An Introduction to DOS FAT Volume and File Structure

Mark Kampe markk@cs.ucla.edu

1. Introduction

When the first personal computers with disks became available, they were very small (a few megabytes of disk and a few dozen kilobytes of memory). A file system implementation for such machines had to impose very little overhead on disk space, and be small enough to fit in the BIOS ROM. BIOS stands for **BASIC I/O Subsystem**. Note that the first word is all upper-case. The purpose of the BIOS ROM was to provide run-time support for a BASIC interpreter (which is what Bill Gates did for a living before building DOS). DOS was never intended to provide the features and performance of real timesharing systems.

Disk and memory size have increased in the last thirty years, People now demand state-of-the-art power and functionality from their PCs. Despite the evolution that the last decades have seen, old standards die hard. Much as European train tracks maintain the same wheel spacing used by Roman chariots, most modern OSs still support DOS FAT file systems. DOS file systems are not merely around for legacy reasons. The ISO 9660 CDROM file system format is a descendent of the DOS file system.

The DOS FAT file system is worth studying because:

- It is heavily used all over the world, and is the basis for more modern file system (like 9660).
- It provides reasonable performance (large transfers and well clustered allocation) with a very simple implementation.
- It is a very successful example of "linked list" space allocation.

2. Structural Overview

All file systems include a few basic types of data structures:

- bootstrap

  code to be loaded into memory and executed when the computer is powered on. MVS volumes reserve the entire first track of the first cylinder for the boot strap.

- volume descriptors

  information describing the size, type, and layout of the file system ... and in particular how to find the other key meta-data descriptors.

- file descriptors

  information that describes a file (ownership, protection, time of last update, etc.) and points where the actual data is stored on the disk.

- free space descriptors

  lists of blocks of (currently) unused space that can be allocated to files.

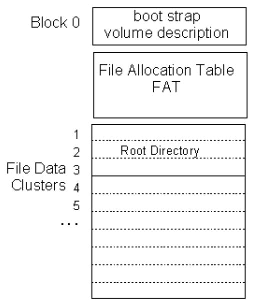
- file name descriptors

  data structures that user-chosen names with each file.

DOS FAT file systems divide the volume into fixed-sized (physical) blocks, which are grouped into larger fixed-sized (logical) block clusters.

The first block of DOS FAT volume contains the bootstrap, along with some volume description information. After this comes a much longer **File Allocation Table** (FAT from which the file system takes its name). The File Allocation Table is used, both as a free list, and to keep track of which blocks have been allocated to which files.

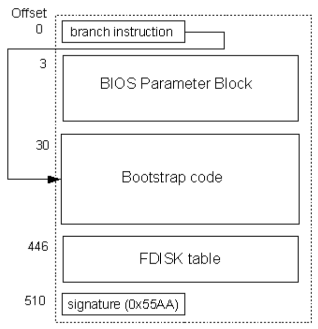
The remainder of the volume is data clusters, which can be allocated to files and directories. The first file on the volume is the root directory, the top of the tree from which all other files and directories on the volume can be reached.



3. Boot block BIOS Parameter Block and FDISK Table

Most file systems separate the first block (pure bootstrap code) from volume description information. DOS file systems often combine these into a single block. The format varies between (partitioned) hard disks and (unpartitioned) floppies, and between various releases of DOS and Windows ... but conceptually, the boot record:

- begins with a branch instruction (to the start of the real boostrap code).
- followed by a volume description (BIOS Parameter Block)
- followed by the real bootstrap code
- followed by an optional disk partitioning table
- followed by a signature (for error checking).



3.1 BIOS Parameter Block

After the first few bytes of the bootstrap comes the BIOS parameter block, which contains a brief summary of the device and file system. It describes the device geometry:

- number of bytes per (physical) sector
- number of sectors per track
- number of tracks per cylinder
- total number of sectors on the volume

It also describes the way the file system is layed out on the volume:

- number of sectors per (logical) cluster
- the number of reserved sectors (not part of file system)
- the number of Alternate File Allocation Tables
- the number of entries in the root directory

These parameters enable the OS to interpret the remainder of the file system.

3.2 FDISK Table

As disks got larger, the people at MicroSoft figured out that their customers might want to put multiple file systems on each disk. This meant they needed some way of partitioning the disk into logical sub-disks. To do this, they added a small partition table (sometimes called the FDISK table, because of the program that managed it) to the end of the boot strap block.

This FDISK table has four entries, each capable of describing one disk partition. Each entry includes

- A partition type (e.g. Primary DOS partition, UNIX partition).
- An ACTIVE indication (is this the one we boot from).
- The disk address where that partition starts and ends.
- The number of sectors contained within that partition.

Partn	Type	Active	Start (C:H:S)	End (C:H:S)	Start (logical)	Size (sectors)
1	LINUX	True	1:0:0	199:7:49	400	79,600
2	Windows NT		200:0:0	349:7:49	80,000	60,000
3	FAT 32		350:0:0	399:7:49	140,000	20,000
4	NONE					

In older versions of DOS the starting/ending addresses were specified as cylinder/sector/head. As disks got larger, this became less practical, and they moved to logical block numbers.

The addition of disk partitioning also changed the structure of the boot record. The first sector of a disk contains the Master Boot Record (MBR) which includes the FDISK table, and a bootstrap that finds the active partition, and reads in its first sector (Partition Boot Record). Most people (essentially everyone but Bill Gates :-) make their MBR bootstrap ask what system you want to boot from, and boot the active one by default after a few seconds. This gives you the opportunity to choose which OS you want to boot. Microsoft makes this decision for you ... you want to boot Windows.

The structure of the Partition Boot Record is entirely operating system and file system specific ... but for DOS FAT file system partitions, it includes a BIOS Parameter block as described above.

4. File Descriptors (directories)

In keeping with their desire for simplicity, DOS file systems combine both file description and file naming into a single file descriptor (directory entries). A DOS directory is a file (of a special type) that contains a series of fixed sized (32 byte) directory entries. Each entry describes a single file:

- an 11-byte name (8 characters of base name, plus a 3 character extension).
- a byte of attribute bits for the file, which include:
  - Is this a file, or a sub-directory.
  - Has this file changed since the last backup.
  - Is this file hidden.
  - Is this file read-only.
  - Is this a system file.
  - Does this entry describe a volume label.
- times and dates of creation and last modification, and date of last access.
- a pointer to the first logical block of the file. (This field is only 16 bits wide, and so when Microsoft introduced the FAT32 file system, they had to put the high order bits in a different part of the directory entry).
- the length (number of valid data bytes) in the file.

Name (8+3)	Attributes	Last Changed	First Cluster	Length
.	DIR	08/01/03 11:15:00	61	2,048
..	DIR	06/20/03 08:10:24	1	4,096
MARK	DIR	10/15/04 21:40:12	130	1,800
README.TXT	FILE	11/02/04 04:27:36	410	31,280

If the first character of a files name is a NULL (0x00) the directory entry is unused. The special character (0xE5) in the first character of a file name is used to indicate that a directory entry describes a deleted file. (See the section on Garbage collection below)

Note on times and dates:

DOS stores file modification times and dates as a pair of 16-bit numbers:

- 7 bits of year, 4 bits of month, 5 bits of day of month
- 5 bits of hour, 6 bits of minute, 5 bits of seconds (x2).

All file systems use dates relative to some epoch (time zero). For DOS, the epoch is midnight, New Year's Eve, January 1, 1980. A seven bit field for years means that the the DOS calendar only runs til 2107. Hopefully, nobody will still be using DOS file systems by then :-)

5. Links and Free Space (File Allocation Table)

Many file systems have very compact (e.g. bitmap) free lists, but most of them use some per-file data structure to keep track of which blocks are allocated to which file. The DOS File Allocation Table is a relatively unique design. It contains one entry for each logical block in the volume. If a block is free, this is indicated by the FAT entry. If a block is allocated to a file, the FAT entry gives the logical block number of the **next** logical block in the file.

5.1 Cluster Size and Performance

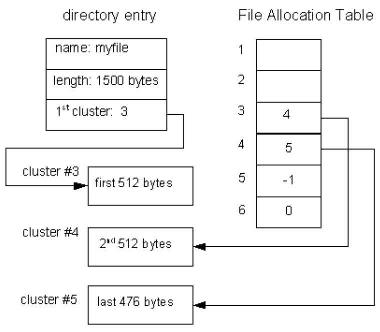
Space is allocated to files, not in (physical) blocks, but in (logical) multi-block clusters. The number of clusters per block is determined when the file system is created.

Allocating space to files in larger chunks improves I/O performance, by reducing the number of operations required to read or write a file. This comes at the cost of higher internal fragmentation (since, on average, half of the last cluster of each file is left unused). As disks have grown larger, people have become less concerned about internal fragmentation losses, and cluster sizes have increased.

The maximum number of clusters a volume can support depends on the width of the FAT entries. In the earliest FAT file systems (designed for use on floppies, and small hard drives). An 8-bit wide FAT entry would have been too small (256 * 512 = 128K bytes) to describe even the smallest floppy, but a 16-bit wide FAT entry would have been ludicrously large (8-16 Megabytes) ... so Microsoft compromised and adopted 12-bit wide FAT entries (two entries in three bytes). These were called FAT-12 file systems. As disks got larger, they created 2-byte wide (FAT-16) and 4-byte wide (FAT-32) file systems.

5.2 Next Block Pointers

A file's directory entry contains a pointer to the first cluster of that file. The File Allocation Table entry for that cluster tells us the cluster number **next** cluster in the file. When we finally get to the last cluster of the file, its FAT entry will contain a -1, indicating that there is no next block in the file.



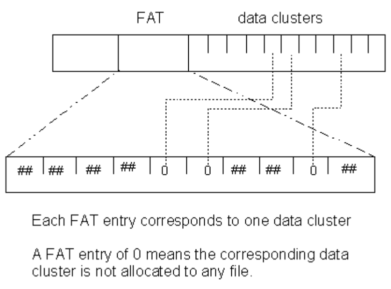
The "next block" organization of the FAT means that in order to figure out what physical cluster is the third logical block of ar file, must know the physical cluster number of the second logical block. This is not usually a problem, because almost all file access is sequential (reading the first block, and then the second, and then the third ...).

If we had to go to disk to re-read the FAT each time we needed to figure out the next block number, the file system would perform very poorly. Fortunately, the FAT is so small (e.g. 512 bytes per megabyte of file system) that the entire FAT can be kept in memory as long as a file system is in use. This means that successor block numbers can be looked up without the need to do any additional disk I/O. It is easy to imagine

5.3 Free Space

The notion of "next block" is only meaningful for clusters that are allocated to a file ... which leaves us free to use the FAT entries associated with free clusters as a free indication. Just as we reserved a value (-1) to mean **end of file** we can rerserve another value (0) to mean **this cluster is free**.

To find a free cluster, one has but to search the FAT for an entry with the value -2. If we want to find a free cluster near the clusters that are already allocated to the file, we can start our search with the FAT entry after the entry for the first cluster in the file.


FAT
data clusters
## 0 0 ## ## 0
Each FAT entry corresponds to one data cluster
A FAT entry of 0 means the corresponding data cluster is not allocated to any file.

5.4 Garbage Collection

Older versions of FAT file systems did not bother to free blocks when a file was deleted. Rather, they merely crossed out the first byte of the file name in the directory entry (with the reserved value 0xE5). This had the advantage of greatly reducing the amount of I/O associated with file deletion ... but it meant that DOS file systems regularly ran out of space.

When this happened, they would initiate garbage collection. Starting from the root directory, they would find every "valid" entry. They would follow the chain of next block pointers to determine which clusters were associated with each file, and recursively enumerate the contents of all sub-directories. After completing the enumeration of all allocated clusters, they inferred that any cluster not found in some file was free, and marked them as such in the File Allocation Table.

This "feature" was probably motivated by a combination of laziness and a desire for performance. It did, however, have an advantage. Since clusters were not freed when files were deleted, they could not be reallocated until after garbage colection was performed. This meant that it might be possible to recover the contents of deleted files for quite a while. The opportunity this created was large enough to enable Peter Norton to start a very successful company.

6. Descendents of the DOS file system

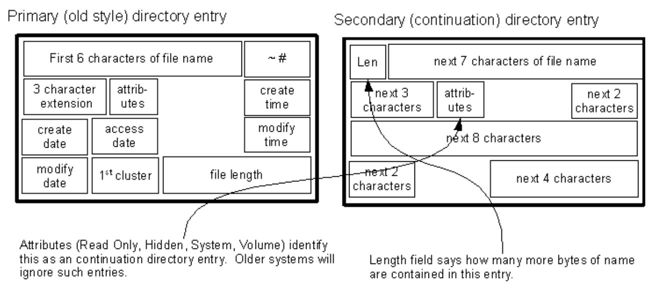
The DOS file system has evolved with time. Not only have wider (16- and 32-bit) FAT entries been used to support larger disks, but other features have been added. The last stand-alone DOS product was DOS 6.x. After this, all DOS support was under Windows, and along with the change to Windows came an enhanced version of the FAT file system called Virtual FAT (or simply VFAT).

6.1 Long File Names

Most DOS and Windows systems were used for personal productivity, and their users didn't demand much in the way of file system features. Their biggest complaints were about the 8+3 file names. Windows users demanded longer and mixed-case file names.

The 32 byte directory entries didn't have enough room to hold longer names, and changing the format of DOS directories would break hundreds or even thousands of applications. It wasn't merely a matter of importing files from old systems to new systems. DOS diskettes are commonly used to carry files between various systems ... which means that old systems still had to be able to read the new directories. They had to find a way to support longer file names without making the new files unreadable by older systems.

The solution they came up was to put the extended filenames in additional (auxiliary) directory entries. Each file would be described by an old-format directory entry, but supplementary directory entries (following the primary directory entry) could provide extenstions to the file name. To keep older systems from being confused by the new directory entries, they were tagged with a very unusual set of attributes (hidden, system, read-only, volume label). Older systems would ignore new entries, but would still be able to access new files under their 8+3 names in the old-style directory entries. New systems would recognize the new directory entries, and be able to access files by either the long or the short names.


Primary (old style) directory entry
Secondary (continuation) directory entry
First 6 characters of file name
~#
3 character extension
attrib-utes
create time
create date
access date
modify time
modify date
1st cluster
file length
Len
next 7 characters of file name
next 3 characters
attrib-utes
next 2 characters
next 8 characters
next 2 characters
next 4 characters
Attributes (Read Only, Hidden, System, Volume) identify this as an continuation directory entry. Older systems will ignore such entries.
Length field says how many more bytes of name are contained in this entry.

The addition of long file names did create one problem for the old directory entries. What would you do if you had two file names that differred only after the 8th character (e.g. datafileread.c and datafilewrite.c)? If we just used the first 8 characters of the file name in the old directory entries (datafile), we would have two files with the same name, and this is illegal. For this reason, the the short file names are not merely the first eight characters of the long file names. Rather, the last two bytes of the short name were merely made unambiguous (e.g. "~1", "~2", etc).

