nd of the size of the whole video, the probability of the `moov` atom existing as a continuous sequence of LSNs is much greater than that of the whole video. Once an intact `moov` atom has been recovered the probability of recovering a viewable video increases significantly. Video samples are often of the order of 512 bytes, and audio samples are typically 128 bytes (AMR) or 743 bytes (AAC). It is therefore likely that a video or audio sample tag will appear on most 512-byte pages in a video file. The probability of a 34-bit H.263 tag occurring by chance alone at a specific place in any sector is: $1{:}2^{34} \approx 6\text{x}10^{-11}$. In a 32M byte file there would be 64k 512-byte sectors and we would expect one false positive in ˜260,000 handset examinations. In reality the data in the Source File is not completely uncorrelated, having a high occurrence of runs of zeros and identical tags from different videos, this does increase the chance of a false positive. However, the probability of rejecting erroneous sectors has been found to be sufficiently high to form the basis of a usable discrimination technique. The probability of a false positive can be reduced by marking in the Master Index as unavailable, sectors that are already known to belong to other files. Files extracted from the Source File using Xtractor can also be excluded subsequently, thus iteratively increasing the chance of recovering heavily fragmented video files.

[14]The sample number can be computed from its offset as in Sec. 5.B.4.

[15]As a scene changes, previously un-encoded material has to be introduced; for example in the form of intra-coded macro blocks. As the video progresses, there may be enough new material that the whole scene is represented.

VII. Conclusion

We suggest that an integrated approach to recovering files from handset memory dumps would be beneficial. This can include rebuilding the FAT volume, using Version Tables to catalogue sectors with the same LSN, recovering lost cluster chains and using the sector status and the sector's physical location. In the case of MPEG-4/3gp video files additional methods can be employed. For every video and audio sample the Xtractor method can determine, from the `moov` atom alone, which page in the original file the sample was on and its Page Offset. By comparing the samples on Candidate sectors with their expected type and position, it is possible to reject erroneous sectors and greatly increase the probability of extracting a viewable video. In cases where deleted files have become interleaved Xtractor can skip over the erroneous sectors, resuming the extraction when the desired file is located again. As each page can be verified independently partial videos can be extracted successfully. Videos consisting of the single I-frame are still viewable and videos in which the I-frame is missing or corrupted may still be viewable also. Complete and partial videos recovered by the Xtractor method can be played on readily available video playback software (e.g. Apple QuickTime 7).

References

[1] E. Casey (ed.), *Digital Evidence and Computer Crime*, 2nd ed. Elsevier Academic Press, 2004.
[2] J. Mintel (ed.), *Upgrading and Repairing PCs*, 10th ed., Que, 1998.
[3] Internet address: beginningtoseethelight.org/fat16/index.php, [first accessed 3rd May 2006].
[4] M. Breeuwsma et al., *Forensic Data Recovery from Flash Memory*, Small Scale Digital Forensics J. Vol. 1(1), June 2007.
[5] *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects Transparent end-to-end packet switched streaming service (PSS); 3GPP file format (3GPP)* 3GPP TS 26.244 V6.5.0 (2006-06).
[6] *Information technology - Coding of audio-visual objects - Pt.12: ISO base media format*, Ref. No. ISO/IEC 14496-12:2005/Cor.1:2005(E).
[7] *Information technology - Coding of audio-visual objects - Pt.14: MP4 file format*, Ref. No. ISO/IEC 14496-14:2003(E).
[8] F. Pereira & T. Ebrahimi (eds.), *The MPEG-4 Book*, Prentice Hall IMSC multimedia series, 2002.
[9] *Video coding for low bit rate communication* ITU-T H.263, 01/2005.
[10] J. Sjoberg et al., *Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs*, IETF RFC 3267, June 2002.
[11] *Information technology - Coding of audio-visual objects - Pt.2: Visual*, Ref. No. ISO/IEC 14496-2:2004(E).
[12] *Information technology - Generic coding of moving pictures and associated audio information - Pt.7: Advanced Audio Coding (AAC)*, Ref. No. ISO/IEC 13818-7:2006(E).

[13] *3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Mandatory speech codec speech processing functions; Adaptive Multi-Rate (AMR) speech codec frame structure*, 3GPP TS 26.101 V7.0.0 (2007-06).

[14] R. van der Knijff, *10 Good Reasons Why You Should Shift Focus to Small Scale Digital Device Forensics*, Available at http://WWW.dfws.org/2007/procedings/vanderknijff_pres.pdf, [first accessed 20th Feb. 2008].

James Luck James Luck is a Forensic Engineer in the Metropolitan Police, Forensic Digital Evidence Unit at New Scotland Yard. He gained his PhD in Electrical Engineering from Kings College London in 1996. He currently leads the research activity in the recovery and understanding of data from embedded systems. James is a member of the IET, the IEEE and the Institute of Physics. He has given expert testimony on digital evidence in several criminal trials.

Mark Stokes Mark Stokes is head of the Metropolitan Police, Forensic Digital Evidence Unit at New Scotland Yard. He has worked in the area of forensic electronic engineering for over 20 years, with the past 12 years dedicated to telecommunications and embedded systems. His current research interests include the recovery of data from embedded systems and cell site analysis. Mark is a member of the IET. He has given expert testimony on digital evidence in numerous criminal trials.