Steganographic File Systems

Conventional Protection Mechanisms in File Systems

- User Access Control
  - The operating system is fully trusted to enforce the security policy.
- Is it good enough?
- Operating System cannot be fully trusted. Attacker can ***circumvent*** Access Control, and look into storage directly
  - Vulnerabilities of the system – attacks from hackers
  - Inadequate physical protection – house breaking
- In some distributed storage systems, data is usually unsafe, e.g., Data-Grid, Cloud
  - You are using others' storage
  - Centralized access control is hard to establish

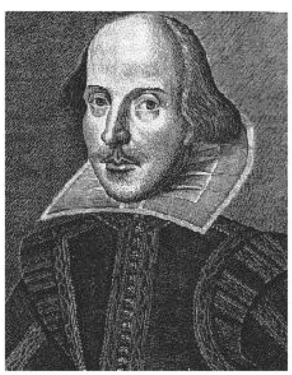
Conventional Protection Mechanisms in File Systems

- Encryption
  - Files are encrypted so that they can only be accessed when users supply the correct encryption key
- Is it good enough?
- What if the adversary *knows that the file exists* and … coerce/compel the owner to reveal the encryption key?

- Police or government officer can ***order*** the owner to give out his encryption key.
- Can you say NO?

How about applying **steganography** to file system?

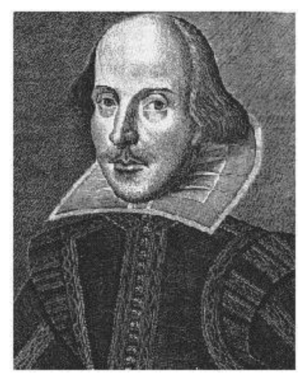
- Steganography is the art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message.
  - Greek Words: STEGANOS – *"Covered"* GRAPHIE – *"Writing"*
- Hide information so that the adversary does not know its existence.
- A higher level of security than cryptography – **plausible deniability**



+

Steganography is the art and science of communicating in a way which hides the existence of the communication.

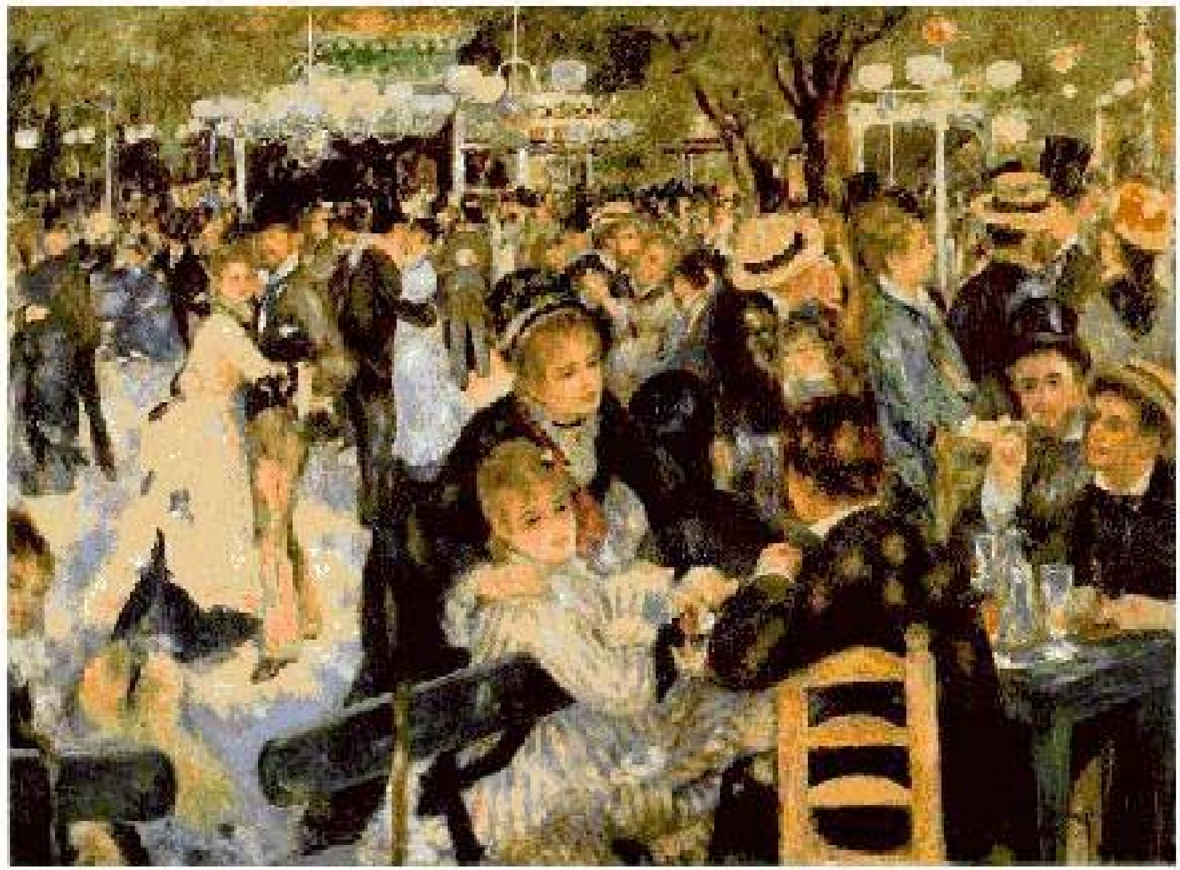
=



Example of Steganography

THE MOST COMMON WORK ANIMAL IS THE HORSE. THEY CAN BE USED
TO FERRY EQUIPMENT TO AND FROM WORKERS OR TO PULL A PLOW.
BE CAREFUL, THOUGH, BECAUSE SOME HAVE SANK UP TO THEIR
KNEES IN MUD OR SAND, SUCH AS AN INCIDENT AT THE BURLINGTON
FACTORY LAST YEAR. BUT HORSES REMAIN A SIGNIFICANT FIND. ON
A FARM, AN ALTERNATE WORK ANIMAL MIGHT BE A BURRO BUT THEY
ARE NOT AS COMFORTABLE AS A TRANSPORT ANIMAL.

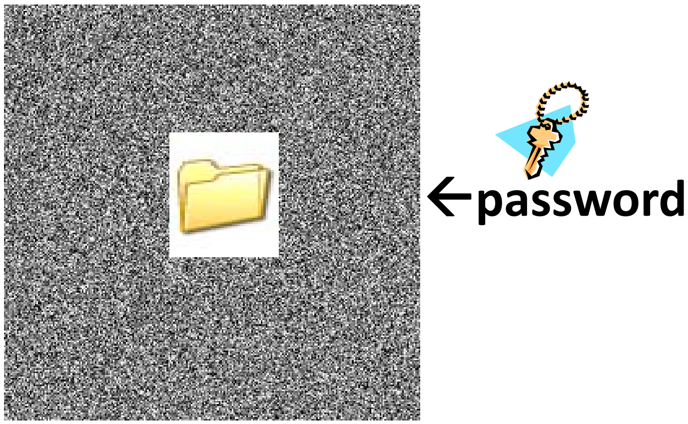
Example of Steganography



Long-Range Aviation Airfield

Steganographic File System

- **How about this: A file is hidden in the storage in such a way that, without the corresponding access key, an attacker cannot prove its very *existence*.**
  - Without access key, attacker can get no information of the file.



- **Plausible Deniability**
  - Even if the attacker or the government compels the owner to disclose his file, the owner can deny the existence of the file. The owner's denial is plausible because it cannot be proved to be wrong. This lovely property is called ***Plausible deniability***.

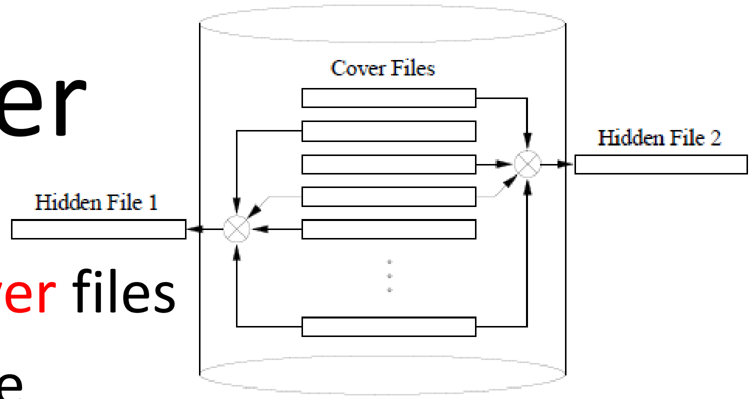
- **We call such a system Steganographic File System.**

Steganography vs Steganographic File Systems

- **Traditional Steganography**
  - Hide small piece of message inside cover-message (Multi-media)
- **Steganographic File System**
  - Hide files inside the secondary storage filled with random data.