6.2 Alternate/back-up FATs

The File Allocation Table is a very concise way of keeping track of all of the next-block pointers in the file system. If anything ever happened to the File Allocation Table, the results would be disastrous. The directory entries would tell us where the first blocks of all files were, but we would have no way of figuring out where the remainder of the data was.

Events that corrupt the File Allocation Table are extremely rare, but the consequences of such an incident are catastrophic. To protect users from such eventualities, MicroSoft added support for alternate FATs. Periodically, the primary FAT would be copied to one of the pre-reserved alternat FAT locations. Then if something bad happened to the primary FAT, it would still be possible to read most files (files created before the copy) by using the back-up FAT. This is an imperfect solution, as losing new files is bad ... but losing all files is worse.

6.3 ISO 9660

When CDs were proposed for digital storage, everyone recognized the importance of a single standard file system format. Dueling formats would raise the cost of producing new products ... and this would be a lose for everyone. To respond to this need, the International Standards Organization chartered a sub-committee to propose such a standard.

The failings of the DOS file system were widely known by this time, but, as the most widely used file system on the planet, the committee members could not ignore it. Upon examination, it became clear that the most ideomatic features of the DOS file system (the File Allocation Table) were irrelevent to a CDROM file system (which is written only once):

- We don't need to keep track of the free space on a CD ROM. We write each file contiguously, and the next file goes immidiately after the last one.
- Because files can be written contiguously, we don't need any "next block" pointers. All we need to know about a file is where its first block resides.

It was decided that ISO 9660 file systems would (like DOS file systems) have tree structured directory hierarchies, and that (like DOS) each directory entry would describe a single file (rather than having some auxiliary data structure like and I-node to do this). 9660 directory entries, like DOS directory entries, contain:

- file name (within the current directory)
- file type (e.g. file or directory)
- location of the file's first block
- number of bytes contained in the file
- time and date of creation

They did, however, learn from DOS's mistakes:

- Realizing that new information would be added to directory entries over time, they made them variable length. Each directory entry begins with a length field (giving the number of bytes in this directory entry, and thus the number of bytes until the next directory entry).
- Recognizing the need to support long file names, they also made the file name field in each entry a variable length field.
- Recognizing that, over time, people would want to associate a wide range of attributes with files, they also created a variable length extened attributes section after the file name. Much of this section has been left unused, but they defined several new attributes for files:
    - file owner
    - owning group
    - permissions
    - creation, modification, effective, and expiration times
    - record format, attributes, and length information

But, even though 9660 directory entries include much more information than DOS directory entries, it remains that 9660 volumes resemble DOS file systems much more than they resemble any other file system format. And so, the humble DOS file system is reborn in a new generation of products.

7. Summary

DOS file systems are very simple, They don't support multiple links to a file, or symbolic links, or even multi-user access control. They are also and very economical in terms of the space they take up. Free block lists, and file block pointers are combined into a single (quite compact) File Allocation Table. File descriptors are incorporated into the directory entries. And for all of these limitations, they are probably the most widely used file system format on the planet. Despite their primitiveness, DOS file systems were used as the basis for much newer CD ROM file system designs.

What can we infer from this? That most users don't need alot of fancy features, and that the DOS file system (primitive as it may be) covers their needs pretty well.

It is also noteworthy that when Microsoft was finally forced to change the file system format to get past the 8.3 upper case file name limitations, they chose to do so with a klugy (but upwards compatible) solution using additional directory entries per file. The fact that they chose such an implementation clearly illustrates the importance of maintaining media interchangability with older systems. This too is a problem that all (successful) file systems will eventually face.

8. References

DOS file system information

- PC Guide's Overview of DOS FAT file systems. (this is a pointer to the long filename article ... but the entire library is nothing short of excellent).
- Free BSD sources, PCFS implementation, BIOS Parameter block format, and (Open Solaris) FDISK table format.
- Free BSD sources, PCFS implementation, Directory Entry format

9660 file system information

- WIkipedia Introduction to ISO 9660 file systems
- Free BSD sources, ISOFS implementation, ISO 9660 data structures.

The latest version of this document may be downloaded from http://www.freesoftware.fsf.org/ext2-doc/

This book is intended as an introduction and guide to the Second Extended File System, also known as Ext2. The reader should have a good understanding of the purpose of a file system as well as the associated vocabulary (file, directory, partition, etc).

Implementing file system drivers is already a daunting task, unfortunately except for tidbits of information here and there most of the documentation for the Second Extended Filesystem is in source files.

Hopefully this document will fix this problem, may it be of help to as many of you as possible.

Unless otherwise stated, all values are stored in little endian byte order.

Chapter 1. Historical Background

Written by Remy Card, Theodore Ts'o and Stephen Tweedie as a major rewrite of the Extended Filesystem, it was first released to the public on January 1993 as part of the Linux kernel. One of its greatest achievement is the ability to extend the file system functionalities while maintaining the internal structures. This allowed an easier development of the Third Extended Filesystem (ext3) and the Fourth Extended Filesystem (ext4).

There are implementations available in most operating system including but not limited to NetBSD, FreeBSD, the GNU HURD, Microsoft Windows, IBM OS/2 and RISC OS.

Although newer file systems have been designed, such as Ext3 and Ext4, the Second Extended Filesystem is still prefered on flash drives as it requires fewer write operations (since it has no journal). The structures of Ext3 and Ext4 are based on Ext2 and add some additional options such as journaling, journal checksums, extents, online defragmentation, delayed allocations and larger directories to name but a few.

Chapter 2. Definitions

The Second Extended Filesystem uses blocks as the basic unit of storage, inodes as the mean of keeping track of files and system objects, block groups to logically split the disk into more manageable sections, directories to provide a hierarchical organization of files, block and inode bitmaps to keep track of allocated blocks and inodes, and superblocks to define the parameters of the file system and its overall state.

Ext2 shares many properties with traditional Unix filesystems. It has space in the specification for Access Control Lists (ACLs), fragments, undeletion and compression. There is also a versioning mechanism to allow new features (such as journalling) to be added in a maximally compatible manner; such as in Ext3 and Ext4.

2.1. Blocks

A partition, disk, file or block device formated with a Second Extended Filesystem is divided into small groups of sectors called "blocks". These blocks are then grouped into larger units called block groups.

The size of the blocks are usually determined when formatting the disk and will have an impact on performance, maximum file size, and maximum file system size. Block sizes commonly implemented include 1KiB, 2KiB, 4KiB and 8KiB although provisions in the superblock allow for block sizes as big as 1024 * (2^31)-1 (see s_log_block_size).

Depending on the implementation, some architectures may impose limits on which block sizes are supported. For example, a Linux 2.6 implementation on DEC Alpha uses blocks of 8KiB but the same implementation on a Intel 386 processor will support a maximum block size of 4KiB.

Table 2-1. Impact of Block Sizes

Upper Limits	1KiB	2KiB	4KiB	8KiB
file system blocks	2,147,483,647	2,147,483,647	2,147,483,647	2,147,483,647
blocks per block group	8,192	16,384	32,768	65,536
inodes per block group	8,192	16,384	32,768	65,536
bytes per block group	8,388,608 (8MiB)	33,554,432 (32MiB)	134,217,728 (128MiB)	536,870,912 (512MiB)
file system size (real)	4,398,046,509,056 (4TiB)	8,796,093,018,112 (8TiB)	17,592,186,036,224 (16TiB)	35,184,372,080,640 (32TiB)
file system size (Linux)	2,199,023,254,528 (2TiB) [a]	8,796,093,018,112 (8TiB)	17,592,186,036,224 (16TiB)	35,184,372,080,640 (32TiB)
blocks per file	16,843,020	134,217,728	1,074,791,436	8,594,130,956
file size (real)	17,247,252,480 (16GiB)	274,877,906,944 (256GiB)	2,199,023,255,552 (2TiB)	2,199,023,255,552 (2TiB)
file size (Linux 2.6.28)	17,247,252,480 (16GiB)	274,877,906,944 (256GiB)	2,199,023,255,552 (2TiB)	2,199,023,255,552 (2TiB)

Notes:
a. This limit comes from the maximum size of a block device in Linux 2.4; it is unclear whether a Linux 2.6 kernel using a 1KiB block size could properly format and mount a Ext2 partition larger than 2TiB.

Note: the 2TiB file size is limited by the i_blocks value in the inode which indicates the number of 512-bytes sector rather than the actual number of ext2 blocks allocated.

2.2. Block Groups

This definition comes from the Linux Kernel Documentation.

Blocks are clustered into block groups in order to reduce fragmentation and minimise the amount of head seeking when reading a large amount of consecutive data. Information about each block group is kept in a descriptor table stored in the block(s) immediately after the superblock. Two blocks near the start of each group are reserved for the block usage bitmap and the inode usage bitmap which show which blocks and inodes are in use. Since each bitmap is limited to a single block, this means that the maximum size of a block group is 8 times the size of a block.

The block(s) following the bitmaps in each block group are designated as the inode table for that block group and the remainder are the data blocks. The block allocation algorithm attempts to allocate data blocks in the same block group as the inode which contains them.

2.3. Directories

This definition comes from the Linux Kernel Documentation with some minor alterations.

A directory is a filesystem object and has an inode just like a file. It is a specially formatted file containing records which associate each name with an inode number. Later revisions of the filesystem also encode the type of the object (file, directory, symlink, device, fifo, socket) to avoid the need to check the inode itself for this information

The inode allocation code should try to assign inodes which are in the same block group as the directory in which they are first created.

The original Ext2 revision used singly-linked list to store the filenames in the directory; newer revisions are able to use hashes and binary trees.

Also note that as directory grows additional blocks are assigned to store the additional file records. When filenames are removed, some implementations do not free these additional blocks.

2.4. Inodes

This definition comes from the Linux Kernel Documentation with some minor alterations.

The inode (index node) is a fundamental concept in the ext2 filesystem. Each object in the filesystem is represented by an inode. The inode structure contains pointers to the filesystem blocks which contain the data held in the object and all of the metadata about an object except its name. The metadata about an object includes the permissions, owner, group, flags, size, number of blocks used, access time, change time, modification time, deletion time, number of links, fragments, version (for NFS) and extended attributes (EAs) and/or Access Control Lists (ACLs).

There are some reserved fields which are currently unused in the inode structure and several which are overloaded. One field is reserved for the directory ACL if the inode is a directory and alternately for the top 32 bits of the file size if the inode is a regular file (allowing file sizes larger than 2GB). The translator field is unused under Linux, but is used by the HURD to reference the inode of a program which will be used to interpret this object. Most of the remaining reserved fields have been used up for both Linux and the HURD for larger owner and group fields, The HURD also has a larger mode field so it uses another of the remaining fields to store the extra bits.

There are pointers to the first 12 blocks which contain the file's data in the inode. There is a pointer to an indirect block (which contains pointers to the next set of blocks), a pointer to a doubly-indirect block (which contains pointers to indirect blocks) and a pointer to a trebly-indirect block (which contains pointers to doubly-indirect blocks).

Some filesystem specific behaviour flags are also stored and allow for specific filesystem behaviour on a per-file basis. There are flags for secure deletion, undeletable, compression, synchronous updates, immutability, append-only, dumpable, no-atime, indexed directories, and data-journaling.

Many of the filesystem specific behaviour flags, like journaling, have been implemented in newer filesystems like Ext3 and Ext4, while some other are still under development.

All the inodes are stored in inode tables, with one inode table per block group.

2.5. Superblocks

This definition comes from the Linux Kernel Documentation with some minor alterations.

The superblock contains all the information about the configuration of the filesystem. The information in the superblock contains fields such as the total number of inodes and blocks in the filesystem and how many are free, how many inodes and blocks are in each block group, when the filesystem was mounted (and if it was cleanly unmounted), when it was modified, what version of the filesystem it is and which OS created it.

The primary copy of the superblock is stored at an offset of 1024 bytes from the start of the device, and it is essential to mounting the filesystem. Since it is so important, backup copies of the superblock are stored in block groups throughout the filesystem.

The first version of ext2 (revision 0) stores a copy at the start of every block group, along with backups of the group descriptor block(s). Because this can consume a considerable amount of space for large filesystems, later revisions can optionally reduce the number of backup copies by only putting backups in specific groups (this is the sparse superblock feature). The groups chosen are 0, 1 and powers of 3, 5 and 7.

Revision 1 and higher of the filesystem also store extra fields, such as a volume name, a unique identification number, the inode size, and space for optional filesystem features to store configuration info.

All fields in the superblock (as in all other ext2 structures) are stored on the disc in little endian format, so a filesystem is portable between machines without having to know what machine it was created on.

2.6. Symbolic Links

This definition comes from Wikipedia.org with some minor alterations.

A symbolic link (also symlink or soft link) is a special type of file that contains a reference to another file or directory in the form of an absolute or relative path and that affects pathname resolution.

Symbolic links operate transparently for most operations: programs which read or write to files named by a symbolic link will behave as if operating directly on the target file. However, programs that need to handle symbolic links specially (e.g., backup utilities) may identify and manipulate them directly.

A symbolic link merely contains a text string that is interpreted and followed by the operating system as a path to another file or directory. It is a file on its own and can exist independently of its target. The symbolic links do not affect an inode link count. If a symbolic link is deleted, its target remains unaffected. If the target is moved, renamed or deleted, any symbolic link that used to point to it continues to exist but now points to a non-existing file. Symbolic links pointing to non-existing files are sometimes called "orphaned" or "dangling".

Symbolic links are also filesystem objects with inodes. For all symlink shorter than 60 bytes long, the data is stored within the inode itself; it uses the fields which would normally be used to store the pointers to data blocks. This is a worthwhile optimisation as it we avoid allocating a full block for the symlink, and most symlinks are less than 60 characters long.

Symbolic links can also point to files or directories of other partitions and file systems.

Chapter 3. Disk Organization

An Ext2 file systems starts with a superblock located at byte offset 1024 from the start of the volume. This is block 1 for a 1KiB block formatted volume or within block 0 for larger block sizes. Note that the size of the superblock is constant regardless of the block size.

On the next block(s) following the superblock, is the Block Group Descriptor Table; which provides an overview of how the volume is split into block groups and where to find the inode bitmap, the block bitmap, and the inode table for each block group.

In revision 0 of Ext2, each block group consists of a copy superblock, a copy of the block group descriptor table, a block bitmap, an inode bitmap, an inode table, and data blocks.

With the introduction of revision 1 and the sparse superblock feature in Ext2, only specific block groups contain copies of the superblock and block group descriptor table. All block groups still contain the block bitmap, inode bitmap, inode table, and data blocks. The shadow copies of the superblock can be located in block groups 0, 1 and powers of 3, 5 and 7.

The block bitmap and inode bitmap are limited to 1 block each per block group, so the total blocks per block group is therefore limited. (More information in the Block Size Impact table).

