CloudCache Design Document

October 9, 2013

Contents

List of Figures

1 Overview

CloudeCache is a utility for storing files among any other connected computers. It can be used for off-site file backups, transferring files to remote computers, accessing your same files anywhere in the world and sharing files publicly. Computer users will be able to leverage their unused hard disk space and network bandwidth to essentially acquire the aforementioned benefits at no significant cost.

2 Core Design Ideas

- remotes cannot be trusted with anything.
  - Everything should be encrypted as strongly as possible before being sent to a remote.
  - Everything stored on the remote should be verified frequently.
  - All claims by remotes (eg. bandwidth, uptime, etc.) should be verified using local experience and peer references

2.1 Backbone

- encrypt using encryptrsync
- send file to remote
- receive file from remote
- decrypt

3 Main Features

3.1 File Priorities

- different files or folders can be set to be backed up different amounts and in different ways (optimize latency over reliability, etc.)

3.2 Verification

- users can ask remote computers for hashes plus salts of their files. This can be used to ensure the remote computer is actually storing the desired files
- users can download and upload files at random times to verify remote bandwidth
- remote computers can periodically hash local files to ensure consistency
  - if consistency failures, primary users should be informed asap.

3.3 Expandability

- all operations by utility will have a utility version number associated so utility upgrades can be more easily backwards compatible
- allow plugins for
  - determining how trust is computed
  - custom bandwidth/cache/space optimizations

3.4 Decentralized DNS

- no DNS server is needed. Clients will communicate regularly and if their IP changes they tell their peers. URLs will be given preference if they exist.

3.5 Version Control

- able to recover previous versions of files
- able to merge files (deal with conflicts)
- able to view file history: who modified it when

3.6 Peers

- users are verified locally using certificates?
- can add specific users as trusted computers (friends, family, etc.)
- can share files to specific peers by encrypting with specific keys
- can share files publicly by not encrypting (how are file lists distributed?)
- users can view which peers have which files and how many backups exist
- users can allow access to groups of people
  - voluntary creation of signing keys. eg. keys for:
    - universities
    - governments
    - companies
- trust networks
  - validity of content can be verified through trust networks
  - trust metrics?

3.7 Bandwidth Optimization

- only transmit file deltas (use rsync?)
- transmits parts of files from numerous computers at once to take advantage of multiple users' worth of upload bandwidth

3.8 Cache Optimization

- store commonly accessed files on computers with high bandwidth (maybe even pay for a small amount of space on something like amazon s3)
- store files on remote computers that are near by to minimize latency, but not too close to risk losses from local disasters

3.9 Space Optimization

- files that are shared publicly don't need to be backed up as many times
- incremental backups
- user can specify how much space they want to use, so at some point incremental backups can be deleted
- users can specify if they want to store full backups as well

3.10 Encryption

- able to encrypt with encryptrsync for fast incremental uploads, but able to use stronger encryption if fast incrementals aren't desired
- encrypt local files for security purposes
- encrypt files before they are sent to a remote computer so the remote computer can store them, but not know what the files consist of
- encrypt files with different keys, which can be given out, thereby controlling outside access to said files
- users can access all files using only their password and ip address or url.
- allow re-encrypting files in case master, or peer key is compromised
- varying levels of encryption with high level explanations
  - encryption that likely can't be hacked by:
    - grandma (+0% storage usage and +0% transfer times)
    - ordinary citizens (+1% storage usage and +5% transfer times)
    - large organization (+4% storage usage and +10% transfer times)
    - powerful governments (+8% storage usage and +20% transfer times)
    - powerful governments for the next 25 years (+16% storage usage and +40% transfer times)

4 Secondary Features

- git like version control?
- nice gui
- interface for adding annotation/comments/voting/editing?
- maintain file metadata. eg. file permissions, hardlinks, etc.
- annotation/comments of files, parts of files, and versions of files from peers and certain groups of peers
    - voting/rating on files/comments
- works on desktops and cell phones
- trending content both from friends and public
- distributed search
- distributed recommender system
- anonymity

5 Economy

- By default, users can **elect** to trade their services for other user's services eliminating the need for money in the transactions (very low barrier to entry). Trading HD space is mutually beneficial, as users will gain off-site backups, backups spanned over multiple HD's and increased file transfer speeds (files are spread across many computers).
- users can publish their HD space, max up and download bandwidth, up and download bandwidth reliability, up/downtime. With this information other users can bid to keep their files there.
- users should be able to maintain a bidding profile so pricing can change dynamically
- bidding should be conscious of pricing of industrial options eg. amazon s3
- users can store the following time series of information about remotes
  - upload and download rates
  - file verification successes and failures
  - up/downtime
- users can ask potential peers for a list of their past peers. Users can then contact those peers and download their time series to verify the remotes claims of bandwidth and uptime. Mainly users would only communicate with a number from 1-10 indicating how **truthful** they find the remotes published stats.
- if a remote has lots of peers that trust them then their ratings of other peers can be trusted more (somewhat like pagerank algo?)
- would need some sort of distributed crawler to get trust numbers for many users?
- allow a trial period where dummy data is uploaded to a remote and trust is measured
- allow trust to be computed in many different ways (plugin), one of which includes a user input.