- **Steganalysis – Attacks to steganography**
  - Statistical test to detect the hidden message
- **Attacks to steganographic file systems**
  - Statistical analysis on the secondary storage
  - Statistical analysis on the accesses on the secondary storage

Early Systems: StegCover



- System is divided into **n** equal-sized cover files
- Every cover is initially a random data file

C1,…Ci,…Cn

- When we want to insert a file F, we replace it with a cover Ci (after XORing F with k cover files)
- How to select Ci for file F?
  - Suppose we have 7 cover files C1-C7, and the password is:

    1 0 1 0 0 0 1

    P1 P3 P7
  - Select C1, C3, C7 to XOR with F

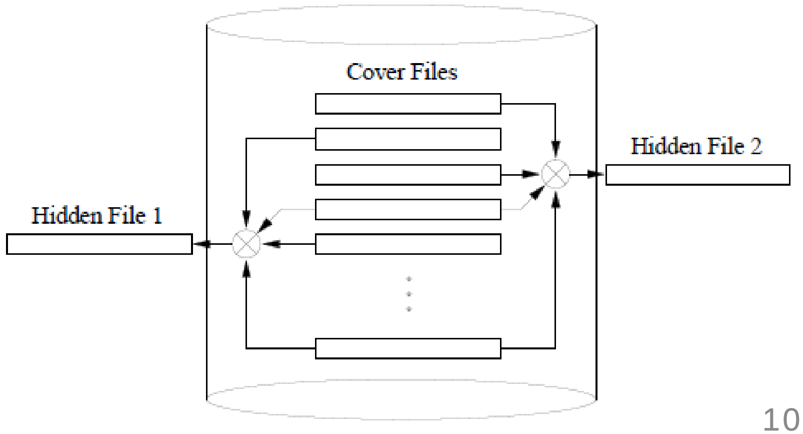
    $F' = C1 \oplus C3 \oplus C7 \oplus F$
  - Replace one of C1, C3,C7 with F' and XOR itself.

    $C3' = F' \oplus C3$
  - **Resultant content:** C1,C2,C3',C4,C5,C6,C7

Early Systems: StegCover

- When we want to get F, we extract it from the k covers with our password.
- How to recover F?
  - Using same password, select C1, C3',C7

$$C1 \oplus C3' \oplus C7 = C1 \oplus (F' \oplus C3) \oplus C7$$
$$= C1 \oplus (C1 \oplus C3 \oplus C7 \oplus F \oplus C3) \oplus C7$$
$$= C1 \oplus (C1 \oplus C7 \oplus F) \oplus C7$$
$$= \mathbf{F}$$

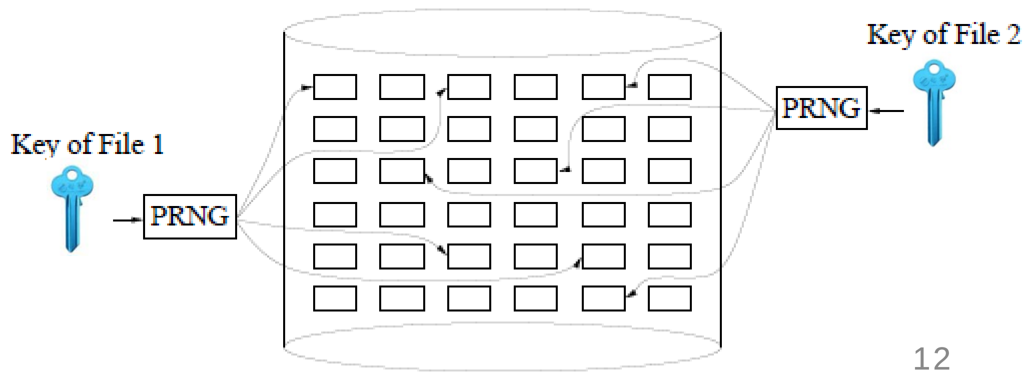


StegCover

- Given n cover files, can securely hide n/2 files
- Impractical: Computationally expensive
  - Need to retrieve all cover files
- If there are more than one file in the system, after inserting a new file, the old file's context is changed
  - e.g., inserting another file that also chooses C3 as one the k cover files!
  - So we must modify the context to make sure we can extract the old file properly.
- Low space utilization
- Vulnerable to traffic analysis to reveal hidden files

Early Systems: StegRand

- Fill the whole hard disk with random bits
- Write each (encrypted) file block at an absolute disk address given by some pseudorandom process (PRNG)
- Assumption
  - we have a block cipher which the opponent cannot distinguish from a random permutation
  - the presence or absence of a block at any location should not be distinguishable.
- To reconstruct hidden file, user provides password as the seed to the PRNG, which generates a sequence of addresses pointing to the data blocks that compose the file



StegRand

- If we have N blocks, we will start to get collisions once we had written a little more than $\sqrt{N}$ blocks (birthday problem)
  - Different file blocks can map to the same disk addresses, thus causing one to overwrite the other (data corruption)
  - Replicate hidden files/blocks by limiting the number of hidden files
    - Cannot eliminate problem completely – no guarantee on data integrity
    - Low storage utilization
- Vulnerable to traffic analysis

Summary

- Existing steganographic file systems have the following problems:
  - Low storage efficiency
  - Long processing time
  - Lack of guarantee on data integrity

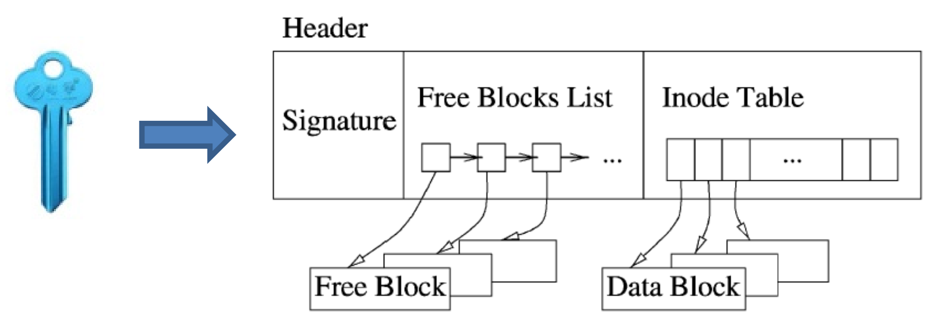
StegFS – A practical steganographic file system for local machine

- Each hidden object in the file system has a ***name*** and an ***access key***.
- A hidden object can be a ***file*** or a ***directory*** that contains many files.
- If a user provides a correct file name and the corresponding access key, the system can use them to locate the file. After that, the user can operate on the file regularly.
- Without the file name or its access key, an attacker could get no information about whether it ever exists, even if the attacker knows the hardware or software of the file system completely.
- Design principles
    - Offer the steganographic property – plausible deniability
    - With data integrity
    - Minimize space and processing overheads

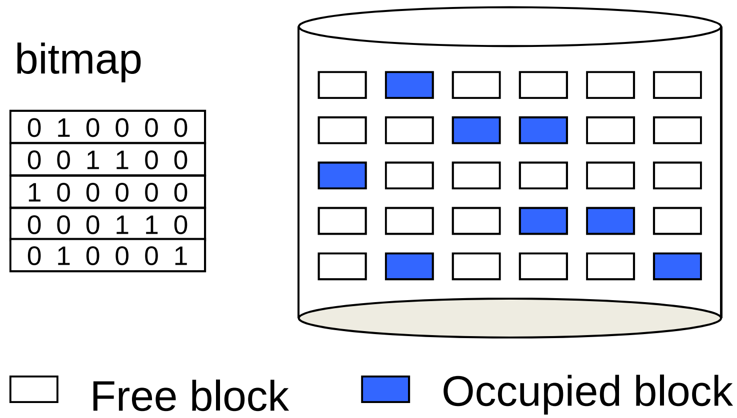
H. Pang, K.L. Tan, X. Zhou: Steganographic Schemes for File System and B+-trees. IEEE Trans. Knowl. Data Eng. 16(6): 701-713 (2004)