Each data block may also be further divided into "fragments". As of Linux 2.6.28, support for fragment was still not implemented in the kernel; it is therefore suggested to ensure the fragment size is equal to the block size so as to maintain compatibility.

Table 3-1. Sample Floppy Disk Layout, 1KiB blocks

Block Offset	Length	Description
byte 0	512 bytes	boot record (if present)
byte 512	512 bytes	additional boot record data (if present)
-- block group 0, blocks 1 to 1439 --		
byte 1024	1024 bytes	superblock
block 2	1 block	block group descriptor table
block 3	1 block	block bitmap
block 4	1 block	inode bitmap
block 5	23 blocks	inode table
block 28	1412 blocks	data blocks

For the curious, block 0 always points to the first sector of the disk or partition and will always contain the boot record if one is present.

The superblock is always located at byte offset 1024 from the start of the disk or partition. In a 1KiB block-size formatted file system, this is block 1, but it will always be block 0 (at 1024 bytes within block 0) in larger block size file systems.

And here's the organisation of a 20MB ext2 file system, using 1KiB blocks:

Table 3-2. Sample 20mb Partition Layout

Block Offset	Length	Description
byte 0	512 bytes	boot record (if present)
byte 512	512 bytes	additional boot record data (if present)
-- block group 0, blocks 1 to 8192 --		
byte 1024	1024 bytes	superblock
block 2	1 block	block group descriptor table
block 3	1 block	block bitmap
block 4	1 block	inode bitmap
block 5	214 blocks	inode table
block 219	7974 blocks	data blocks
-- block group 1, blocks 8193 to 16384 --		
block 8193	1 block	superblock backup
block 8194	1 block	block group descriptor table backup
block 8195	1 block	block bitmap
block 8196	1 block	inode bitmap
block 8197	214 blocks	inode table
block 8408	7974 blocks	data blocks
-- block group 2, blocks 16385 to 24576 --		
block 16385	1 block	block bitmap
block 16386	1 block	inode bitmap
block 16387	214 blocks	inode table
block 16601	3879 blocks	data blocks

The layout on disk is very predictable as long as you know a few basic information; block size, blocks per group, inodes per group. This information is all located in, or can be computed from, the `superblock` structure.

Nevertheless, unless the image was crafted with controlled parameters, the position of the various structures on disk (except the superblock) should never be assumed. Always load the superblock first.

Notice how block 0 is not part of the block group 0 in 1KiB block size file systems. The reason for this is block group 0 always starts with the block containing the superblock. Hence, on 1KiB block systems, block group 0 starts at block 1, but on larger block sizes it starts on block 0. For more information, see the s_first_data_block superblock entry.

3.1. Superblock

The superblock is always located at byte offset 1024 from the beginning of the file, block device or partition formatted with Ext2 and later variants (Ext3, Ext4).

Its structure is mostly constant from Ext2 to Ext3 and Ext4 with only some minor changes.

Table 3-3. Superblock Structure

Offset (bytes)	Size (bytes)	Description
0	4	s_inodes_count
4	4	s_blocks_count
8	4	s_r_blocks_count
12	4	s_free_blocks_count
16	4	s_free_inodes_count
20	4	s_first_data_block
24	4	s_log_block_size
28	4	s_log_frag_size
32	4	s_blocks_per_group
36	4	s_frags_per_group
40	4	s_inodes_per_group
44	4	s_mtime
48	4	s_wtime
52	2	s_mnt_count
54	2	s_max_mnt_count
56	2	s_magic
58	2	s_state
60	2	s_errors
62	2	s_minor_rev_level
64	4	s_lastcheck
68	4	s_checkinterval
72	4	s_creator_os
76	4	s_rev_level
80	2	s_def_resuid
82	2	s_def_resgid
-- EXT2_DYNAMIC_REV Specific --		
84	4	s_first_ino
88	2	s_inode_size
90	2	s_block_group_nr
92	4	s_feature_compat
96	4	s_feature_incompat
100	4	s_feature_ro_compat
104	16	s_uuid
120	16	s_volume_name
136	64	s_last_mounted
200	4	s_algo_bitmap
-- Performance Hints --		
204	1	s_prealloc_blocks
205	1	s_prealloc_dir_blocks
206	2	(alignment)
-- Journaling Support --		
208	16	s_journal_uuid
224	4	s_journal_inum
228	4	s_journal_dev
232	4	s_last_orphan
-- Directory Indexing Support --		
236	4 x 4	s_hash_seed
252	1	s_def_hash_version
253	3	padding - reserved for future expansion
-- Other options --		
256	4	s_default_mount_options
260	4	s_first_meta_bg
264	760	Unused - reserved for future revisions

3.1.1. s_inodes_count

32bit value indicating the total number of inodes, both used and free, in the file system. This value must be lower or equal to (s_inodes_per_group * number of block groups). It must be equal to the sum of the inodes defined in each block group.

3.1.2. s_blocks_count

32bit value indicating the total number of blocks in the system including all used, free and reserved. This value must be lower or equal to (s_blocks_per_group * number of block groups). It must be equal to the sum of the blocks defined in each block group.

3.1.3. s_r_blocks_count

32bit value indicating the total number of blocks reserved for the usage of the super user. This is most useful if for some reason a user, maliciously or not, fill the file system to capacity; the super user will have this specified amount of free blocks at his disposal so he can edit and save configuration files.

3.1.4. s_free_blocks_count

32bit value indicating the total number of free blocks, including the number of reserved blocks (see s_r_blocks_count). This is a sum of all free blocks of all the block groups.

3.1.5. s_free_inodes_count

32bit value indicating the total number of free inodes. This is a sum of all free inodes of all the block groups.

3.1.6. s_first_data_block

32bit value identifying the first data block, in other word the id of the block containing the `superblock` structure.

Note that this value is always 0 for file systems with a block size larger than 1KB, and always 1 for file systems with a block size of 1KB. The `superblock` is *always* starting at the 1024th byte of the disk, which normally happens to be the first byte of the 3rd sector.

3.1.7. s_log_block_size

The block size is computed using this 32bit value as the number of bits to shift left the value 1024. This value may only be positive.

```
block size = 1024 << s_log_block_size;
```

Common block sizes include 1KiB, 2KiB, 4KiB and 8Kib. For information about the impact of selecting a block size, see Impact of Block Sizes.

In Linux, at least up to 2.6.28, the block size must be at least as large as the sector size of the block device, and cannot be larger than the supported memory page of the architecture.

3.1.8. s_log_frag_size

The fragment size is computed using this 32bit value as the number of bits to shift left the value 1024. Note that a negative value would shift the bit right rather than left.

```
if( positive )
  fragmnet size = 1024 << s_log_frag_size;
else
  framgnet size = 1024 >> -s_log_frag_size;
```

As of Linux 2.6.28 no support exists for an Ext2 partition with fragment size smaller than the block size, as this feature seems to not be available.

3.1.9. s_blocks_per_group

32bit value indicating the total number of blocks per group. This value in combination with s_first_data_block can be used to determine the block groups boundaries.

3.1.10. s_frags_per_group

32bit value indicating the total number of fragments per group. It is also used to determine the size of the `block bitmap` of each block group.

3.1.11. s_inodes_per_group

32bit value indicating the total number of inodes per group. This is also used to determine the size of the `inode bitmap` of each block group. Note that you cannot have more than (block size in bytes * 8) inodes per group as the inode bitmap must fit within a single block. This value must be a perfect multiple of the number of inodes that can fit in a block ((1024<<s_log_block_size)/s_inode_size).

3.1.12. s_mtime

Unix time, as defined by POSIX, of the last time the file system was mounted.

3.1.13. s_wtime

Unix time, as defined by POSIX, of the last write access to the file system.

3.1.14. s_mnt_count

32bit value indicating how many time the file system was mounted since the last time it was fully verified.

3.1.15. s_max_mnt_count

32bit value indicating the maximum number of times that the file system may be mounted before a full check is performed.

3.1.16. s_magic

16bit value identifying the file system as Ext2. The value is currently fixed to `EXT2_SUPER_MAGIC` of value 0xEF53.

3.1.17. s_state

16bit value indicating the file system state. When the file system is mounted, this state is set to `EXT2_ERROR_FS`. After the file system was cleanly unmounted, this value is set to `EXT2_VALID_FS`.

When mounting the file system, if a valid of `EXT2_ERROR_FS` is encountered it means the file system was not cleanly unmounted and most likely contain errors that will need to be fixed. Typically under Linux this means running fsck.

Table 3-4. Defined s_state Values

Constant Name	Value	Description
EXT2_VALID_FS	1	Unmounted cleanly
EXT2_ERROR_FS	2	Errors detected

3.1.18. s_errors

16bit value indicating what the file system driver should do when an error is detected. The following values have been defined:

Table 3-5. Defined s_errors Values

Constant Name	Value	Description
EXT2_ERRORS_CONTINUE	1	continue as if nothing happened
EXT2_ERRORS_RO	2	remount read-only
EXT2_ERRORS_PANIC	3	cause a kernel panic

3.1.19. s_minor_rev_level

16bit value identifying the minor revision level within its revision level.

3.1.20. s_lastcheck

Unix time, as defined by POSIX, of the last file system check.

3.1.21. s_checkinterval

Maximum Unix time interval, as defined by POSIX, allowed between file system checks.

3.1.22. s_creator_os

32bit identifier of the os that created the file system. Defined values are:

Table 3-6. Defined s_creator_os Values

Constant Name	Value	Description
EXT2_OS_LINUX	0	Linux
EXT2_OS_HURD	1	GNU HURD
EXT2_OS_MASIX	2	MASIX
EXT2_OS_FREEBSD	3	FreeBSD
EXT2_OS_LITES	4	Lites

3.1.23. s_rev_level

32bit revision level value.

Table 3-7. Defined s_rev_level Values

Constant Name	Value	Description
EXT2_GOOD_OLD_REV	0	Revision 0
EXT2_DYNAMIC_REV	1	Revision 1 with variable inode sizes, extended attributes, etc.

3.1.24. s_def_resuid

16bit value used as the default user id for reserved blocks.

In Linux this defaults to `EXT2_DEF_RESUID` of 0.

3.1.25. s_def_resgid

16bit value used as the default group id for reserved blocks.

In Linux this defaults to `EXT2_DEF_RESGID` of 0.

3.1.26. s_first_ino

32bit value used as index to the first inode useable for standard files. In revision 0, the first non-reserved inode is fixed to 11 (`EXT2_GOOD_OLD_FIRST_INO`). In revision 1 and later this value may be set to any value.

3.1.27. s_inode_size

16bit value indicating the size of the inode structure. In revision 0, this value is always 128 (`EXT2_GOOD_OLD_INODE_SIZE`). In revision 1 and later, this value must be a perfect power of 2 and must be smaller or equal to the block size (1<<s_log_block_size).

3.1.28. s_block_group_nr

16bit value used to indicate the block group number hosting this superblock structure. This can be used to rebuild the file system from any superblock backup.

3.1.29. s_feature_compat

32bit bitmask of compatible features. The file system implementation is free to support them or not without risk of damaging the meta-data.

Table 3-8. Defined s_feature_compat Values

Constant Name	Value	Description
EXT2_FEATURE_COMPAT_DIR_PREALLOC	0x0001	Block pre-allocation for new directories
EXT2_FEATURE_COMPAT_IMAGIC_INODES	0x0002	
EXT3_FEATURE_COMPAT_HAS_JOURNAL	0x0004	An Ext3 journal exists
EXT2_FEATURE_COMPAT_EXT_ATTR	0x0008	Extended inode attributes are present
EXT2_FEATURE_COMPAT_RESIZE_INO	0x0010	Non-standard inode size used
EXT2_FEATURE_COMPAT_DIR_INDEX	0x0020	Directory indexing (HTree)

3.1.30. s_feature_incompat

32bit bitmask of incompatible features. The file system implementation should refuse to mount the file system if any of the indicated feature is unsupported.

An implementation not supporting these features would be unable to properly use the file system. For example, if compression is being used and an executable file would be unusable after being read from the disk if the system does not know how to uncompress it.

Table 3-9. Defined s_feature_incompat Values

Constant Name	Value	Description
EXT2_FEATURE_INCOMPAT_COMPRESSION	0x0001	Disk/File compression is used
EXT2_FEATURE_INCOMPAT_FILETYPE	0x0002	
EXT3_FEATURE_INCOMPAT_RECOVER	0x0004	
EXT3_FEATURE_INCOMPAT_JOURNAL_DEV	0x0008	
EXT2_FEATURE_INCOMPAT_META_BG	0x0010	

3.1.31. s_feature_ro_compat

32bit bitmask of "read-only" features. The file system implementation should mount as read-only if any of the indicated feature is unsupported.

Table 3-10. Defined s_feature_ro_compat Values

Constant Name	Value	Description
EXT2_FEATURE_RO_COMPAT_SPARSE_SUPER	0x0001	Sparse Superblock
EXT2_FEATURE_RO_COMPAT_LARGE_FILE	0x0002	Large file support, 64-bit file size
EXT2_FEATURE_RO_COMPAT_BTREE_DIR	0x0004	Binary tree sorted directory files

3.1.32. s_uuid

128bit value used as the volume id. This should, as much as possible, be unique for each file system formatted.

3.1.33. s_volume_name

16 bytes volume name, mostly unused. A valid volume name would consist of only ISO-Latin-1 characters and be 0 terminated.

3.1.34. s_last_mounted

64 bytes directory path where the file system was last mounted. While not normally used, it could serve for auto-finding the mountpoint when not indicated on the command line. Again the path should be zero terminated for compatibility reasons. Valid path is constructed from ISO-Latin-1 characters.

3.1.35. s_algo_bitmap

32bit value used by compression algorithms to determine the compression method(s) used.

Compression is supported in Linux 2.4 and 2.6 via the e2compr patch. For more information, visit http://e2compr.sourceforge.net/

Table 3-11. Defined s_algo_bitmap Values

Constant Name	Bit Number	Description
EXT2_LZV1_ALG	0	Binary value of 0x00000001
EXT2_LZRW3A_ALG	1	Binary value of 0x00000002
EXT2_GZIP_ALG	2	Binary value of 0x00000004
EXT2_BZIP2_ALG	3	Binary value of 0x00000008
EXT2_LZO_ALG	4	Binary value of 0x00000010

3.1.36. s_prealloc_blocks

8-bit value representing the number of blocks the implementation should attempt to pre-allocate when creating a new regular file.

Linux 2.6.28 will only perform pre-allocation using Ext4 although no problem is expected if any version of Linux encounters a file with more blocks present than required.

3.1.37. s_prealloc_dir_blocks

8-bit value representing the number of blocks the implementation should attempt to pre-allocate when creating a new directory.

Linux 2.6.28 will only perform pre-allocation using Ext4 and only if the `EXT4_FEATURE_COMPAT_DIR_PREALLOC` flag is present. Since Linux does not de-allocate blocks from directories after they were allocated, it should be safe to perform pre-allocation and maintain compatibility with Linux.