6 Processes and Data Structures

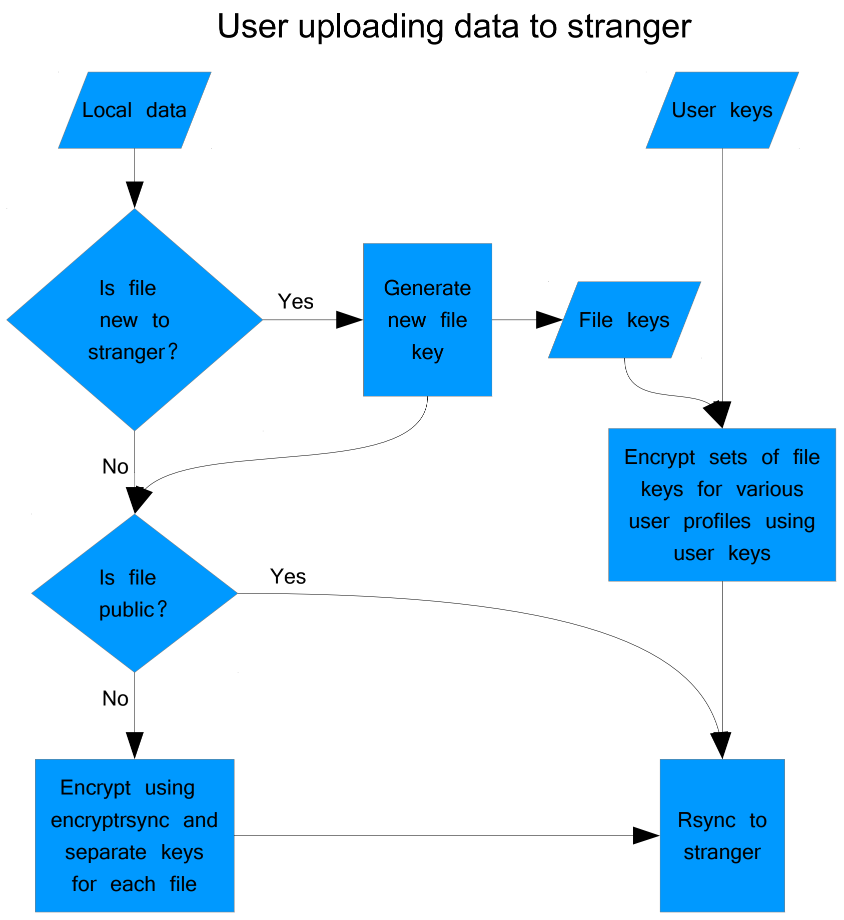
User uploading data to stranger
Local data
User keys
Is file new to stranger?
Yes
Generate new file key
File keys
No
Encrypt sets of file keys for various user profiles using user keys
Is file public?
Yes
No
Encrypt using encryptrsync and separate keys for each file
Rsync to stranger

Figure 1: User uploading to stranger flowchart

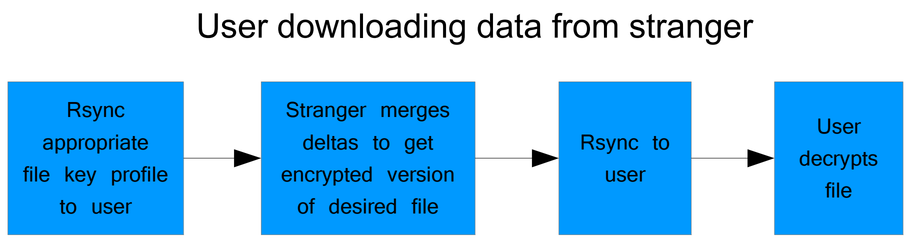
User downloading data from stranger
Rsync appropriate file key profile to user
Stranger merges deltas to get encrypted version of desired file
Rsync to user
User decrypts file

Figure 2: User downloading data from stranger flowchart. (Note: deltas are computed on encrypted files and not decrypted files. This reduces the amount of data that needs to be transferred while not revealing any data to the stranger. For example, it makes step 2 in this flowchart possible. If a given file is always edited in the same place, then the cumulative delta could be very small. This would not be possible if the deltas were computed on a decrypted file and then encrypted and then uploaded.)

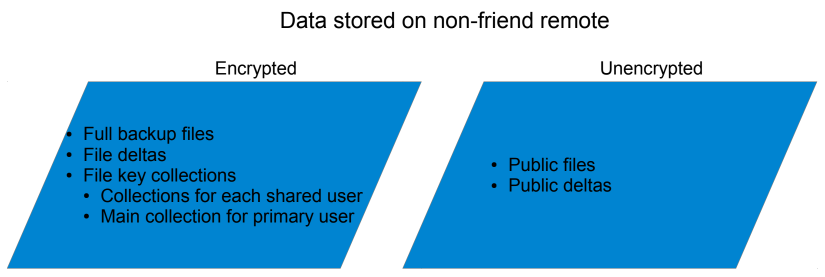
Data stored on non-friend remote
Encrypted
Full backup files
File deltas
File key collections
Collections for each shared user
Main collection for primary user
Unencrypted
Public files
Public deltas

Figure 3: Data stored on non-friendly remote

7 Programming Tools

- python (use Kivy)
- HTML 5 app?
- rsynclib
- curl
- encryprsync?
- duplicity?