StegFS

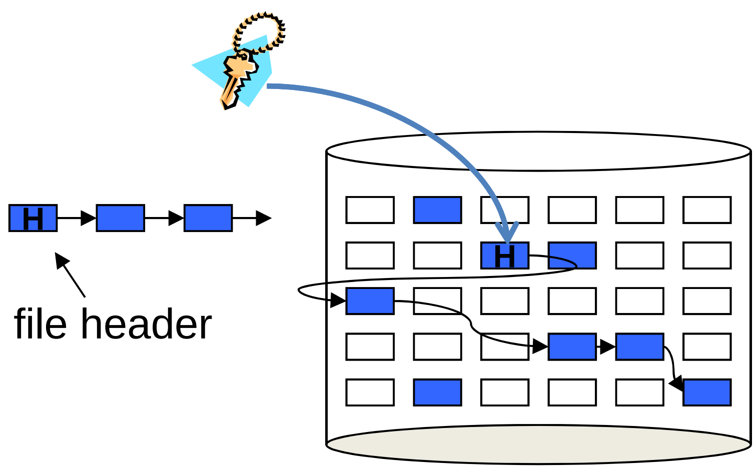
- To hide a file, all information related to its existence should be excluded from the file system
  - Object's structure (inode table) should not be in the central directory
  - Usage statistics not stored in metadata
- Instead, all these are isolated within the object itself
  - Header node
- User accesses header node (and data) with the accesss key



StegFS Construction

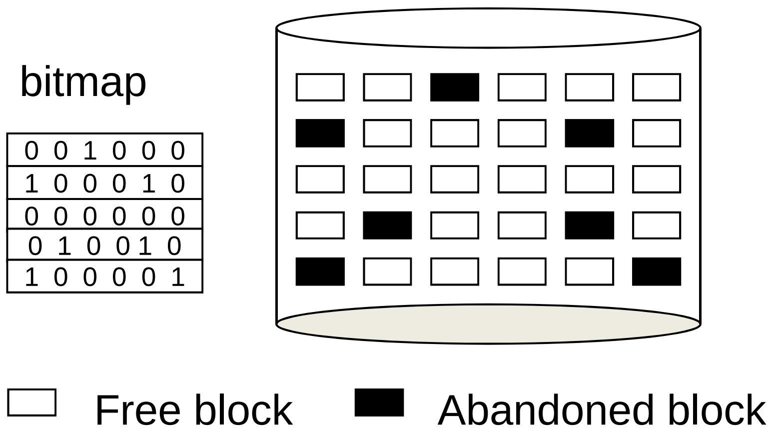


- The storage space is partitioned into ***standard-size blocks***, and a ***bitmap*** tracks whether a block is free or has been allocated – a ***0*** bit indicates a free block and a ***1*** bit signifies an allocated block.

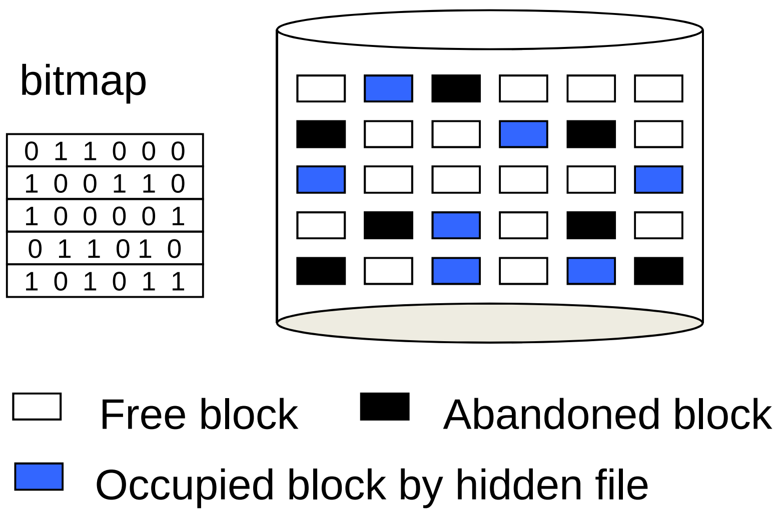


- A ***file*** is a ***link-list*** of data blocks. To locate a file in the storage space, we only need to locate the file ***header***.

StegFS Construction



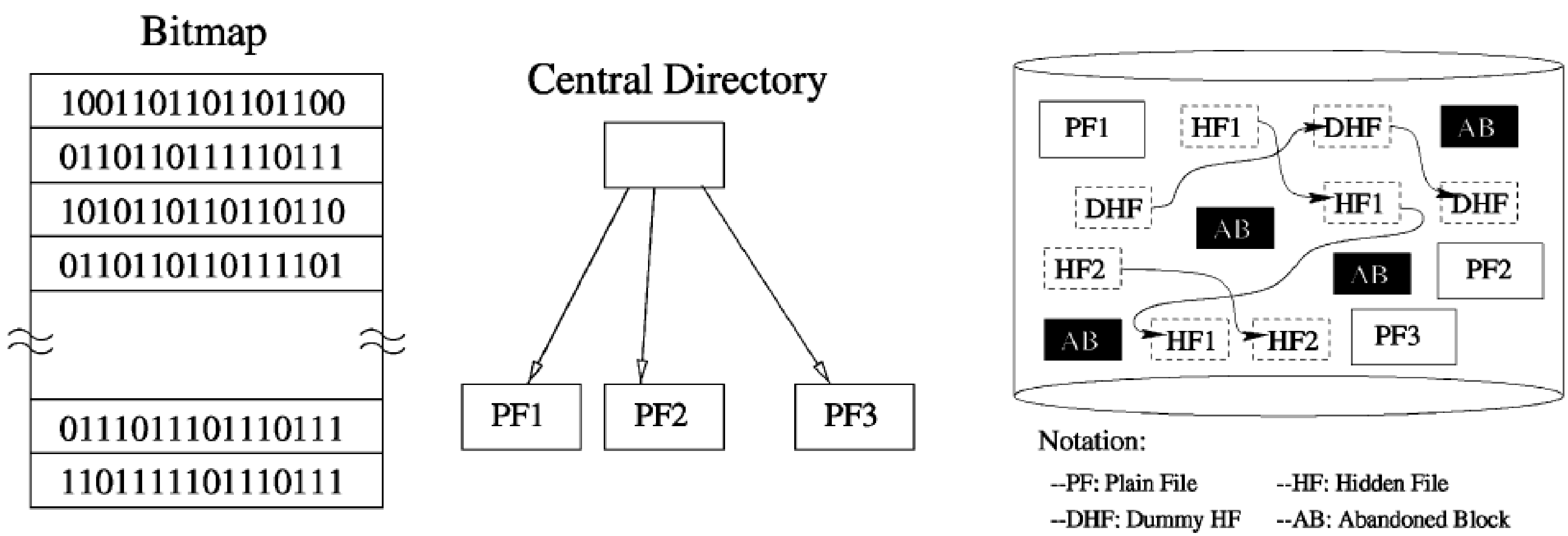
- When system is created, randomly generated numbers are written into all the blocks.
- Some randomly selected blocks are ***abandoned*** by turning on the corresponding bits in bitmap.



- The data blocks of a hidden file are ***randomly*** selected from the storage space (bitmap has to be updated)
- All the blocks, including the file header, are ***encrypted*** under a secret key, so that they are ***indistinguishable*** from the abandoned blocks.

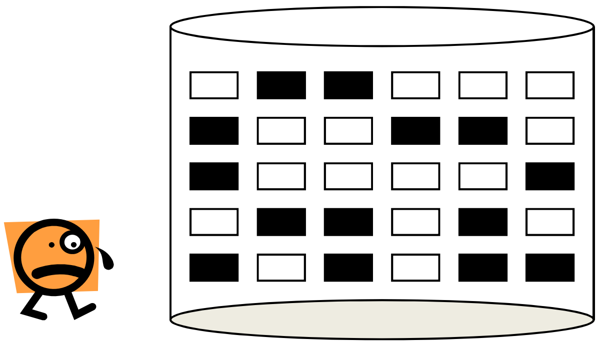
StegFS Construction

- StegFS additionally maintains one or more dummy hidden files that it *updates* periodically.
- Finally, plain (non-hidden) files are stored in the usual way (in the open)



How StegFS Facilitates Security?

- Why abandoned blocks?
    - For attacker, all the occupied data blocks in the file system look like abandoned blocks. It's difficult for him to figure out whether any files are hidden, and even if he knows, it is not clear how many files are hidden

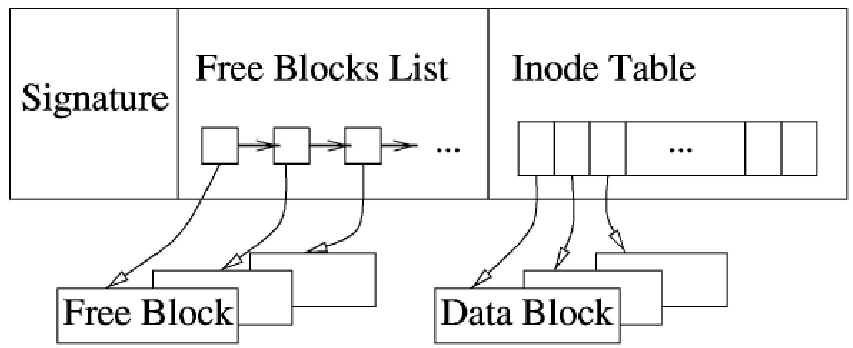


- Why dummy hidden files and why update them?
    - To prevent observer from deducing that blocks allocated between successive snapshots of the bitmap that do not belong to any plain files must hold hidden data
- Abandoned blocks vs dummy files
    - The former cannot be traced, but the latter (maintained by StegFS) are vulnerable to attackers

How StegFS Facilitates Security?