3.1.38. s_journal_uuid

16-byte value containing the uuid of the journal superblock. See Ext3 Journaling for more information.

3.1.39. s_journal_inum

32-bit inode number of the journal file. See Ext3 Journaling for more information.

3.1.40. s_journal_dev

32-bit device number of the journal file. See Ext3 Journaling for more information.

3.1.41. s_last_orphan

32-bit inode number, pointing to the first inode in the list of inodes to delete. See Ext3 Journaling for more information.

3.1.42. s_hash_seed

An array of 4 32bit values containing the seeds used for the hash algorithm for directory indexing.

3.1.43. s_def_hash_version

An 8bit value containing the default hash version used for directory indexing.

3.1.44. s_default_mount_options

A 32bit value containing the default mount options for this file system. TODO: Add more information here!

3.1.45. s_first_meta_bg

A 32bit value indicating the block group ID of the first meta block group. TODO: Research if this is an Ext3-only extension.

3.2. Block Group Descriptor Table

The block group descriptor table is an array of block group descriptor, used to define parameters of all the block groups. It provides the location of the inode bitmap and inode table, block bitmap, number of free blocks and inodes, and some other useful information.

The block group descriptor table starts on the first block following the superblock. This would be the third block on a 1KiB block file system, or the second block for 2KiB and larger block file systems. Shadow copies of the block group descriptor table are also stored with every copy of the superblock.

Depending on how many block groups are defined, this table can require multiple blocks of storage. Always refer to the superblock in case of doubt.

The layout of a block group descriptor is as follows:

Table 3-12. Block Group Descriptor Structure

Offset (bytes)	Size (bytes)	Description
0	4	bg_block_bitmap
4	4	bg_inode_bitmap
8	4	bg_inode_table
12	2	bg_free_blocks_count
14	2	bg_free_inodes_count
16	2	bg_used_dirs_count
18	2	bg_pad
20	12	bg_reserved

For each block group in the file system, such a `group_desc` is created. Each represent a single block group within the file system and the information within any one of them is pertinent only to the group it is describing. Every block group descriptor table contains all the information about all the block groups.

NOTE: All indicated "block id" are absolute.

3.2.1. bg_block_bitmap

32bit block id of the first block of the "block bitmap" for the group represented.

The actual block bitmap is located within its own allocated blocks starting at the block ID specified by this value.

3.2.2. bg_inode_bitmap

32bit block id of the first block of the "inode bitmap" for the group represented.

3.2.3. bg_inode_table

32bit block id of the first block of the "inode table" for the group represented.

3.2.4. bg_free_blocks_count

16bit value indicating the total number of free blocks for the represented group.

3.2.5. bg_free_inodes_count

16bit value indicating the total number of free inodes for the represented group.

3.2.6. bg_used_dirs_count

16bit value indicating the number of inodes allocated to directories for the represented group.

3.2.7. bg_pad

16bit value used for padding the structure on a 32bit boundary.

3.2.8. bg_reserved

12 bytes of reserved space for future revisions.

3.3. Block Bitmap

On small file systems, the "Block Bitmap" is normally located at the first block, or second block if a superblock backup is present, of each block group. Its official location can be determined by reading the "bg_block_bitmap" in its associated group descriptor.

Each bit represent the current state of a block within that block group, where 1 means "used" and 0 "free/available". The first block of this block group is represented by bit 0 of byte 0, the second by bit 1 of byte 0. The 8th block is represented by bit 7 (most significant bit) of byte 0 while the 9th block is represented by bit 0 (least significant bit) of byte 1.

3.4. Inode Bitmap

The "Inode Bitmap" works in a similar way as the "Block Bitmap", difference being in each bit representing an inode in the "Inode Table" rather than a block.

There is one inode bitmap per group and its location may be determined by reading the "bg_inode_bitmap" in its associated group descriptor.

When the inode table is created, all the reserved inodes are marked as used. In revision 0 this is the first 11 inodes.

3.5. Inode Table

The inode table is used to keep track of every directory, regular file, symbolic link, or special file; their location, size, type and access rights are all stored in inodes. There is no filename stored in the inode itself, names are contained in directory files only.

There is one inode table per block group and it can be located by reading the bg_inode_table in its associated group descriptor. There are s_inodes_per_group inodes per table.

Each inode contain the information about a single physical file on the system. A file can be a directory, a socket, a buffer, character or block device, symbolic link or a regular file. So an inode can be seen as a block of information related to an entity, describing its location on disk, its size and its owner. An inode looks like this:

Table 3-13. Inode Structure

Offset (bytes)	Size (bytes)	Description
0	2	i_mode
2	2	i_uid
4	4	i_size
8	4	i_atime
12	4	i_ctime
16	4	i_mtime
20	4	i_dtime
24	2	i_gid
26	2	i_links_count
28	4	i_blocks
32	4	i_flags
36	4	i_osd1
40	15 x 4	i_block
100	4	i_generation
104	4	i_file_acl
108	4	i_dir_acl
112	4	i_faddr
116	12	i_osd2

The first few entries of the inode tables are reserved. In revision 0 there are 11 entries reserved while in revision 1 (EXT2_DYNAMIC_REV) and later the number of reserved inodes entries is specified in the s_first_ino of the superblock structure. Here's a listing of the known reserved inode entries:

Table 3-14. Defined Reserved Inodes

Constant Name	Value	Description
EXT2_BAD_INO	1	bad blocks inode
EXT2_ROOT_INO	2	root directory inode
EXT2_ACL_IDX_INO	3	ACL index inode (deprecated?)
EXT2_ACL_DATA_INO	4	ACL data inode (deprecated?)
EXT2_BOOT_LOADER_INO	5	boot loader inode
EXT2_UNDEL_DIR_INO	6	undelete directory inode

3.5.1. i_mode

16bit value used to indicate the format of the described file and the access rights. Here are the possible values, which can be combined in various ways:

Table 3-15. Defined i_mode Values

Constant	Value	Description
-- file format --		
EXT2_S_IFSOCK	0xC000	socket
EXT2_S_IFLNK	0xA000	symbolic link
EXT2_S_IFREG	0x8000	regular file
EXT2_S_IFBLK	0x6000	block device
EXT2_S_IFDIR	0x4000	directory
EXT2_S_IFCHR	0x2000	character device
EXT2_S_IFIFO	0x1000	fifo
-- process execution user/group override --		
EXT2_S_ISUID	0x0800	Set process User ID
EXT2_S_ISGID	0x0400	Set process Group ID
EXT2_S_ISVTX	0x0200	sticky bit
-- access rights --		
EXT2_S_IRUSR	0x0100	user read
EXT2_S_IWUSR	0x0080	user write
EXT2_S_IXUSR	0x0040	user execute
EXT2_S_IRGRP	0x0020	group read
EXT2_S_IWGRP	0x0010	group write
EXT2_S_IXGRP	0x0008	group execute
EXT2_S_IROTH	0x0004	others read
EXT2_S_IWOTH	0x0002	others write
EXT2_S_IXOTH	0x0001	others execute

3.5.2. i_uid

16bit user id associated with the file.

3.5.3. i_size

In revision 0, (signed) 32bit value indicating the size of the file in bytes. In revision 1 and later revisions, and only for regular files, this represents the lower 32-bit of the file size; the upper 32-bit is located in the i_dir_acl.

3.5.4. i_atime

32bit value representing the number of seconds since january 1st 1970 of the last time this inode was accessed.

3.5.5. i_ctime

32bit value representing the number of seconds since january 1st 1970, of when the inode was created.

3.5.6. i_mtime

32bit value representing the number of seconds since january 1st 1970, of the last time this inode was modified.

3.5.7. i_dtime

32bit value representing the number of seconds since january 1st 1970, of when the inode was deleted.

3.5.8. i_gid

16bit value of the POSIX group having access to this file.

3.5.9. i_links_count

16bit value indicating how many times this particular inode is linked (referred to). Most files will have a link count of 1. Files with hard links pointing to them will have an additional count for each hard link.

Symbolic links do not affect the link count of an inode. When the link count reaches 0 the inode and all its associated blocks are freed.

3.5.10. i_blocks

32-bit value representing the total number of 512-bytes blocks reserved to contain the data of this inode, regardless if these blocks are used or not. The block numbers of these reserved blocks are contained in the i_block array.

Since this value represents 512-byte blocks and not file system blocks, this value should not be directly used as an index to the i_block array. Rather, the maximum index of the i_block array should be computed from i_blocks / ((1024<<s_log_block_size)/512), or once simplified, i_blocks/(2<<s_log_block_size).

3.5.11. i_flags

32bit value indicating how the ext2 implementation should behave when accessing the data for this inode.

Table 3-16. Defined i_flags Values

Constant Name	Value	Description
EXT2_SECRM_FL	0x00000001	secure deletion
EXT2_UNRM_FL	0x00000002	record for undelete
EXT2_COMPR_FL	0x00000004	compressed file
EXT2_SYNC_FL	0x00000008	synchronous updates
EXT2_IMMUTABLE_FL	0x00000010	immutable file
EXT2_APPEND_FL	0x00000020	append only
EXT2_NODUMP_FL	0x00000040	do not dump/delete file
EXT2_NOATIME_FL	0x00000080	do not update .i_atime
-- Reserved for compression usage --		
EXT2_DIRTY_FL	0x00000100	Dirty (modified)
EXT2_COMPRBLK_FL	0x00000200	compressed blocks
EXT2_NOCOMPR_FL	0x00000400	access raw compressed data
EXT2_ECOMPR_FL	0x00000800	compression error
-- End of compression flags --		
EXT2_BTREE_FL	0x00001000	b-tree format directory
EXT2_INDEX_FL	0x00001000	hash indexed directory
EXT2_IMAGIC_FL	0x00002000	AFS directory
EXT3_JOURNAL_DATA_FL	0x00004000	journal file data
EXT2_RESERVED_FL	0x80000000	reserved for ext2 library

3.5.12. i_osd1

32bit OS dependant value.

3.5.12.1. Hurd

32bit value labeled as "translator".

3.5.12.2. Linux

32bit value currently reserved.

3.5.12.3. Masix

32bit value currently reserved.

3.5.13. i_block

15 x 32bit block numbers pointing to the blocks containing the data for this inode. The first 12 blocks are direct blocks. The 13th entry in this array is the block number of the first indirect block; which is a block containing an array of block ID containing the data. Therefore, the 13th block of the file will be the first block ID contained in the indirect block. With a 1KiB block size, blocks 13 to 268 of the file data are contained in this indirect block.

The 14th entry in this array is the block number of the first doubly-indirect block; which is a block containing an array of indirect block IDs, with each of those indirect blocks containing an array of blocks containing the data. In a 1KiB block size, there would be 256 indirect blocks per doubly-indirect block, with 256 direct blocks per indirect block for a total of 65536 blocks per doubly-indirect block.

The 15th entry in this array is the block number of the triply-indirect block; which is a block containing an array of doubly-indirect block IDs, with each of those doubly-indirect block containing an array of indrect block, and each of those indirect block containing an array of direct block. In a 1KiB file system, this would be a total of 16777216 blocks per triply-indirect block.

A value of 0 in this array effectively terminates it with no further block being defined. All the remaining entries of the array should still be set to 0.

3.5.14. i_generation

32bit value used to indicate the file version (used by NFS).

3.5.15. i_file_acl

32bit value indicating the block number containing the extended attributes. In revision 0 this value is always 0.

Patches and implementation status of ACL under Linux can generally be found at http://acl.bestbits.at/

3.5.16. i_dir_acl

In revision 0 this 32bit value is always 0. In revision 1, for regular files this 32bit value contains the high 32 bits of the 64bit file size.

Linux sets this value to 0 if the file is not a regular file (i.e. block devices, directories, etc). In theory, this value could be set to point to a block containing extended attributes of the directory or special file.

3.5.17. i_faddr

32bit value indicating the location of the file fragment.

In Linux and GNU HURD, since fragments are unsupported this value is always 0. In Ext4 this value is now marked as obsolete.

In theory, this should contain the block number which hosts the actual fragment. The fragment number and its size would be contained in the i_osd2 structure.

3.5.18. Inode i_osd2 Structure

96bit OS dependant structure.

3.5.18.1. Hurd

Table 3-17. Inode i_osd2 Structure: Hurd

Offset (bytes)	Size (bytes)	Description
0	1	h_i_frag
1	1	h_i_fsize
2	2	h_i_mode_high
4	2	h_i_uid_high
6	2	h_i_gid_high
8	4	h_i_author

3.5.18.1.1. h_i_frag

8bit fragment number. Always 0 GNU HURD since fragments are not supported. Obsolete with Ext4.

3.5.18.1.2. h_i_fsize

8bit fragment size. Always 0 in GNU HURD since fragments are not supported. Obsolete with Ext4.

3.5.18.1.3. h_i_mode_high

High 16bit of the 32bit mode.

3.5.18.1.4. h_i_uid_high

High 16bit of user id.

3.5.18.1.5. h_i_gid_high

High 16bit of group id.

3.5.18.1.6. h_i_author

32bit user id of the assigned file author. If this value is set to -1, the POSIX user id will be used.

3.5.18.2. Linux

Table 3-18. Inode i_osd2 Structure: Linux

Offset (bytes)	Size (bytes)	Description
0	1	l_i_frag
1	1	l_i_fsize
2	2	reserved
4	2	l_i_uid_high
6	2	l_i_gid_high
8	4	reserved

3.5.18.2.1. l_i_frag

8bit fragment number.

Always 0 in Linux since fragments are not supported.

A new implementation of Ext2 should completely disregard this field if the i_faddr value is 0; in Ext4 this field is combined with l_i_fsize to become the high 16bit of the 48bit blocks count for the inode data.

3.5.18.2.2. l_i_fsize

8bit fragment size.

Always 0 in Linux since fragments are not supported.

A new implementation of Ext2 should completely disregard this field if the i_faddr value is 0; in Ext4 this field is combined with l_i_frag to become the high 16bit of the 48bit blocks count for the inode data.

3.5.18.2.3. l_i_uid_high

High 16bit of user id.

3.5.18.2.4. l_i_gid_high

High 16bit of group id.

3.5.18.3. Masix

Table 3-19. Inode i_osd2 Structure: Masix

Offset (bytes)	Size (bytes)	Description
0	1	m_i_frag
1	1	m_i_fsize
2	10	reserved

3.5.18.3.1. m_i_frag

8bit fragment number. Always 0 in Masix as framgents are not supported. Obsolete with Ext4.

3.5.18.3.2. m_i_fsize

8bit fragment size. Always 0 in Masix as fragments are not supported. Obsolete with Ext4.

3.6. Locating an Inode

Inodes are all numerically ordered. The "inode number" is an index in the inode table to an inode structure. The size of the inode table is fixed at format time; it is built to hold a maximum number of entries. Due to the large amount of entries created, the table is quite big and thus, it is split equally among all the block groups (see Chapter 3 for more information).

The s_inodes_per_group field in the superblock structure tells us how many inodes are defined per group. Knowing that inode 1 is the first inode defined in the inode table, one can use the following formulaes:

```
block group = (inode - 1) / s_inodes_per_group
```

Once the block is identified, the local inode index for the local inode table can be identified using:

```
local inode index = (inode - 1) % s_inodes_per_group
```

Here are a couple of sample values that could be used to test your implementation:

Table 3-20. Sample Inode Computations

Inode Number	Block Group Number	Local Inode Index
s_inodes_per_group = 1712		
1	0	0
963	0	962
1712	0	1711
1713	1	0
3424	1	1711
3425	2	0

As many of you are most likely already familiar with, an index of 0 means the first entry. The reason behind using 0 rather than 1 is that it can more easily be multiplied by the structure size to find the final byte offset of its location in memory or on disk.

Chapter 4. Directory Structure

Directories are used to hierarchically organize files. Each directory can contain other directories, regular files and special files.

Directories are stored as data block and referenced by an inode. They can be identified by the file type `EXT2_S_IFDIR` stored in the i_mode field of the inode structure.

The second entry of the Inode table contains the inode pointing to the data of the root directory; as defined by the `EXT2_ROOT_INO` constant.

In revision 0 directories could only be stored in a linked list. Revision 1 and later introduced indexed directories. The indexed directory is backward compatible with the linked list directory; this is achieved by inserting empty directory entry records to skip over the hash indexes.

4.1. Linked List Directory

A directory file is a linked list of directory entry structures. Each structure contains the name of the entry, the inode associated with the data of this entry, and the distance within the directory file to the next entry.

In revision 0, the type of the entry (file, directory, special file, etc) has to be looked up in the inode of the file. In revision 0.5 and later, the file type is also contained in the directory entry structure.

Table 4-1. Linked Directory Entry Structure

Offset (bytes)	Size (bytes)	Description
0	4	inode
4	2	rec_len
6	1	name_len[a]
7	1	file_type[b]
8	0-255	name

Notes:
a. Revision 0 of Ext2 used a 16bit `name_len`; since most implementations restricted filenames to a maximum of 255 characters this value was truncated to 8bit with the upper 8bit recycled as file_type.
b. Not available in revision 0; this field was part of the 16bit name_len field.

4.1.1. inode

32bit inode number of the file entry. A value of 0 indicate that the entry is not used.

4.1.2. rec_len

16bit unsigned displacement to the next directory entry from the start of the current directory entry. This field must have a value at least equal to the length of the current record.

The directory entries must be aligned on 4 bytes boundaries and there cannot be any directory entry spanning multiple data blocks. If an entry cannot completely fit in one block, it must be pushed to the next data block and the rec_len of the previous entry properly adjusted.

Since this value cannot be negative, when a file is removed the previous record within the block has to be modified to point to the next valid record within the block or to the end of the block when no other directory entry is present.

If the first entry within the block is removed, a blank record will be created and point to the next directory entry or to the end of the block.

4.1.3. name_len

8bit unsigned value indicating how many bytes of character data are contained in the name.

This value must never be larger than rec_len - 8. If the directory entry name is updated and cannot fit in the existing directory entry, the entry may have to be relocated in a new directory entry of sufficient size and possibly stored in a new data block.

4.1.4. file_type

8bit unsigned value used to indicate file type.

In revision 0, this field was the upper 8-bit of the then 16-bit name_len. Since all implementations still limited the file names to 255 characters this 8-bit value was always 0.

This value must match the inode type defined in the related inode entry.

Table 4-2. Defined Inode File Type Values

Constant Name	Value	Description
EXT2_FT_UNKNOWN	0	Unknown File Type
EXT2_FT_REG_FILE	1	Regular File
EXT2_FT_DIR	2	Directory File
EXT2_FT_CHRDEV	3	Character Device
EXT2_FT_BLKDEV	4	Block Device
EXT2_FT_FIFO	5	Buffer File
EXT2_FT_SOCK	6	Socket File
EXT2_FT_SYMLINK	7	Symbolic Link

4.1.5. name

Name of the entry. The ISO-Latin-1 character set is expected in most system. The name must be no longer than 255 bytes after encoding.

4.1.6. Sample Directory

Here's a sample of the home directory of one user on my system:

```
$ ls -1a ~
.
..
.bash_profile
.bashrc
mbox
public_html
tmp
```

For which the following data representation can be found on the storage device:

Table 4-3. Sample Linked Directory Data Layout, 4KiB blocks

Offset (bytes)	Size (bytes)	Description
Directory Entry 0		
0	4	inode number: 783362
4	2	record length: 12
6	1	name length: 1
7	1	file type: `EXT2_FT_DIR`=2
8	1	name: .
9	3	padding
Directory Entry 1		
12	4	inode number: 1109761
16	2	record length: 12
18	1	name length: 2
19	1	file type: `EXT2_FT_DIR`=2
20	2	name: ..
22	2	padding
Directory Entry 2		
24	4	inode number: 783364
28	2	record length: 24
30	1	name length: 13
31	1	file type: `EXT2_FT_REG_FILE`
32	13	name: .bash_profile
45	3	padding
Directory Entry 3		
48	4	inode number: 783363
52	2	record length: 16
54	1	name length: 7
55	1	file type: `EXT2_FT_REG_FILE`
56	7	name: .bashrc
63	1	padding
Directory Entry 4		
64	4	inode number: 783377
68	2	record length: 12
70	1	name length: 4
71	1	file type: `EXT2_FT_REG_FILE`
72	4	name: mbox
Directory Entry 5		
76	4	inode number: 783545
80	2	record length: 20
82	1	name length: 11
83	1	file type: `EXT2_FT_DIR`=2
84	11	name: public_html
95	1	padding
Directory Entry 6		
96	4	inode number: 669354
100	2	record length: 12
102	1	name length: 3
103	1	file type: `EXT2_FT_DIR`=2
104	3	name: tmp
107	1	padding
Directory Entry 7		
108	4	inode number: 0
112	2	record length: 3988
114	1	name length: 0
115	1	file type: `EXT2_FT_UNKNOWN`
116	0	name:
116	3980	padding

4.2. Indexed Directory Format

Using the standard linked list directory format can become very slow once the number of files starts growing. To improve performances in such a system, a hashed index is used, which allow to quickly locate the particular file searched.

Bit EXT2_INDEX_FL in the i_flags of the directory inode is set if the indexed directory format is used.

In order to maintain backward compatibility with older implementations, the indexed directory also maintains a linked directory format side-by-side. In case there's any discrepency between the indexed and linked directories, the linked directory is preferred.

This backward compatibility is achieved by placing a fake directory entries at the beginning of block 0 of the indexed directory data blocks. These fake entries are part of the dx_root structure and host the linked directory information for the "." and ".." folder entries.

Immediately following the Section 4.2.1 structure is an array of Section 4.2.2 up to the end of the data block or until all files have been indexed.

When the number of files to be indexed exceeds the number of Section 4.2.2 that can fit in a block (Section 4.2.2.3), a level of indirect indexes is created. An indirect index is another data block allocated to the directory inode that contains directory entries.

4.2.1. Indexed Directory Root

Table 4-4. Indexed Directory Root Structure

Offset (bytes)	Size (bytes)	Description
-- Linked Directory Entry: . --		
0	4	inode: this directory
4	2	rec_len: 12
6	1	name_len: 1
7	1	file_type: `EXT2_FT_DIR`=2
8	1	name: .
9	3	padding
-- Linked Directory Entry: .. --		
12	4	inode: parent directory
16	2	rec_len: (blocksize - this entry's length(12))
18	1	name_len: 2
19	1	file_type: `EXT2_FT_DIR`=2
20	2	name: ..
22	2	padding
-- Indexed Directory Root Information Structure --		
24	4	reserved, zero
28	1	hash_version
29	1	info_length
30	1	indirect_levels
31	1	reserved - unused flags

4.2.1.1. hash_version

8bit value representing the hash version used in this indexed directory.

Table 4-5. Defined Indexed Directory Hash Versions

Constant Name	Value	Description
DX_HASH_LEGACY	0	TODO: link to section
DX_HASH_HALF_MD4	1	TODO: link to section
DX_HASH_TEA	2	TODO: link to section

4.2.1.2. info_length

8bit length of the indexed directory information structure (dx_root); currently equal to 8.

4.2.1.3. indirect_levels

8bit value indicating how many indirect levels of indexing are present in this hash.

In Linux, as of 2.6.28, the maximum indirect levels value supported is 1.

4.2.2. Indexed Directory Entry

The indexed directory entries are used to quickly lookup the inode number associated with the hash of a filename. These entries are located immediately following the fake linked directory entry of the directory data blocks, or immediately following the Section 4.2.1.

The first indexed directory entry, rather than containing an actual hash and block number, contains the maximum number of indexed directory entries that can fit in the block and the actual number of indexed directory entries stored in the block. The format of this special entry is detailed in Table 4-7.

The other directory entries are sorted by hash value starting from the smallest to the largest numerical value.

Table 4-6. Indexed Directory Entry Structure (dx_entry)

Offset (bytes)	Size (bytes)	Description
0	4	hash
4	4	block

Table 4-7. Indexed Directory Entry Count and Limit Structure

Offset (bytes)	Size (bytes)	Description
0	2	limit
2	2	count

4.2.2.1. hash

32bit hash of the filename represented by this entry.

4.2.2.2. block

32bit block index of the directory inode data block containing the (linked) directory entry for the filename.

4.2.2.3. limit

16bit value representing the total number of indexed directory entries that fit within the block, after removing the other structures, but including the count/limit entry.

4.2.2.4. count

16bit value representing the total number of indexed directory entries present in the block. TODO: Research if this value includes the count/limit entry.

Network equipment like switches and gateways is provided by Cisco, Huawei, Ericsson, etc. The current research trends reflect the tremendous potential of the Fog.

The Fog features with location awareness, low latency and edge location [13]. It fits to a scenario where a huge number of heterogeneous ubiquitous and decentralized devices communicate, need to cooperate, and perform storage and processing tasks [6]. Users can visit their Fog anytime by using any device that can be connected to the Fog network. The Fog has many applications in such areas as smart city [14–16] and healthcare [17–20]. It can also provide better Quality of Service (QoS) in terms of fast response and small energy consumption [21,22].

The Fog uses network devices (named Fog nodes in this paper) for latency-aware processing of data collected from Internet of Thing (IoT) [23]. Fog nodes are denoted as heterogeneous components deployed in an edge network in Fog environments. They include gateways, routers, switchers, access points, base stations, and specific Fog servers [24]. The Fog facilitates uniform and seamless resource management including computation, networking and storage allocation [25]. Fog nodes are often the first set of processors that data encounter in IoT, and have the resources to implement a full hardware root of trust. This root of trust can be extended to all the processes and applications running on them, and then to the Cloud [26]. Without a hardware root of trust, various attack scenarios can compromise the software infrastructures of the Fog, allowing hackers to gain a foothold. The requirements of life safety-critical systems mandate the sorts of security capabilities available on the Fog [27]. Hence, new security and trust challenges emerge with the rise of the Fog. The existing methods cannot be directly applied to the Fog because of its mobility, heterogeneity, large-scale geo-distribution [12]. This work reviews these concerns in the Fog and the existing solutions. Differing from other survey papers about Fog computing, this paper focuses on its security and trust issues, especially in the region of the Fog.

The rest of this paper is organized as follows. Section 2 reveals a Fog architecture as well as related security and trust issues. Section 3 summarizes the related work to cope with security and trust issues. Section 4 presents open research problems. Section 5 discusses the future work. Finally, Section 6 concludes this survey paper.

2. Fog computing architecture

2.1. General architecture

Based on the modern computing architecture with three layers [11,21]: the Cloud, the Fog and the Edge, we provide a comprehensive fog architecture as shown in Fig. 1. Between the Cloud and the Fog lies a core network to offer network services. From it we can see that the Cloud lies at the upper core level and is far away from edge devices. The Fog lies at the middle level and is closer to edge devices than the Cloud. Each Fog node is connected to the Cloud. Each edge device is connected to a Fog node [28]. In addition, we can see that Fog nodes can be connected to each other. Communications between Fog–Fog, Fog–Cloud, and Fog–Edge are all bi-directional.

The Cloud: It includes high-performance servers and storage devices for broadcasting, data warehousing and big data analysis [19]. It is the remote control and management center that can store large data, and process highly complex but often non-urgent tasks. The data is sent to the Cloud through high-speed wireless or wired communications. The Cloud provides ultimate and global coverage. As a repository, it provides data storage to meet users' long-term needs and intelligent data analysis.

The Fog: It consists of a network of interconnected Fog nodes [19,24]. It provides geo-distributed, low latency and urgent computation as well as location awareness. Each Fog node is a resource

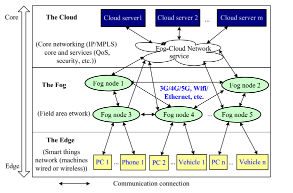


Fig. 1. A comprehensive Fog architecture.

center for ephemeral storage. Its functions include network transform, data collection, communications, data upload, data storage, computation and management. Compared with the edge devices, Fog nodes have more memory or storage ability for computing, which makes it possible to process a significant amount of data from edge devices. On the other hand, when needing a more complex and longtime computation, the computation work should be sent to the Cloud by Fog nodes through various available communications technologies, e.g., 3G/4G/5G cellular networks and WiFi. Fog nodes are bridges between Cloud and edge devices.

Fog nodes are independent and can be interconnected for collaboration. Management and collaborative procedures are applied on Fog nodes to implement management and control. The collaboration among Fog nodes can be executed via remote or local communications among them.

The Edge: It consists of several physical devices (edge devices) enabled with their ubiquitous identification, sensing, and communication capacity [19], such as vehicles, machines and cell phones. Each edge device is connected to one of the Fog nodes. Edge devices have a large variety of sensors and local data. It is very expensive and time-consuming to send all the data from terminal edge devices to the Cloud through a network. Hence, by connecting them to Fog nodes, one can deal with the urgent data but not transfer from edge devices to the cloud immediately.

Some easily misunderstood concepts are discussed in the following.

Edge computing vs. the Fog: Edge computing is different from the Fog in that the latter is a highly virtualized platform that provides computation, storage, and networking services between end devices and Cloud computing data centers [11]. Both of them need to push intelligence and processing capabilities out of centralized data centers and down closer to edge devices, such as IoT sensors, relays, and motors. The key difference between them is where intelligence and computing power are placed [12]. The Fog pushes intelligence down to the local area network (LAN) level, processing data in Fog nodes. While Edge computing pushes the intelligence, processing power, and communication capabilities further down to edge devices [29]. More details between them are discussed in [24]. In [30], Endpoint computing is regarded as Edge computing. Other similar concepts such as Cloudlets and Micro-data centers are discussed in [31].