- Hidden files have free blocks
  - To deter any intruders who starts to monitor the file system right after it is created
    - Abandoned blocks are not useful here – they would have been eliminated from consideration
    - If intruder continues to take snapshots frequently enough to track block allocations in between updates to the dummy hidden files, then he would probably be able to isolate some of the blocks that are assigned to hidden files.
  - With an internal pool of free blocks, it is more challenging for intruder to distinguish blocks that contain useful data from the free blocks.
    - NOTE: Free blocks are randomly allocated to store data so as to increase the difficulty in identifying the blocks belonging to the file and the order between them

StegFS: Header Node



How to locate file header?

- At creation
  - Compute h = hash(filename, access key)
  - Use h as seed to a pseudorandom block number generator
  - Check each successive generated block number against the bitmap until the file system finds a free block to store the header
  - Subsequent blocks can be assigned randomly from any free space by consulting the bitmap, and linked to the file's inode table
  - Store signature (one-way hash function computed from filename, access key) in header block
- What if multiple users issue same filename and access key?
- To retrieve hidden file
  - Compute hash value h, and look for *first* block number that is marked as assigned in the bitmap *and* contains a matching file signature
    - Initial block numbers given by the generator may not hold the correct file header because they were unavailable when the file was created.

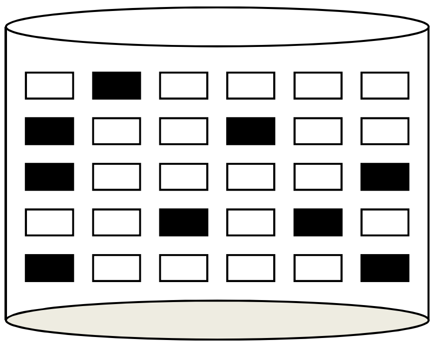
Other Issues

- StegFS is most effective for multi-user environment! Why?
- File Sharing
  - Need to distinguish between user access key UAK , and file access key FAK (to be shared)
- File system backup and recovery
  - To minimize overhead, saves image of blocks marked in bitmap but do not belong to plain files
    - Overhead for abandoned blocks, dummy hidden files, free blocks within hidden files
  - To recover
    - Restore image of abandoned and hidden blocks to their original addresses
      - Hidden files contain their own inode tables, so cannot be adjusted by the recovery process to reflect new block assignments
    - Plain files reconstructed last – possibly at new block addresses
  - How to handle accidental errors that result in corruption of data?
    - The header of a hidden file can be replicated and placed in pseudorandom locations derived from its FAK. Thus, if the file header is corrupted, the replica can be retrieved to recover the hidden file.
    - Additionally, a signature can be inserted in each data block, so that, if necessary, a hidden file can be recovered by scanning the disk volume for blocks with matching signatures.

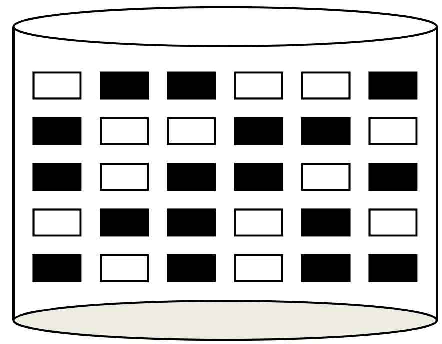
Security Measures

- Perfect Security for Cryptography
  - Pr(Data A|Cipher-text) = Pr(Data B|Cipher-text)
- Perfect Security for Steganography
  - Pr(Exist|Appearance) = Pr(Not exist|Appearance)

How Secure is StegFS ?



Pr(exist) = 0.2



Pr(exist) = 0.5

- StegFS is not perfectly secure, as the bitmap reveals probability information.
- However, StegFS is good enough to preserve **plausible deniability**.

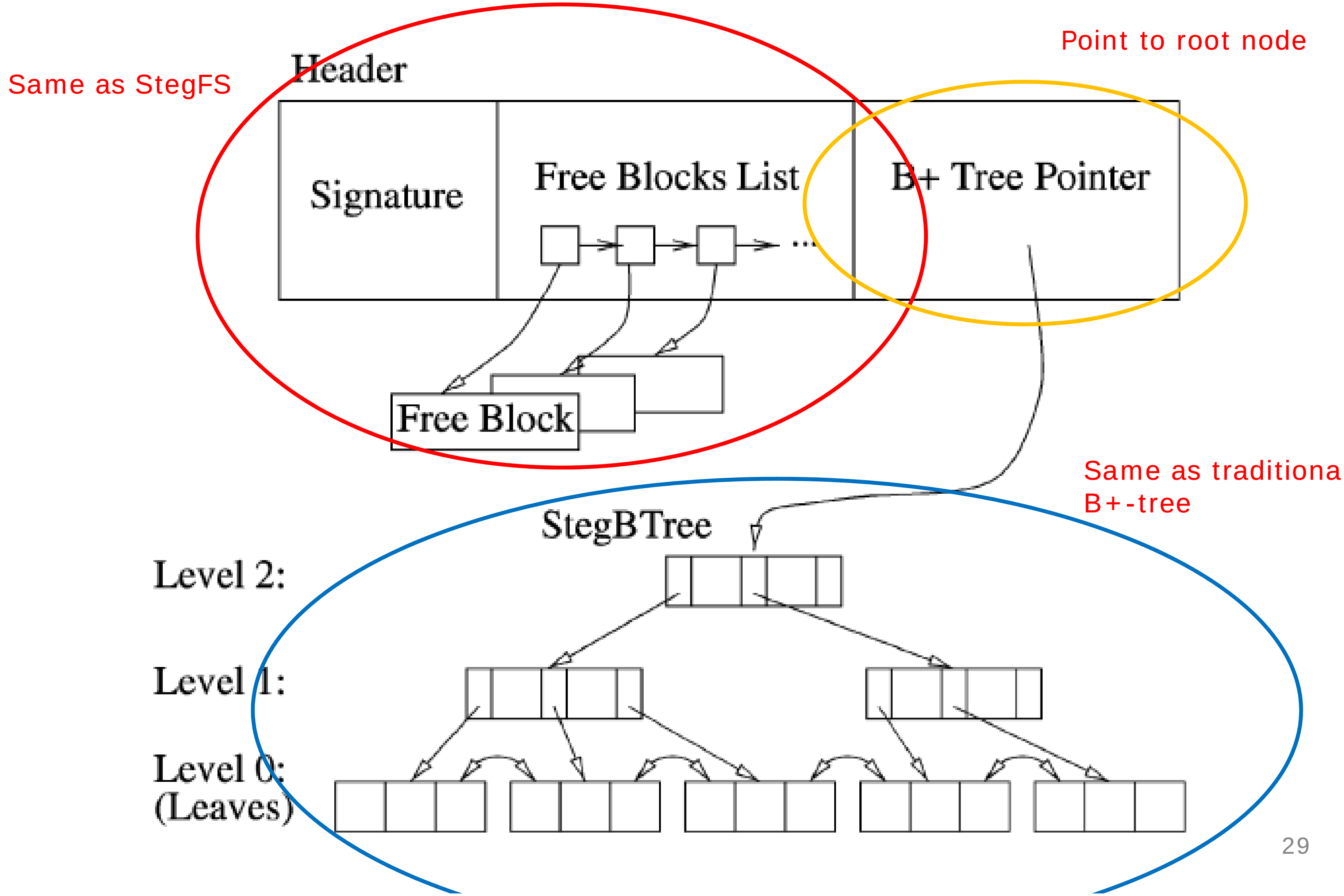
Space Utilization of StegFS

- **Abandon blocks and dummy files are crucial in StegFS.**
- **Trivially, more abandon blocks, more secure the StegFS.**
- **Space Utilization = 1 -** $\dfrac{\textbf{abandon blocks + dummy blocks}}{\textbf{total number of blocks}}$
- **Around 40% ~ 90% >> 10% (Steg-Random)**