Wireless sensor networks (WSNs) vs. the Fog: WSNs are designed to operate at very low power to extend battery life or use energy harvesting to sustain themselves. Most of them face the problems of small memory motes, low processing power, and

Future Generation Computer Systems 88 (2018) 16–27

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Security and trust issues in Fog computing: A survey

PeiYun Zhang [a], MengChu Zhou [b,c,*], Giancarlo Fortino [d]

[a] School of Computer and Information, Anhui Normal University, Wuhu 241003, China
[b] Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA
[c] The Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China
[d] Department of Informatics, Modeling, Electronics, and Systems, University of Calabria, Italy

H I G H L I G H T S

- We discuss and analyze the architectures of Fog computing, and indicate the related potential security and trust issues.
- We analyze how such issues have been tackled in the existing investigations.
- We indicate the open challenges, research trends and future topics of security and trust in Fog computing.

A R T I C L E I N F O

Article history:
Received 27 December 2017
Received in revised form 3 March 2018
Accepted 8 May 2018
Available online 15 May 2018

Keywords:
Fog computing
Cloud computing
Trust
Security

A B S T R A C T

Fog computing uses one or more collaborative end users or near-user edge devices to perform storage, communication, control, configuration, measurement and management functions. It can well solve latency and bandwidth limitation problems encountered by using cloud computing. First, this work discusses and analyzes the architectures of Fog computing, and indicates the related potential security and trust issues. Then, how such issues have been tackled in the existing literature is comprehensively reported. Finally, the open challenges, research trends and future topics of security and trust in Fog computing are discussed.

1. Introduction

Cloud computing (the Cloud in brief) has drastically changed the landscape of information technology (IT) by providing some major benefits to IT users, including eliminating upfront IT investment, scalability, proportional costs, and so on [1–5]. However, as more and more devices are connected, latency-sensitive applications seriously face the problem of large latency. In addition, Cloud computing is unable to meet the requirements of mobility support and location awareness. To overcome these problems, a new paradigm called Fog computing (the Fog in brief) was proposed in 2012 [6,7].

According to Bonomi et al. [8], the Fog is a highly virtualized platform that provides storage, computing and networking services between the Cloud data centers and end devices. Both Cloud and Fog provide data, computation, storage and application services to end users [9]. However, the latter is distinguished from the former by its decentralization, processing large amounts of data locally, software installation on heterogeneous hardware [10], proximity to end-users, dense geographical distribution, and support for mobility [11].

Here, we show an example of a traffic light system to discuss the relationship between them when dealing with latency. In a traffic light system without the Fog, there may be 3∼4 hops from the monitoring probe to the server in the Cloud. Hence, real-time decisions cannot be made immediately and the system faces the challenge of network latency. However, by using the Fog, the monitoring probe acts as a sensor, and the traffic lights act as an actuator. The Fog node can send conventional compressed video that may endure some time latency to the Cloud. When the Fog node detects an ambulance's headlight flashing, it makes an immediate decision to turn on the corresponding traffic lights, so as to let the ambulance go through without any delay. However, the Fog cannot replace the Cloud but supplements it.

Many companies and institutes, such as ARM, Cisco, Dell, Intel, Microsoft Corp., Cloudlet, Intelligent Edge by Intel and the Princeton University Edge Laboratory are devoted to research and development of the Fog. OpenFog (Found in 2015) Consortium workgroups are working towards creating an open architecture for the Fog to enable its interoperability and scalability [12].

* Corresponding author.
E-mail address: zhou@njit.edu (M.C. Zhou).

https://doi.org/10.1016/j.future.2018.05.008
0167-739X/© 2018 Published by Elsevier B.V.

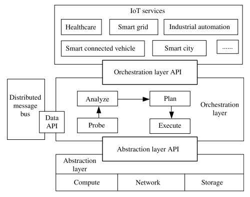


Fig. 2. The Fog architecture implementation [13].

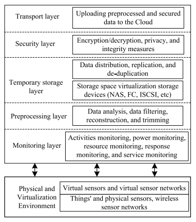

Transport layer
Uploading preprocessed and secured data to the Cloud
Security layer
Encryption/decryption, privacy, and integrity measures
Temporary storage layer
Data distribution, replication, and de-duplication
Storage space virtualization storage devices (NAS, FC, ISCSI, etc)
Preprocessing layer
Data analysis, data filtering, reconstruction, and trimming
Monitoring layer
Activities monitoring, power monitoring, resource monitoring, response monitoring, and service monitoring
Physical and Virtualization Environment
Virtual sensors and virtual sensor networks
Things' and physical sensors, wireless sensor networks

Fig. 3. The layered architecture of a gateway in the Fog [37].

sometimes unreliable sensors. The Fog is a suitable platform to support WSNs [11]. In addition, Romana et al. differentiate the features among Mobile Edge computing, Mobile Cloud computing, Fog and Cloud computing [32].

From Fig. 1 and the above analysis, we reveal the following security and trust issues for the Fog: Since the Fog is designed upon traditional networking components, it is highly vulnerable to security attacks. It is hard to ensure authenticated access to services and maintenance of privacy in a large Fog [28]. We can also see that the distributed and open nature of its hierarchical structure makes it vulnerable to security threats. Traditional security and trust issues, such as wiretapping, tampering, loss of information, and Trojan horses, still exist. The new ones caused by Fog characteristics are listed as follows:

- For an edge device, it may have to switch the connection to its nearby Fog node when faults happen. This may need to build a new connection and transfer data from the former Fog node to the new one, which may bring about more chance for new security when such connection fails.
- It also faces the crisis of re-locating a new Fog node for communications if its connected Fog node malfunctions when building a new connection from a Fog node to the Cloud.

2.2. Detailed architecture design

Based on the architecture in Fig. 1, the topological structures enable communications among Fog nodes and include net-like and bus topology as shown in [28] and [33], respectively. The former is more complex and may cause more communication overhead, security and trust risk. The shortcomings of bus topology include data collision, limited communication distance and limited range. It is also difficult to diagnose and isolate faults with it.

From the communication relationship among the Fog nodes, we reveal the following security and trust issues:

- Each Fog node is not only independent, but also can be collaborative with other ones through message passing. If Fog nodes fail to be well integrated into the network and work together to create meaningful services, trust problems may happen.
- When a new Fog node comes or an old one cannot provide its service and has to quit, the other Fog nodes potentially have to change their topology and rebuild their communication structure so as to enable communications among them. This may bring about new security issues as caused by such a topology rebuilding process.
- When Fog nodes collaborate with other nodes, if one node is attacked by malicious users and is infected, the infected one may attack or infect other nodes. This induces security or trust crisis among all the collaborative nodes.
- Fog nodes can communicate via a network. The communication overhead among them may be large. For latency-sensitive applications, it can be a serious problem. It will low the system reliability and then cause trust crisis for users when their task deadlines are missed. Hence, how to decrease the communication overhead to an acceptable level is a significant issue.

2.3. Architecture implementation

A three-layer Fog architecture [13] is realized as shown in Fig. 2. It has IoT services, orchestration layer and distributed message bus, and abstraction layer.

IoT services: The Fog platform hosts a set of diverse service applications, such as smart city. An IoT service is viewed as a way for users to access their needed functionality/application [34].

Orchestration layer and distributed message bus: Orchestration is the process through which the Fog systems determine how the virtualized resources are allocated and inter-operated. It is essential that orchestration is aware of the instantaneous state of the Fog network, and reacts quickly to any changes in its configuration, load, or status [27]. The core issues, challenges, and future research directions in fog-enabled orchestration for IoT services is overviewed in [35]. Because its services and infrastructure are distributive, the Fog should provide necessary means for distributed policy-based orchestration, which results in the scalable management of individual subsystems and the overall services. The orchestration functionality is distributed across the Fog deployment while distributed control provides better resiliency, higher scalability, and faster orchestration for geographically distributed deployments than centralized control [35]. The messaging bus is used for communications. The layer provides policy-based and dynamic life cycle management for Fog services. There are four parts in the life cycle: probing, analyzing, planning, and executing a task.

Abstraction layer: It provides uniformity in a heterogeneous infrastructure environment over various access, control and management of resources through customized Application Programming

Interfaces (APIs) and User Interfaces. Generic APIs concern security, isolation and privacy to various tenants.

In Fig. 2, from the Fog nodes themselves, we observe the following security and trust issues:

- New security and trust issues are raised because of service orchestration: It needs to specify how and what services must be chained before delivery. Highly trusted services should be chosen to execute the entire service.
- At the abstraction layer, security is related with isolation and privacy in multi-tenancy.

Because a gateway can be a Fog node in the Fog, which is different from the gateway in [36]. A five-layer architecture of the Fog gateway as a Fog node is shown in Fig. 3 [37]. Below the five layers are many things, sensors and WSNs in a physical and virtualization environment. The following five layers are on the top of the environment:

Monitoring layer: It monitors activities, power, responses, and services when performing tasks. Effective measures are taken in time.

Preprocessing layer: It analyzes, filters, reconstructs and trims data.

Temporary storage layer: It supports data distribution, replication and storage space virtualization. It stores preprocessed data in Fog nodes and passes them to the Cloud.

Security layer: Some private data may be generated via IoT devices and WSNs, such as ubiquitous health care data. Location-aware data sometimes is sensitive when security and privacy issues are concerned.

Transport layer: It uploads the ready-to-send data to the Cloud. After being uploaded, the data is removed from Fog nodes.

From Fig. 3, we find the following security and trust issues:

- Storage is needed in the Fog. Regarding the storage, some problems may be faced. For example, after an edge device sends data to its connected Fog node, what data is chosen to be stored in the Fog node and in the Cloud? If the urgent data is large and not permitted to transfer to the Cloud, how can we use the Fog node's orchestration to realize security and high efficient storage? A hybrid storage approach for IoT in PaaS cloud federation [38] is proposed. The issues about secure data storage and search for industrial IoT by integrating Fog Computing and Cloud Computing [39] are investigated.
- How can we realize cost-effectiveness and high resiliency against failures in crucial applications [30]?
- If a large job is needed to be processed at the preprocessing layer and is beyond the computation ability of the connected Fog node, how should it do? Some measures should be taken for it.
- Gateways serving as Fog nodes may be compromised or taken over by fake ones [40].

From the above analysis of the fog computing architecture, we can see many trust and security issues to be addressed. The related work is discussed next.

3. Trust and security in Fog computing

3.1. Existing surveys and their overview

We have retrieved 86 references about security and trust in the Fog, including eight survey papers. We summarize their covered security issues and characteristics in Table 1. These studies mainly focus on the security issues. Since their publications, we have seen some new security issues emerging, e.g., context-aware and data-dependent security ones. As a vitally important issue, trust has drawn much attention. Hence, this work intends to write a new survey paper about the security and trust issues related to Fog computing and its architectures.

We classify security and trust issues of current researches into six types: (1) trust, (2) authentication and access control, (3) attack, (4) privacy, (5) secure communications and (6) the others (include service availability, secure applications and secure sharing technology). According to these types, the number of the references is listed in Table 2 from 2014. In Table 2, the first column denotes the research topic types. The first row denotes years. The entries indicate the number of references per topic in a year. We can see that there are 10 papers in 2016 and 35 ones in 2017. The number of papers increases clearly. We note that only the papers in the first two months of 2018 are retrieved. As the classic security issues, authentication and access control as well as privacy preservation have drawn more researchers' attention. There are eight papers about the trust issue, which is relatively significant. There are only two papers about service availability, eight about secure applications and one about secure sharing technology.

3.2. Trust

Trust plays a major role in fostering relations based on previous interactions among Fog nodes and edge devices. A Fog node is considered as the most critical component as it is in charge of ensuring privacy and anonymity for end-users [18]. Moreover, this component must be trusted for delegation, as they must be assured that the Fog node implements the global concealing process on their released data and triggers non-malicious activities only. This requires all nodes that are part of the Fog network to have a certain level of trust on one another.

3.2.1. Trust middleware and model

Elmisery et al. propose a Fog-based middleware, where trust agents calculate the approximated interpersonal trust between a Fog node and the Cloud [18]. The trust computation is done in a decentralized fashion by using the entropy definition in [41]. The local concealment agent implements the local concealment process to achieve user privacy. The global concealment agent only exists in a Fog node. It executes the global concealment process on the aggregated user profile. A service layer is adopted in the Fog architecture to improve and manage trust [18], as shown in Fig. 4.

The difference between the middle parts in Figs. 4 and 1 is that a service layer is added on the top of the Fog in [18]. At the service layer, there are two main components: (1) Security authority center: This center is used to select Fog nodes with the highest reputation to act as members for trust computation. It is a trusted third party responsible for making an assessment on those Fog nodes according to the latter's reports. Moreover, it records the calculated reputation scores for various Fog nodes and trust levels for different Clouds. (2) Service server: A Fog node can seamlessly interact with the service server which is in the Cloud.

Soleymani et al. point out that trust establishment among vehicles is important to secure integrity and reliability of applications [46]. A secure trust model can deal with uncertainties and risks taking from unreliable information in vehicular environments. However, inaccurate, incomplete, and imprecise information collected by vehicles as well as movable/immovable obstacles yield interrupting effects. A fuzzy trust model based on experience and plausibility is proposed to secure a vehicular network [46]. It executes a series of security checks to ensure the correctness of the information received from authorized vehicles. Moreover, Fog nodes are adopted as a facility to evaluate the level of accuracy of

Table 1
The summary of security and trust issues in major survey papers.

Ref.	Security and trust issues in the Fog	Characteristics
[10]	(1) Virtualization issues (2) Web security issues (3) Internal/external communication issues (4) Data security related issues (5) Wireless security issues (6) Malware protection	Detailed definition and discussion of potential security issues of a Fog platform inherited from Cloud computing are given: (1) Access to services: persistent threats, access control issues, account hijacking and denial of services; (2) Platform or system: insecure APIs, system and application vulnerabilities, malicious insiders, insufficient due diligence, abuse and nefarious use, and shared technology issues; (3) Data: data loss and data breaches.
[12]	(1) Trust (2) Authentication (3) Secure communications in the Fog (4) End user's privacy (5) Malicious attacks	Challenges about the following issues are presented: (1) Malicious or malfunctioning in Fog nodes (2) Malicious insider attack (3) Mutual authentication (4) Privacy preservation (5) Fog forensics (6) Authentication and key agreement in fog computing-based radio access networks.
[42]	(1) Man-in-the-middle (MitM) Attack (2) Intrusion detection (3) Malicious detection technique in the Fog environments (4) Malicious Fog node problem (5) Data protection (6) Data management issues	Several unique security threats are introduced to the Fog.
[40]	(1) Trust issue (2) Preserving integrity (3) Logs-related issues (4) Dependability for data acquisition (5) Compliance issue (6) Live forensics issues (7) Multi-tenancy issues (8) Chain of custody	More extensive researches in Fog security and Fog forensics are promoted.
[28]	(1) Authentication (2) Privacy (3) Encryption (4) Denial of service (DoS) attack	Reliability in the Fog can be discussed in terms of consistency of Fog nodes, availability of high-performance services, secured interactions and fault tolerance [28].
[43]	(1) Network security (2) Data security (3) Access control (4) Privacy	Location privacy and data confidentiality are focused. The greatest risk comes from the outside attack.
[44,45]	(1) Intrusion detection (2) Authentication	Stealthy features of MitM attack is investigated by checking the CPU and memory consumption of a Fog node. Authentication of connection between the Fog and the Cloud.