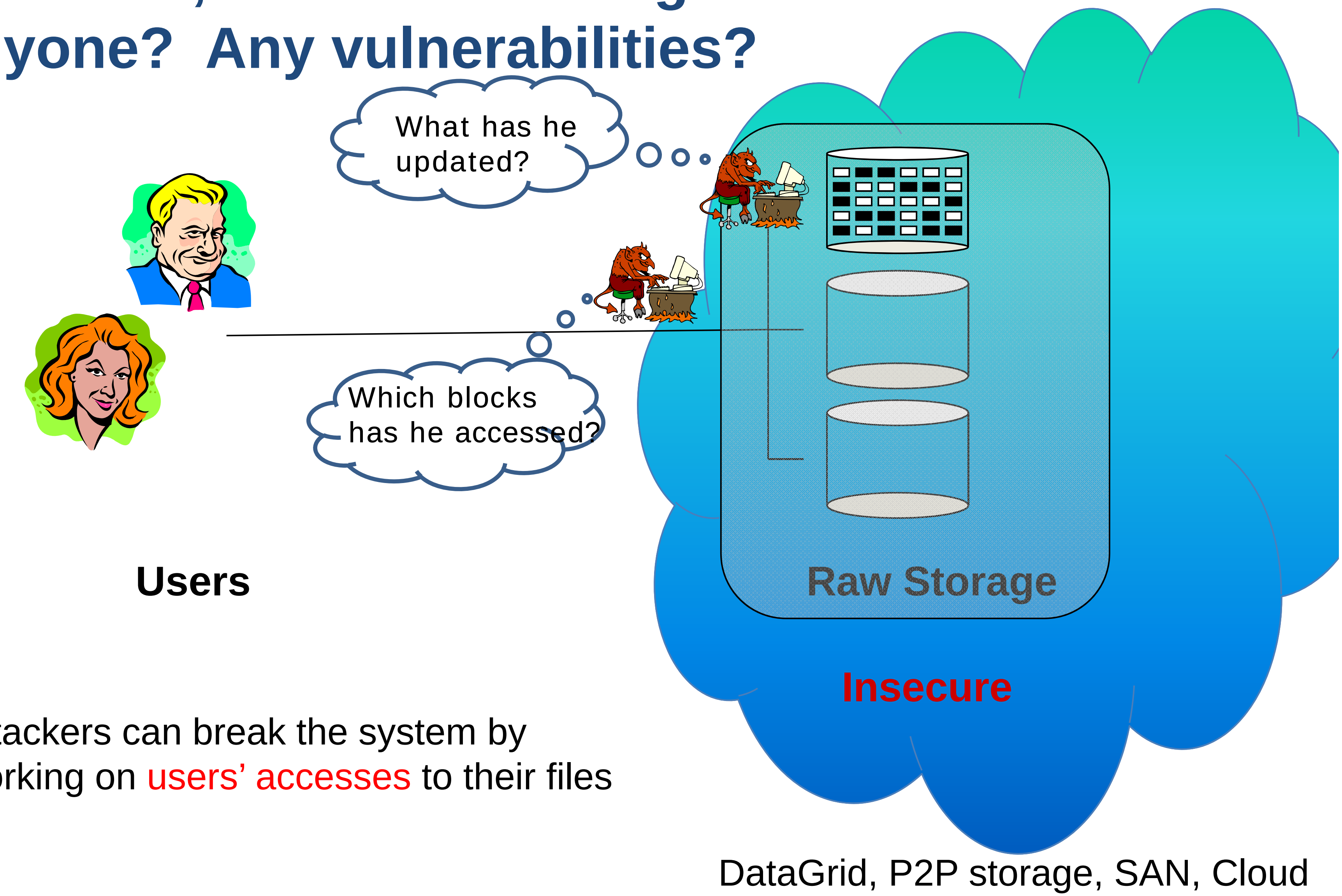
How about indexes?

- It is not always necessary to access an entire file
- Index support would be useful
- Two approaches
  - Can “install” a DBMS or an index structure on top of a steganographic file system
    - May suffer performance penalty if the block boundaries are not well aligned
  - Implement such a structure directly in a steganographic disk volumne

Steganographic B-trees

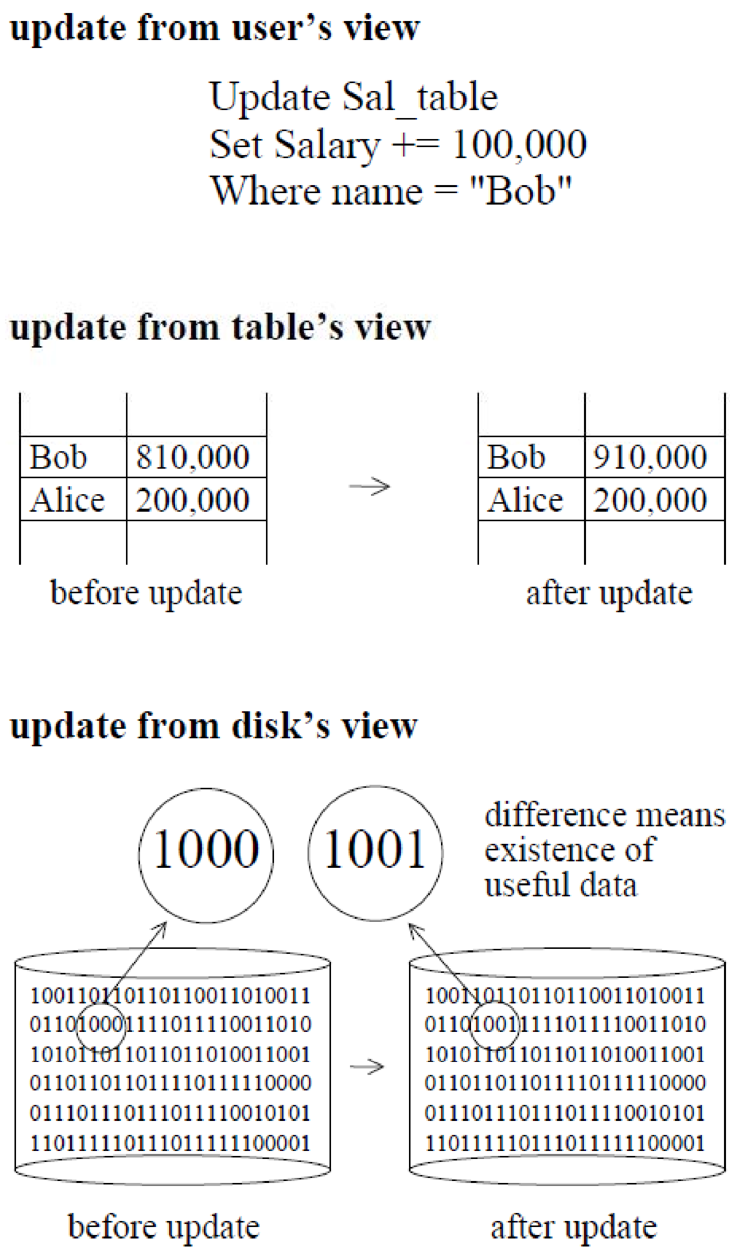


What will happen if we migrate StegFS to open networks, where the storage is accessible to anyone? Any vulnerabilities?



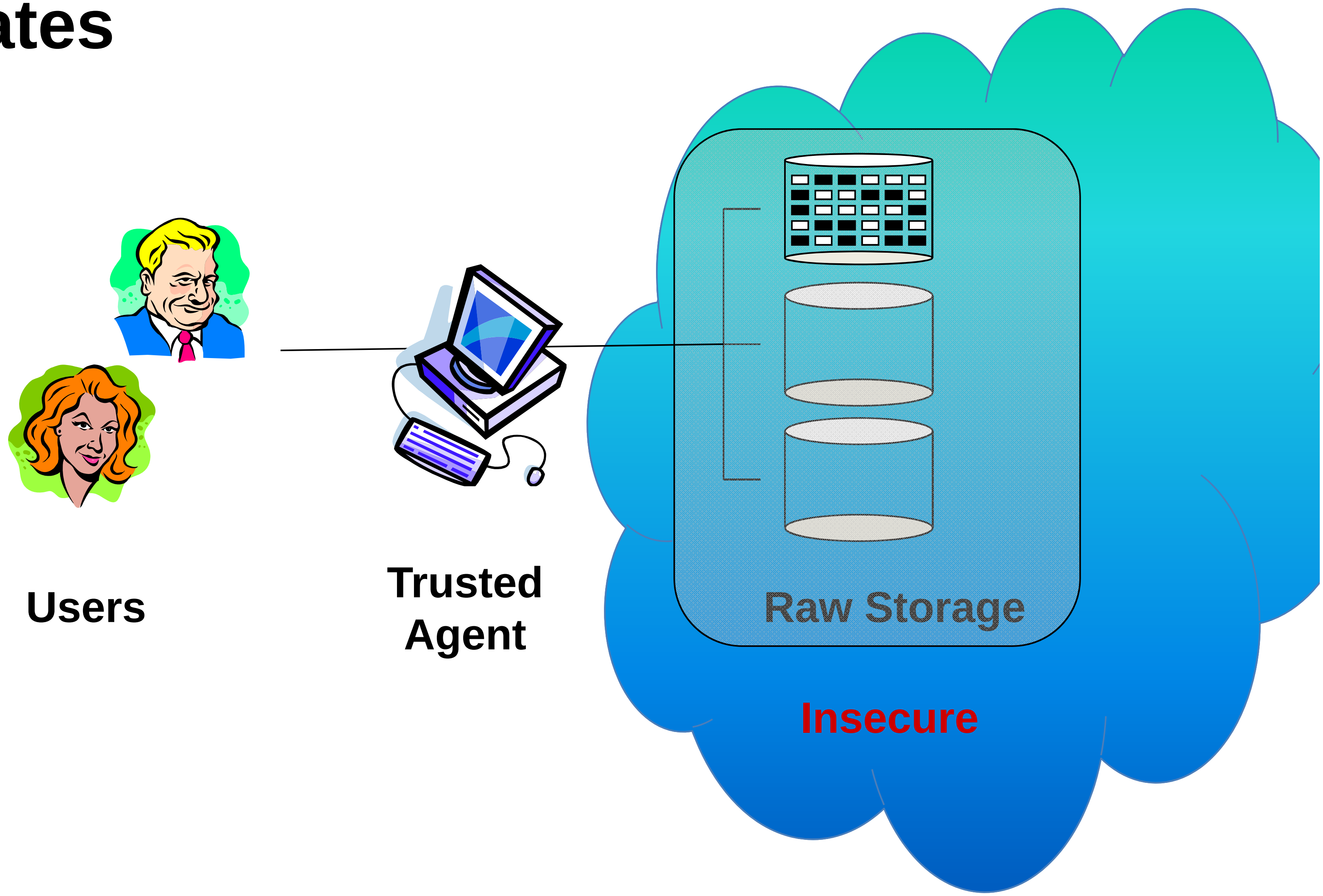
- Attackers can break the system by working on users' accesses to their files

Problem incurred by Updates



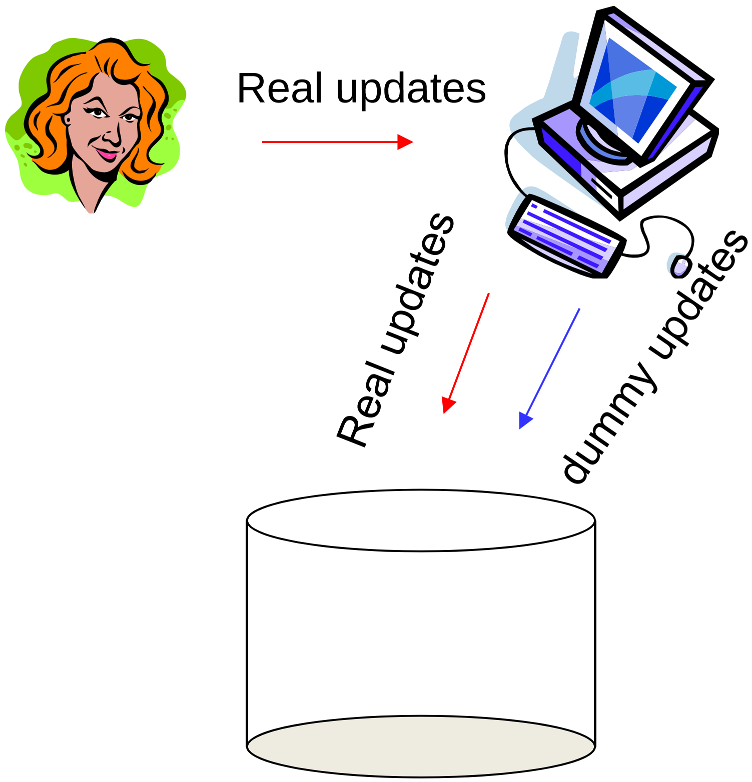
- **Update analysis**: If an attacker can compare two snapshots of the raw storage, he might discover the updates. Through the observed updates, he can deduce the existence of hidden data.

Countering Update Analysis: Dummy Updates

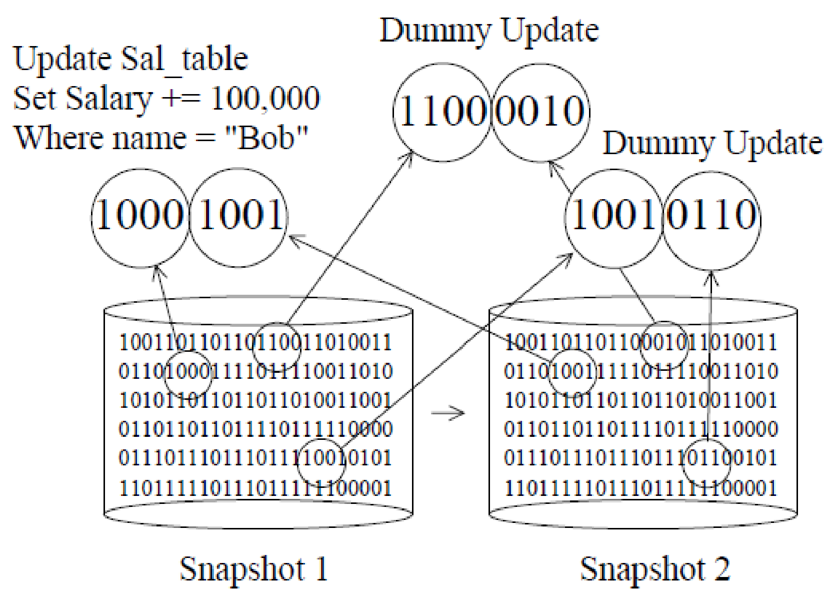


X. Zhou, H. Pang, K.L. Tan: Hiding Data Accesses in Steganographic File Systems. ICDE 2004: 572-583

Dummy Updates

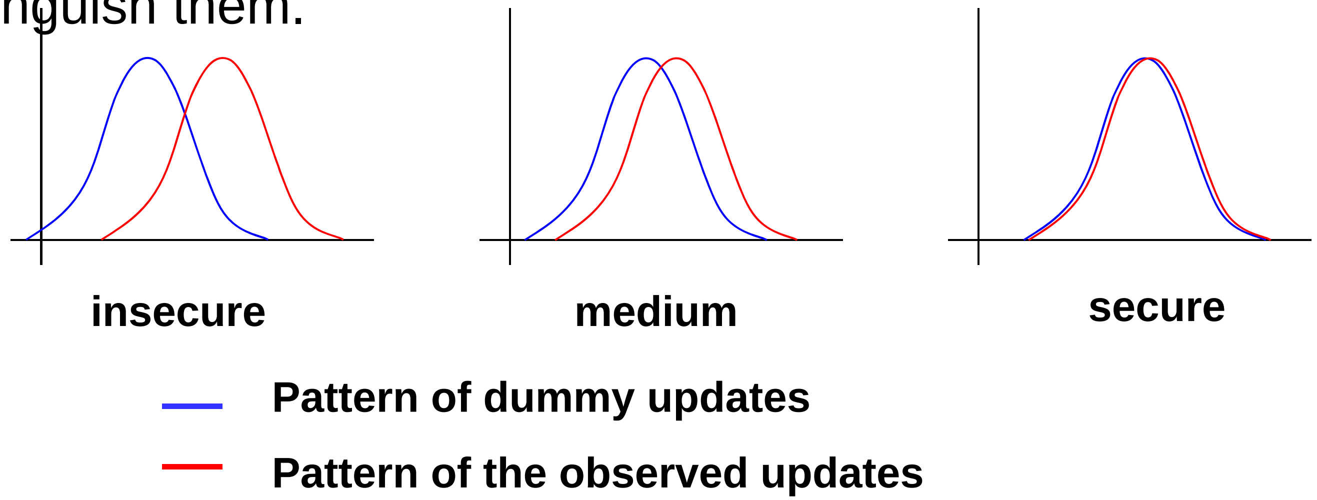


- Because of dummy updates, an attacker can no longer simply deduce the existence of hidden data from the observed updates.