Table 2
The summary of the references.

	2014	2015	2016	2017	2018	Total
Trust	1	0	3	5	0	9
Authentication and Access Control	2	0	0	5	3	10
Attacks	2	2	1	3	0	8
Privacy preservation	0	1	5	9	3	18
Secure communications	0	0	0	5	1	6
The others	0	0	1	8	2	11
Total	5	3	10	35	9	62

an event's location. Koo et al. present a hybrid secure deduplication protocol by taking untrustworthy Fog storage environments into account [47]. The work [48] introduces a trusted third party (TTP) into its privacy preservation. Because malicious attacks can cause sensor communications to be unreliable, a trust evaluation method is needed to ensure the reliability relationship among sensors to resist malicious attacks. Fog nodes are adopted to help the system compute trust values [49].

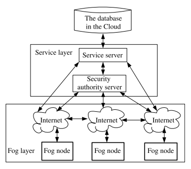


Fig. 4. Service middleware in the Fog [18].

3.2.2. Collusion deception

Wang et al. [50] study the trust scheme of a publish–subscribe system (PSS) [51] in the Fog, so as to defend trust against collusion attack. PSS has been widely applied in many modern large-scale critical systems, for example, traffic monitoring. A generic broker-based PSS is given in [52].

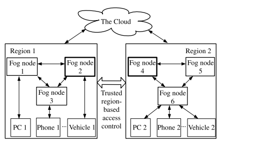


Fig. 5. A scenario of trusted region-based access control in the Fog [53].

The role of a broker is an important part of a PSS. Brokers communicate with different entities (e.g., publishers and subscribers), match the suitable user requirement and transmit users' data [52]. They can be used to decouple users' interaction and provide asynchronous communications. A malicious node (may be publisher or subscriber) who is supposed to keep the secret (the encryption key or content of data) of other nodes would deliberately leak the secret to the hostile brokers. Malicious brokers and nodes can collude with each other and share secret as follows:

- A malicious node provides other users' sensitive data to a malicious broker who can analyze these data, and
- The malicious broker provides other nodes' data to its colluders such that the latter could pretend as the most suitable candidate to other users. We can see that brokers may be malicious and the Fog may face collusion attacks. To decrease the security risks and vulnerabilities, the study [50] proposes content-based PSS with differential privacy in a Fog context to ensure the trustworthy function of publish–subscribe.

3.2.3. Region-based trust

There are numerous physical devices at different locations and with various communication types and connections structures in the Fog. Nevertheless, Fog nodes can provide local and regional computation to users for faster response. Therefore, how to accomplish these goals is one of future studies fitting to the Fog's characteristics. Dang et al. propose a region-based trust model for trust communications among Fog nodes at different regions [53]. The scenario of their trusted region-based access control in the Fog is shown in Fig. 5. A Fog node is selected to delegate computational resource management and task execution in a region. For example, node 2 in Region 1 and node 4 in Region 2 are selected as delegates, respectively. Delegate nodes are used to compute trust values for nodes in the same region. For example, if node 1 wants to obtain the trust value of node 3, it needs to obtain it via node 2 in Region 1. If node 1 wants to obtain the trust value of node 5, it needs to obtain it via node 4 in Region 2. They can also compute their region trust values and send them to the Cloud.

3.3. Access control

Based on different technologies for access control, we summarize the current related work into Table 3.

The detailed information about the work is summarized as follows.

3.3.1. Behavior profiling

Mandlekar et al. point out that unauthorized access needs to be detected and the real data should be saved without being hacked [54]. Hence, they exploit user behavior profiling and decoy

Table 3
The summary for access control in the Fog.

Related technology	Ref.
Behavior profiling	[54]
Attribute encryption	[33,55–61]
Certificate authority	[62]
Policy-based access control	[63]

information technology to compare the behavior of a user with that of normal users for user authentication.

3.3.2. Attribute encryption

Since the Fog originates from and is a non-trivial extension of Cloud computing, it inherits many security and privacy challenges of the latter. Commonly used encryption techniques can be used, e.g., Advanced Encryption Standard algorithm and Rivest Shamir Adleman algorithm [60].

Fan et al. point out that Ciphertext-policy attribute-based encryption (ABE) can help to achieve data access control in fog–cloud systems [59]. Hence, they propose an access control scheme based on a verifiable outsourced multi-authority.

Attribute-based cryptography is a well-known technology to guarantee data confidentiality and fine-grained data access control. The work [55] proposes a secure and fine-grained data access control scheme with ciphertext update and computation outsourcing in the Fog for IoT to decrease computational cost and realize secure data access control. The system consists of attribute authority, Cloud servers, Fog nodes and users. The Attribute Authority generates a public key for each Cloud server and Fog node. It also generates a secret key for each edge device from users. The communication data of Fog–Fog and Fog–Cloud is ciphertext, while it is partial ciphertext of Edge–Fog. Smart meters can encrypt and send the data to a Fog device, such as a home-area network gateway, then aggregate the results and finally pass them to the Cloud if needed.

Jiang et al. point out that some undesirable situations and the violation of an access control policy can appear because a user can generate a new private key for the access right [58]. To solve the problem, they propose a method to resolve this issue by formalizing security requirements and constructing an attribute-based encryption (ABE) scheme to satisfy the new security requirements.

To enable authentic and confidential communications among a group of Fog nodes, Alrawais et al. propose an efficient key exchange protocol based on ciphertext-policy attribute-based encryption (CP-ABE) to establish secure communications among participants [33]. To achieve confidentiality, authentication, verifiability, and access control, they combine CP-ABE and digital signature techniques together. The architecture with a key generator server is composed of the following entities: the Cloud, the key generator server, Fog nodes, and IoT devices. The key generator server is used to generate and distribute the keys among the involved entities. The Cloud defines an access structure to all Fog nodes and performs the encryption to get ciphertext [33].

In addition, Zhang et al. propose an access control scheme with outsourcing capability and attribute update for the Fog [57]. Yu et al. provide an access control method with leakage-resilient functional encryptions against side-channel attacks in the Fog [61]. Abdul et al. develop a biometric security method based on face images by using visual cryptography and zero-watermarking in the Fog [56].

3.3.3. Certificate authority and policy-based access control

Alrawais et al. focus on the security and privacy issues in IoT environments and propose a scheme that employs Fog nodes to improve the distribution of certificate revocation information among

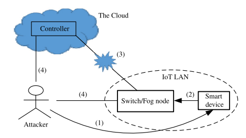


Fig. 6. A MitM model [65].

IoT devices for security enhancement [62]. It consists of a certificate authority (CA), a back-end Cloud, Fog nodes, and IoT devices. Dsouza et al. propose a policy-based resource access control in the Fog to support secure collaboration and interoperability among heterogeneous resources [63]. They adopt eXtensible Access Control Markup Language to define formalized and refined operational, security and network policy specifications.

3.4. Attacks

Because Fog nodes are usually deployed in some places with relatively weak protection, they may encounter various malicious attacks [24]. In addition, a malicious user can either tamper with its own smart meter, report false readings, or spoof IP addresses [45].

3.4.1. Malicious nodes and their attacks

Attacks from malicious Fog nodes: Lee et al. examine the components and several unique security threats of IoT Fog [42]. As one of the potential threats, a malicious Fog node problem is serious. In their research, the heavy workloads in the Fog are divided into several jobs and processed by Fog nodes. If some of these nodes are attacked by malicious users, it is hard to ensure the security of the data. However, they fail to give a detailed solution to solve such attack problem. In [21], the propagation of malicious nodes in the Fog is studied. The interaction between vulnerable nodes and malicious nodes in the Fog is first investigated as a non-cooperative differential game. The complex decision making process is then reviewed, analyzed and realized.

Attacks from malicious edge devices of users: Identifying malicious edge devices is crucial in data security in the Fog. However, it is difficult to prevent the attack because of certain privileges granted to them to use and process the data. Sohal et al. propose a framework by using a Markov model, intrusion detection system and virtual honeypot device to solve the problem [64].

3.4.2. MitM attack

Secure transmission channels protect all the traffic between edge devices and their Fog nodes. However, a user's released data can be eavesdropped or altered by an external attacker before the Fog node implements a global concealing process [18]. MitM is the typical kind of attack whose model is shown in Fig. 6.

In Fig. 6, after implementing four following steps, the outside attacker successfully intercepts the OpenFlow channel and controls the gateway as follows.

- For a device inside the IoT LAN, the outside attacker can take control of it by launching firmware updating attack because embedded smart devices are vulnerable to attacks.
- The smart device installs a client certificate at the Fog node, claiming that the Fog node needs to use this certificate to identify itself in the future communications.
- After the Fog node installs the client certificate, the outside attacker breaks the connection between the controller and it.
- The attacker performs MitM attack on the OpenFlow control channel.

In IoT Fog, the security issue of an OpenFlow channel between the controller and its switches is addressed in [65]. Because all the controller commands are sent through this channel, once attacked, the network is completely controlled by an attacker. It is a disaster for both the network service providers and their users. A countermeasure using the Bloom filter to detect MitM attack is thus proposed [65]. Stojmenovic et al. study MitM attack and investigate its stealthy features by examining its CPU and memory consumption of Fog devices [44]. One can also solve the problem by adopting encryption and decryption [24].

3.5. Privacy preservation

Privacy preservation is important because of users' many concerns about their sensitive data [47]. Different privacy preservation policies, schemes and methods are proposed, especially in healthcare systems [17–20]. Based on different technologies for access control, we summarize the current related work into Table 4 and discuss it next.

3.5.1. Location privacy preservation

A location privacy issue remains a challenge in the Fog [68]. Location-based services are popular and can achieve low-latency with the help of the Fog. A privacy preservation scheme for locations in the Fog is studied in [68]. Kang et al. address location privacy issues for the Fog supporting Internet of Vehicles (IoV), which aims to overcome big latency and high cost [23]. Hence, a privacy-preserved pseudonym is presented for effective pseudonym management. The work [48] realizes trajectory privacy preservation based on the Fog for Cloud location services. The paper [66] presents a redundant Fog loop-based scheme to preserve the source node-location privacy and achieve energy efficiency in the Fog. The study [67] proposes a position cryptography protocol for preserving location privacy. Fog nodes are specifically to meet the requirements for location-specific applications and location-aware data management, such as in vehicular ad hoc networks [23,48,66] .

3.5.2. Other data privacy preservation

Du et al. concern the privacy issue inherent in a Fog platform, and propose a differential privacy-based query model [69]. Wang et al. propose a privacy-preserving scheme by using differential privacy in the Fog, which can simultaneously ensure users' privacy and confidentiality [50]. In [70], most privacy-preserving data aggregation schemes support data aggregation for homogeneous IoT devices only and cannot aggregate hybrid IoT devices' data into one. Hence, a lightweight privacy-preserving data aggregation scheme for the Fog-enhanced IoT is presented. Al Hamid et al. formulates a secure privacy data and authenticated key agreement protocol based on the bilinear pairing cryptography [17]. Elmisery et al. study and reveal the disclosure boundary between privacy and publicity as well as the identity boundary between self and other [18]. Basudan et al. propose a privacy-preserving protocol for enhancing security in a vehicular crowdsensing-based road surface condition monitoring system by using the Fog [72]. Hu et al. propose a privacy-preserving scheme for identifying face by using the Fog [73]. Fog-based Vehicular ad hoc network is a new paradigm with the advantages of both vehicular cloud and the Fog, a secure and privacy-preservation navigation scheme is proposed for it [74].

Table 4
The summary for privacy preserving in the Fog.

Object	Technology	Ref.
Location privacy preservation	Position cryptography protocol	[66,67]
	Privacy preservation scheme	[23,68]
	Privacy-preserved pseudonym	[48]
Other Data privacy preservation	Differential privacy-based query	[50,69]
	Privacy-preserving data aggregation	[70]
	Cryptographyand decryption	[17,71]
	Disclosure boundary	[18]
	Privacy-preserving protocol and scheme	[72–74]

3.6. Secure communications

There are two kinds of secure communications [12]: (1) Communications between constrained-IoT devices and Fog nodes; and (2) Communications among Fog nodes. There may be bogus messages during communication when wrong information is sent by attackers in the network [46]. Mukherjee et al. indicate that security should be robust and customizable in a resource-constrained Fog environment when transferring data from the Edge to the Cloud [75]. They design an intermittent and flexible end-to-end security mechanism for Fog–Cloud communications. It can deal with unreliable network connections and achieve security configurations suited for different application needs. Wang et al. propose a scheme to protect the identities of edge devices by using pseudonyms and guarantee data secrecy via a homomorphic encryption technique when uploading data from edge devices to the Cloud [76]. During communications from edge devices to the Cloud, the confidentiality and integrity of data should be guaranteed by using light-weight encryption or masking algorithms [24]. Fang et al. design a lightweight secure routing protocol based on a source anonymity in the Fog to improve the transmission performance and enhance security [77].

3.7. Others

Service availability: It includes how to decrease denial of service (DOS). When there are millions of user requests for a same service, DOS occurs if hackers take this advantage for attacking [78]. New scheme for defending DOS attacks [79] should be designed; novel methods should be proposed to avoid the unnecessary resource consumption and offer sufficient caching capabilities so as to improve the service availability.

Secure applications: Khan et al. summarize the potential security issues found in the following Fog applications: virtualized radio access, web optimization, 5G mobile networks, smart meters, healthcare systems, surveillance video processing, vehicular networks and road safety, food traceability, speech data, augmented brain–computer interface, resource management, energy reduction, disaster response, and hostile environment [10]. Not all Fog nodes are resource enriched. Therefore, large scale applications requiring resource-constrained nodes are not quite easy compared to conventional data centrals. The hot applications mainly focus on healthcare [17,18] and vehicles [23,46,72,80,81]. Encrypting sensitive data can improve the security of the applications when invoking APIs. However, if too many APIs are invoked and deployed, they may consume too many resources, thereby affecting the normal access to them, and even leading to the application system paralysis.

Secure sharing technology: This occurs when the information is shared among many sites [82], such as orchestration between Fog nodes and services. Social network can be creatively used for service sharing in the Fog. A novel security service provision model is proposed to recommend security services accurately with a crowd sensing-enabling mechanism in the social Fog [83].

4. Open research issues

Offering high security and trust is important to the Fog customers. Several open research issues remain.

4.1. Trusted execution environment

Devices in the Fog are often deployed without strict monitoring and protection, thereby facing all kinds of security threats. How to increase the trust in the Fog is the primary challenge. Public key infrastructure (PKI) based technique could partially solve this problem. Trusted execution environment (TEE) technique may have its potential in the Fog [18,62,84,85]. The open network environment of the Fog makes a malicious procedure easily spread to intelligent devices and bring heavy threat to user data [21]. How to control and avoid such spread of a malicious procedure is very important in trusted execution environment.

4.2. Trust and security during Fog orchestration

Achieving inter-nodal secure collaboration and trusted resource provisioning is an important issue [63]. Fog nodes are distributed, virtualized and shared. Fog orchestration should be optimized for a multi-tenant model. Complete isolation should be maintained among tenants through a virtualization system to avoid cross-disclosure and privacy compromise of application-specific data [27]. Based on policy-driven security management for the Fog, the work [63] studies resource management expanding the current Fog platform to support secure collaboration and interoperability among different user-requested resources in the Fog. However, multi-level collaboration results in large security and trust issues, mainly including identity management, authentication, access control, information sharing and QoS, for example, when a Fog node experiences a failure, nearby Fog nodes on the same or adjacent layers should step in to carry the load [27], which deserve further studies.

4.3. Access control

It is hard but important to improve confidentiality in accessing largely distributed Fog. Confidentiality [33,50,52,55] in the Fog needs to be scalable and efficient to cope with resource-constrained Fog devices. Traditional methods lack efficiency for the Fog. Lightweight encryption algorithms should be helpful for Fog nodes and edge devices [62]. In the Fog, we can also raise questions like how to design new access control methods spanning user-Fog–Cloud, to overcome the limitations of edge devices at different levels [63]. The Fog is visualized as an ideal candidate to grant access tokens to authorized parties. A centralized architecture may be used to authorize access and relay data between authorized parties and edge devices (IoT devices) [31]. However, in a distributed Fog environment, a centralized architecture may fail and result in inconsistent information. Hence, new mechanisms providing consistent guarantee need to designed.

4.4. Collusion attack

Collusion attackers may be disguised. New technologies should be developed to protect the system privacy when facing a possible collusion attack [50]. In [50], there are two kinds of collusion attacks:

- Brokers collude with service providers: Malicious brokers and providers may spread the fake or duplicate events to the Fog network. Moreover, the latter can leak confidential information.
- Brokers collude with users: Malicious brokers and users may deny admitting any match accomplishing or utilizing data and services from legal providers.

However, the current work has not considered the attack from providers colluding with users. Malicious providers may leak confidential information to users for some profit from the latter. Colluding users may deny admitting obtaining any confidential information from the former. Hence, how to build high-security and low-cost collusion attack detection countermeasure is one of the key problems in the Fog. New methods, such as collaborative detection, may be used to observe and detect this kind of challenging attacks.

4.5. Data-dependent security and context-aware security

In the Fog, to improve data-dependent security, backup is needed for vital data. In addition, the data across multiple end-user devices faces larger attack possibility and more challenges [86]. Because an adversary can compromise data integrity by attempting to modify or destroy legitimate data, it is essential to define a security mechanism to provide data integrity verification of the transmitted data between the Fog nodes and the Cloud [33]. The context-aware applications present new and interesting security challenges for emerging applications [34]. The nature of supporting for mobility and proximity to end-users of the Fog calls for a new security approach for transparently and adaptively dealing with context changes, especially for authentication, access control and trust degree in different context-aware environments.

4.6. Service trust

In the Fog, a secure and trustworthy manner for users is needed [18]. However, the Service Level Agreement (SLA) is often affected by many factors such as, service cost, energy usage, application characteristics, data flow, and network status. Therefore, given a particular scenario, it is quite difficult to specify a service trust [84,85], which deserves in-depth study. Trust evaluation mechanisms in the Cloud can be modified to increase the security of Fog services [87]. Based on operational requirements and execution environments, one has to select suitable and trusted Fog nodes, their corresponding resource configuration and places of deployment in the Fog. Current methods [84,85], lack compatibility and scalability to track and verify their trust values and monitor them for the Fog. SLA [12] that has gained success in designing a trust model in Cloud computing may be modified to use in the Fog.

5. Future work

Based on the current research results, we propose the following issues to be addressed in the future:

5.1. Trust management models

A trusted third party is adopted in [18,48,88,89]. It can improve the security. However, the past work fails to consider the region problem. The method in [53] involves region issues for the Fog, as shown in Fig. 5. However, the Fog node elected as a delegate from a region Fog may be a malicious node, which makes the Fog face a larger trust risk. To solve the problem and improve trust reliability, an improved trust management model is needed. One solution may be proposed as follows:

- Using a trusted third party can decrease the trust management and computation overhead of the Fog through the detailed consideration of the Fog scope. Then using the partition concept similar to [53] can decrease the management and computation overhead in the Fog;
- Managing the inter-region trust values for Fog nodes via using communication and historical data with each other in the region Fog;
- Managing the region–region trust values for the region Fog by communicating with the Cloud and using historical data in the Cloud;
- Some strategies can be adopted to improve the trust management in a region Fog, for example, by using strong identity verification mechanisms for Fog nodes [90].

5.2. Identification of trusted nodes

Fog users with resource-constrained devices can outsource their tasks to their trusted Fog nodes [88]. How to identify such nodes is crucial. Several strategies may be adopted:

- If there are direct or indirect trust evaluations of the goal nodes, trusted nodes are chosen according to their trust values.
- Otherwise, by using a probability model based on historical behavior of the goal nodes, we can assess their trust values and make the choice accordingly.
- Collusion deceptions may be conducted by collaborated users or Fog nodes. It means that collusion behavior is initialized and carried out both by users or by malicious Fog nodes. The kind of collusion behavior need to be detected to identify trusted nodes in a region Fog.
- Because Fog nodes tend to be resource-constrained, it is necessary to design lightweight solutions [91]. Hence, the trust computation algorithm should be simple and occupy limited memory only. The Fog's dynamic nature brings new challenge for trust computing when facing constrained resources and low latency requirements. A light-weight strategy for trust computation based on a region Fog may be designed for meeting certain constrained resource requirements. If the computation task is large, it may be executed in the Cloud.

5.3. Secure orchestration

Optimized Fog orchestration is needed to avoid cross-disclosure and privacy compromise of application-specific data. QoS and service level agreements (SLA) are needed to ensure that all applications receive the minimum and enough levels of resources for orchestration [27]. Two kinds of orchestration deserve further study.

Application orchestration: Large-scale IoT services within the Fog infrastructures, such as smart cities, healthcare, and marine monitoring are composed of sensors, computing resources and devices. Orchestrating such applications can simplify maintenance and system reliability. However, how to efficiently monitor and process these applications' transient behavior and dynamic changes is a crucial challenge [35]. An agent-based trust computation model

may be used for orchestration. A multi-agent orchestration method may be adopted to monitor the applications and coordinate them securely in real time in the Fog.

Fog nodes' orchestration: The orchestration of Fog nodes can deploy virtual services to Fog infrastructure and facilitate resource collaboration [91]. When Fog nodes collaborate with others, one node may be attacked by malicious users or Fog nodes. If it is infected, the infected one may attack and infect other nodes. This induces security and trust risk spreading among Fog nodes even to a whole region Fog. Some strategies should be taken to decrease this kind of risk, by failing the infected node. If a Fog node is made in a failure status during an orchestration, the other Fog nodes potentially have to change their topology and rebuild their communication structure reliably. The data in the failed node should also be transferred to other reliable Fog nodes such that the orchestration can proceed. A policy-based service orchestration may be adopted.

6. Conclusions

The Fog is a highly virtualized platform but not a replacement of Cloud computing. It provides storage, computing and networking services among edge devices as well as traditional Cloud computing data centers [13]. It mainly solves the problems of low latency, mobility support and location awareness in many cyber–physical systems [92,93]. However, its distributed and open structure makes it vulnerable and weak to security threats.

This work analyzes the architectures of the Fog from a general to detailed ones. The related security and trust research results about the Fog are summarized, and the discussions about open security and trust issues are given, and future work is outlined. A distributed and remotely operated Fog can pose new security and trust challenges, which are not presented in the centralized Cloud. We make some concluding remarks for the Fog:

New methods are needed. Because the Fog is highly distributed, implementation of security mechanisms for data-centric integrity can affect its QoS to a great extent [28]. Hence, we need to find new methods to improve the security and trust of the Fog. Low-latency and a large number of resource-constrained devices in the Fog motivate researchers and engineers to propose new methods such that the Fog can enjoy high security and trust.

New interfaces are needed. Because Fog nodes need to interact with different hardware platforms provided by different vendors, new interfaces are required for Fog software to ensure trusted computing.

New protocols are required. Some protocols have already been designed, such as those in [77,94]. However, novel protocols are lacking to automatically detect security and trust compromises in the Fog.

Considering the limitation of the current researches and research trends, it is critical to develop and carefully design a suite of elaborate security and trust countermeasures. Many security and trust issues seem to remain open and have thus attracted much attention from researchers and engineers. As a final remark, it should be underlined that these issues should be addressed by design [95].

Acknowledgments

The work was supported by National Natural Science Foundation of China under Grants 61472005, 61201252; CERNET Innovation Project, China under Grant NGII20160207, FDCT (Fundo para oDesenvolvimento das Ciencias e da Tecnologia) under Grant 119/2014/A3, and the European Union through the INTER-IoT, Research and Innovation action - Horizon 2020 European Project under Grant Agreement #687283.

References

[1] M.H. Ghahramani, M.C. Zhou, C.T. Hon, Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services, IEEE/CAA J. Autom. Sin. 4 (1) (2017) 5–17.
[2] Y. Xia, M. Zhou, X. Luo, Q. Zhu, Stochastic modeling and quality evaluation of Infrastructure-as-a-Service clouds, IEEE Trans. Autom. Sci. Eng. 12 (1) (2015) 160–172.
[3] H. Yuan, J. Bi, W. Tan, M.C. Zhou, B.H. Li, J. Li, TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds, IEEE Trans. Cybernet. 47 (11) (2017) 3658–3668.
[4] P.Y. Zhang, M.C. Zhou, Dynamic cloud task scheduling based on a two-stage strategy, IEEE Trans. Autom. Sci. Eng. (2017). http://dx.doi.org/10.1109/TASE.2017.2693688. (on-line).
[5] W.B. Zheng, M.C. Zhou, Y.N. Xia, L. Wu, Xin. Luo, S.C. Pang, Q.S. Zhu, Percentile performance estimation of unreliable IaaS clouds and their cost-optimal capacity decision, IEEE Access 5 (2017) 2808–2818.
[6] G. Luo, Y. Pan, ZTE communications special issue on cloud computing, fog computing, and dew computing, ZTE Commun. 15 (1) (2017) 2.
[7] T.H. Luan, L. Gao, Z. Li, L. Sun, Fog computing: Focusing on mobile users at the edge, 2015. arXiv preprint arXiv:1502.01815.
[8] F. Bonomi, inConnected vehicles, the Internet of Things, and fog computing, in: Proc. VANET, Las Vegas, CA, USA, Sep. 23, 2011, pp. 13–15.
[9] S. Ivan, W. Sheng, The fog computing paradigm scenarios and security issues, in: Proc. of the 2014 Federated Conference on Computer Science and Information Systems, 2014, pp. 1–8.
[10] S. Khan, S. Parkinson, Y. Qin, Fog computing security: a review of current applications and security solutions, J. Cloud Comput. Adv. Syst. Appl. 6 (19) (2017). http://dx.doi.org/10.1186/s13677-017-0090-3.
[11] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the Internet of Things, in: Proc. of MCC'12, 2012, pp. 13–15.
[12] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M.A. Ferrag, N. Choudhury, V. Kumar, Security and privacy in fog computing: Challenges, IEEE Access (2017). http://dx.doi.org/10.1109/ACCESS.2017.2749422.
[13] F. Bonomi, R. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for Internet of Things and analytics, in: N. Bessis, C. Dobre (Eds.), Big Data and Internet of Things: A Roadmap for Smart Environments, in: Studies in Computational Intelligence, vol. 546, Springer, Cham, New York, 2014, pp. 169–186.
[14] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, Q. Yang, Incorporating intelligence in fog computing for big data analysis in smart cities, IEEE Trans. Ind. Inform. 13 (5) (2017) 2140–2150.
[15] B. Molina, C.E. Palau, G. Fortino, A. Guerrieri, C. Savaglio, Empowering smart cities through interoperable sensor network enablers, in: Proc. of 2014 IEEE International Conference on Systems, Man and Cybernetics, SMC, IEEE, 2014, pp. 7–12.
[16] F. Cicirelli, A. Guerrieri, G. Spezzano, A. Vinci, An edge-based platform for dynamic smart city applications, Future Gener. Comput. Syst. (ISSN: 0167-739X) 76 (2017) 106–118. http://dx.doi.org/10.1016/j.future.2017.05.034.
[17] H.A. Al Hamid, S.M.M. Rahman, M.S. Hossain, A. Almogren, A. Alamri, A security model for preserving the privacy of medical big data in a healthcare cloud using a fog computing facility with pairing-based cryptography, IEEE Access (2017). http://dx.doi.org/10.1109/ACCESS.2017.2757844.
[18] A.M. Elmisery, S. Rho, D. Botvich, A fog based middleware for automated compliance with OECD privacy principles in internet of healthcare things, IEEE Access 4 (2016) 8418–8841.
[19] S.R. Moosavia, T.N. Gia, E. Nigussie, A.M. Rahmania, S. Virtanen, H. Tenhunena, J. Isoaho, End-to-end security scheme for mobility enabled healthcare Internet of Things, Future Gener. Comput. Syst. 64 (2016) 108–124.
[20] X. Liu, R.H. Deng, Y. Yang, H.N. Tran, S. Zhong, Hybrid privacy-preserving clinical decision support system in fog-cloud computing, Future Gener. Comput. Syst. 78 (2018) (2018) 825–837.
[21] Z. Li, X. Zhou, Y. Liu, H. Xu, L. Miao, A non-cooperative differential game-based security model in fog computing, China Commun. 14 (1) (2017) 180–189.
[22] V. Sharma, J.D. Lim, J.N. Kim, I. You, SACA: Self-aware communication architecture for IoT using mobile fog servers, Mobile Inf. Syst. (2017). http://dx.doi.org/10.1155/2017/3273917.
[23] J. Kang, R. Yu, X. Huang, Y. Zhang, Privacy-preserved pseudonym scheme for fog computing supported internet of vehicles, IEEE Trans. Intell. Transp. Syst. (2017). http://dx.doi.org/10.1109/TITS.2017.2764095. (in press).
[24] P. Hua, S. Dhelima, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, J. Netw. Comput. Appl. 98 (2017) 27–42.
[25] C. Dsouza, G.J. Ahn, M. Taguinod, Policy-driven security management for fog computing: Preliminary framework and a case study, in: IEEE IRI, 2014, pp. 16–23.
[26] G. Fortino, A. Guerrieri, W. Russo, C. Savaglio, Integration of agent-based and cloud computing for the smart objects-oriented IoT, in: Proc. of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design, IEEE, 2014.