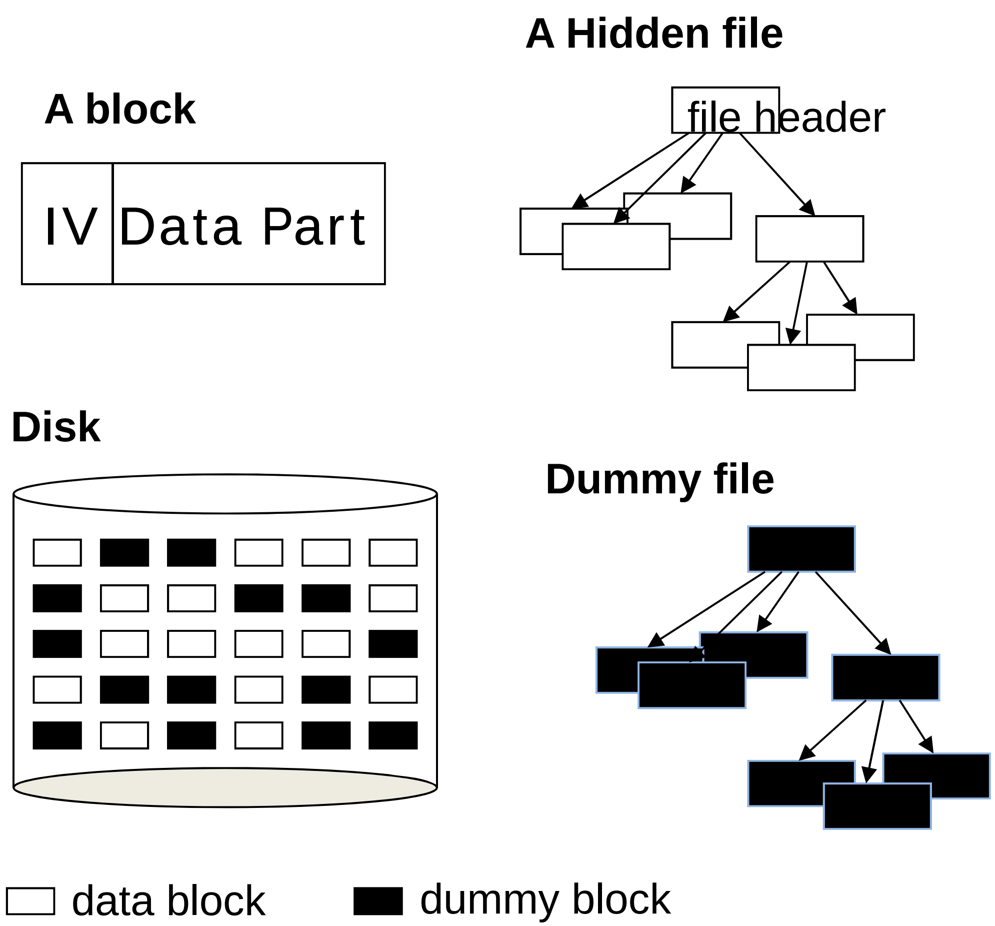
Principles of Design

- Perfect Security for Steganography
  - Pr(Exist|Appearance) = Pr(Not exist|Appearance)
- **Security –** the pattern of dummy updates and the pattern of real updates should be sufficiently similar, so that attackers cannot distinguish them.



- **Data Integrity –** the dummy updates should not affect the integrity of the existing data.
- **Performance –** the processing overhead should be minimized.

System Construction

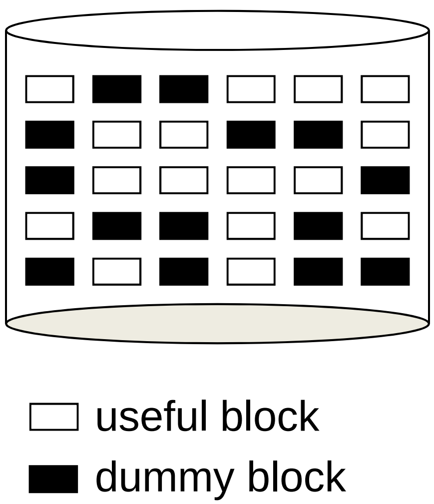


- Storage is partitioned into standard-size blocks. Each block can be either a data block or a dummy block.
- Each block is composed of an *initial vector* (IV) and a data part, and is encrypted using Cipher Block Chaining with IV as seed.
- Data blocks contain useful information, and are organized into hidden files.
- All dummy blocks contain random data, and are organized into a single dummy file.
- Agent holds two keys: FAK of dummy file, and secret key for encrypting data. To access data, owner must also pass the FAK of file to agent.

Dummy Updates

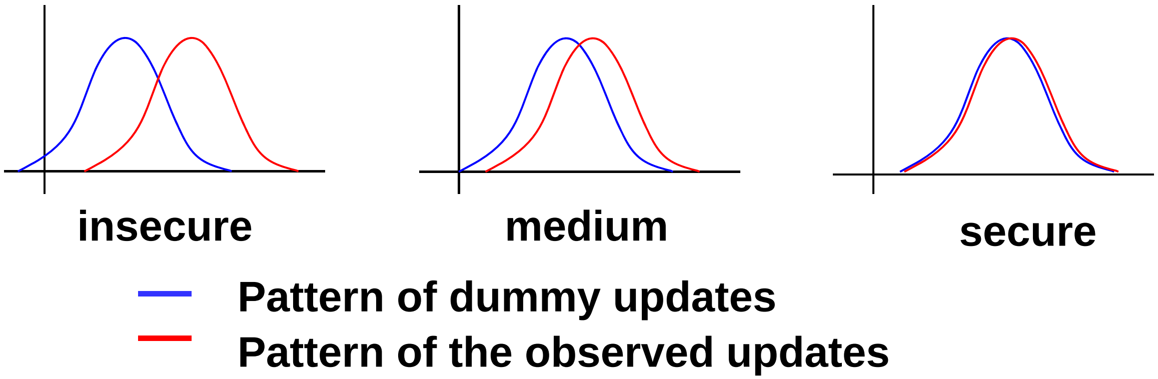
A Data block

Disk



- **Dummy updates** – agent randomly selects a data block, decrypts it, updates its IV, re-encrypts it, and then writes it back
- As the data block is encrypted, attacker cannot distinguish whether the IV or the data part is modified
- As dummy updates only change IVs, they do not affect the integrity of existing data

Dummy Updates *vs*. Real Updates



58921235168497130984274618	**Normal (absolutely random)**
8892**8**2**8**516**8**497**8**309**88**27**8**61**8**	**Abnormal - frequency**
1234509876**123**4509876**123**450	**Abnormal - correlation**
55889922**11**22**5511**6688449977	**Abnormal - correlation**

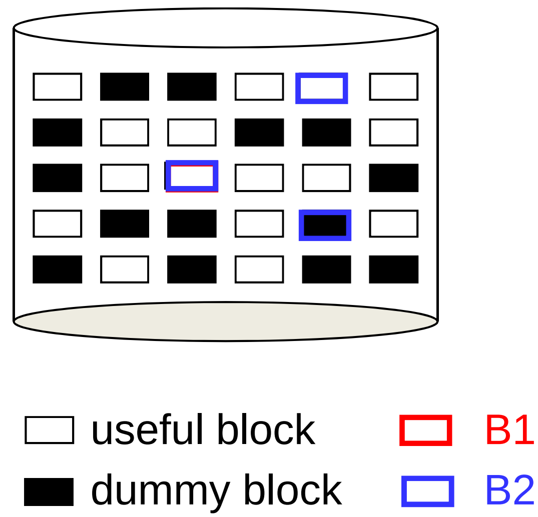
Real Updates

- *change the block's position each time it's updated*

```
Func real_update(B1)
do: randomly pick up a block B2;
if B2 = B1, then
    update on B1;
else if B2 is a dummy block, then
    substitute B2 for B1;
    update on B2;
else
    conduct dummy update on B2;
    goto do;
Func end
```

A Data block

Disk

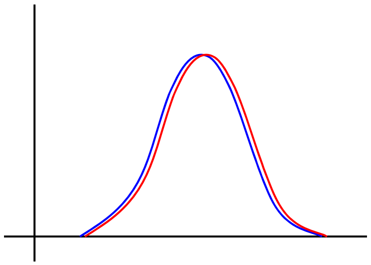


Proof of Security

- Real updates are also absolutely random:
  Each time, each data block has the same probability of being selected.

- pattern of Real updates = pattern of dummy updates

- Conclusion: secure



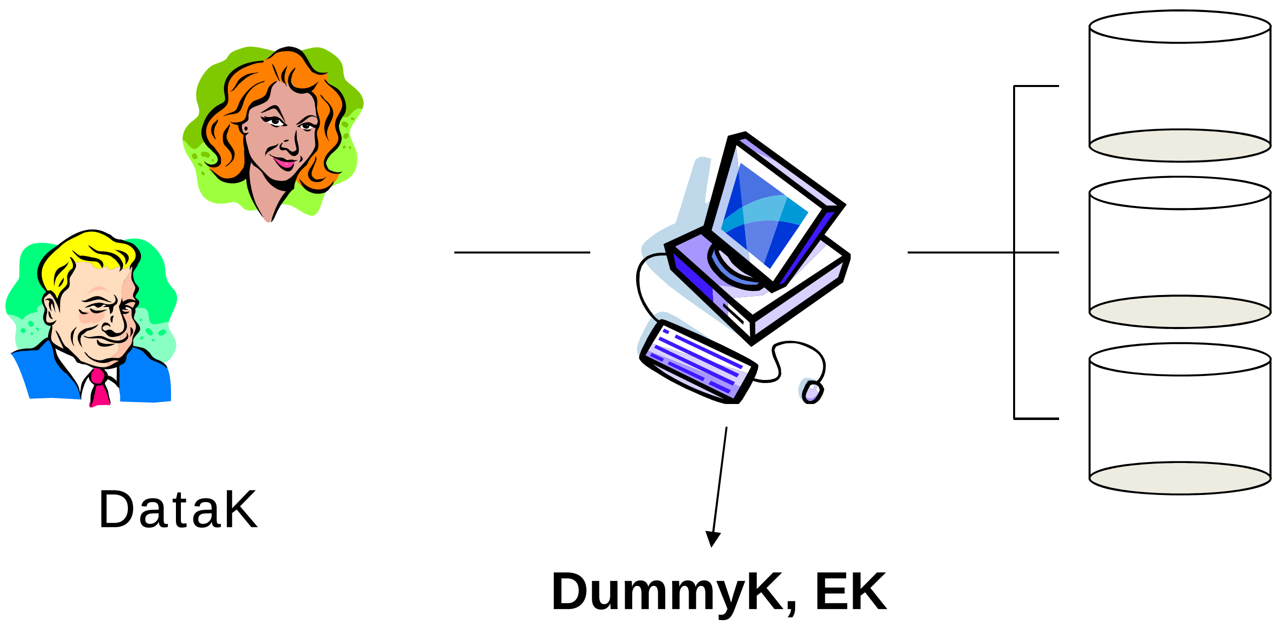
Processing Overhead

- Traditionally, each update requires 2 I/O operations – read and write
- With dummy updates, we need to repeat a block selection procedure until it successfully completes the update – each such operation requires 2 I/O
- N – Number of blocks, D – number of dummy blocks
- The probability of picking a dummy block is p=D/N
- The probability of repeating the selection i times is $(1-p)^{i-1}p$
- Expected number of repeats (overhead in terms of number of updates)

  $E = p+2p(1-p)+3p(1-p)^2+\ldots =$ **N/D**
- The more the dummy blocks, the better the throughput
  - Storage space is cheap today, we use extra space to exchange better performance
- File header needs to be updated when its data block is updated. But a file header need not incur I/O so frequently, as it can be kept in buffer

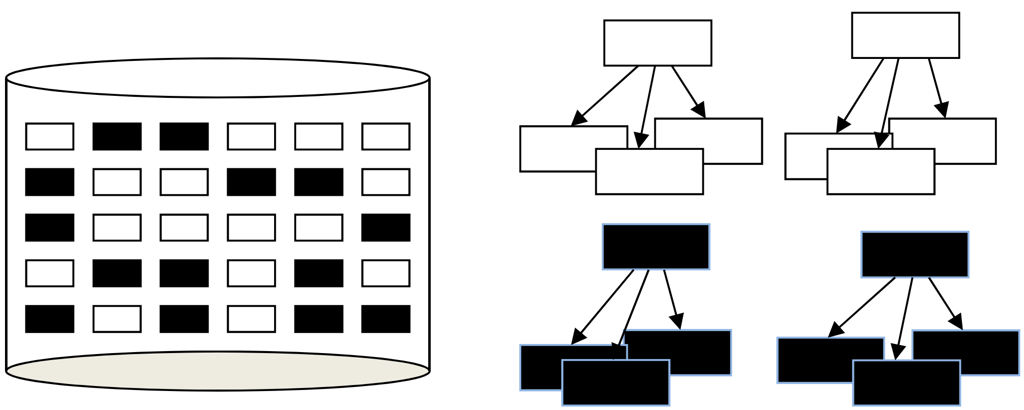
Key Management

- DataK - access key to identify a data file,
  DummyK - access key to identify the dummy file,
  EK - encrypting key
- If the agent is in a secure environment, it can maintain DummyK and EK.
- Otherwise, DummyK and EK are distributed to users.



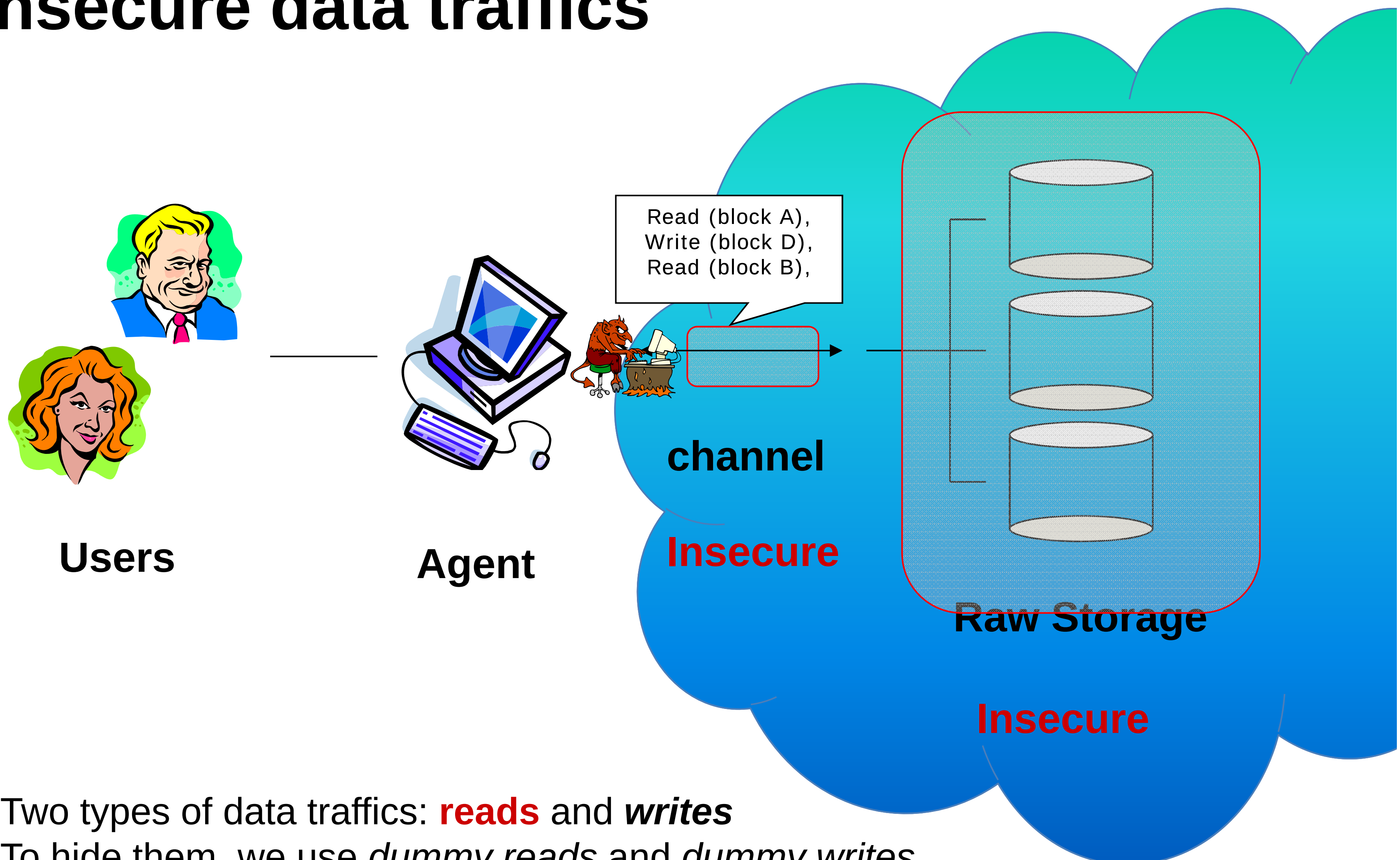
Distribute Keys to Users

- DummyK - Dummy blocks are organized into ***multiple dummy files***, and these dummy files are distributed to users
- EK - Each Data file or dummy file has its own encrypting key, which is given to user.
- Each user may possess several hidden files and several dummy files
- When a user logs on, he exposes all his hidden files and dummy files to the agent. The agent operates on the data blocks that users have exposed to it.



Multiple hidden files & Multiple dummy files

Insecure data traffics



- Two types of data traffics: **reads** and ***writes***
- To hide them, we use *dummy reads* and *dummy writes*
- ***Oblivious Storage! But very inefficient – 70 times the cost!***

Summary

- Steganography can be applied to hide the existence of files, resulting in Steganographic File System
- While steganographic file systems have been developed, it remains a challenge to realize a practical systems
- Data accesses in StegFS pose a threat
  - Updates analysis
  - Traffics analysis
- Other directions
  - More efficient scheme that can hide traffics
  - Distributed steganographic file system