On the Secure and Safe Data Synchronization

Pavel Ocenasek and Jaromir Karmazin

Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic
ocenaspa@fit.vutbr.cz, xkarma06@stud.fit.vutbr.cz

Abstract. This paper deals the aspects of data synchronization. The first part focuses on existing technologies and their features. We follow with the proposal of application that can be used as an alternative to the existing solutions. The proposed peer-to-peer application includes several safety improvements as well as it supports secure communication and data storage.

Keywords: Synchrnozation, security, safety, cloud, networking.

1 Introduction

Data synchronization becomes popular way how to share documents, personal information and program data between computers and also mobile devices. Dropbox is one of such program. This program is server-based and therefore we have to connect the synchronized computer to the internet (with access to the Dropbox central server). The situation is similar with the other available programs (SkyDrive, SugarSync etc.) Furthermore, these programs do not provide secure data storage. The data is stored usually in an unencrypted form, available to the server owners.

To allow for such an alternative, we have designed a synchronization tool whose network model is peer-to-peer rather than client-server. This allows the user to interconnect their devices in a way they see fit, be it a star topology with the user's own server, or a fully meshed topology of laptops and smartphones [3], none of which is permanently online.

2 Dropbox

Dropbox [6] is a popular cloud synchronization tool with client implementations available for Windows, Linux, Mac OS, Android [3], iOS, BlackBerry and Kindle Fire. To the user, it presents itself as a special type of folder (a „Dropbox“) whose contents are automatically mirrored to the Dropbox server and all other devices.

2.1 Introduction

The Dropbox service uses a centralized client-server model [6], requiring the user to create an account on the Dropbox server before they may synchronize their files. To

L. Marinos and I. Askoxylakis (Eds.): HAS/HCII 2013, LNCS 8030, pp. 326–331, 2013.

create an account, the user supplies their first and last name, e-mail address, and password of choice. With the default free plan, they receive 2.5 gigabytes of usable space after signing up.

The user can then install a client application on any of the supported platforms [3] and link it to their user account. The client application will then run a background service that watches for changes on the server and downloads them to the client, and that also watches for changes on the client and uploads them to the server. When the service starts up, it also checks for any changes that may have happened during the time it was stopped [6].

2.2 Features

Aside from synchronization, Dropbox allows viewing and downloading stored files using a web interface, sharing files with other users, using an API [6] to support third-party applications, and other additional functions.

When two devices linked to the same amount discover each other's presence in the LAN (using broadcast messages), they may begin a so-called “LAN Sync”, which allows one client to download data from another client instead of the server, potentially gaining speed. Both client still need Internet connection to coordinate. The change-originating client still needs to upload its version of the data to the server, so only the download process is accelerated using LAN Sync. [6]

The user can choose specific subfolders of the Dropbox folder to synchronize. This affects only downloads, not uploads. The mobile versions do not download whole directories; instead, they only download files that are marked as “favorites” to save storage space. [6]

2.3 Security

To secure a user's account, Dropbox provides the following services that are available in the account settings [6]:

- Password changing. Obligatory and self-explanatory.
- Two-step verification. This extends user authentication to include one-time codes, either random numbers sent to a mobile phone using text messages or Time-based One-Time Passwords (TOTP) generated by a mobile application. The one-time codes are required for web sign-ins and for linking new devices.
- Notifications. The user may choose to receive notifications by e-mail whenever a new device is linked to their account, and/or whenever a new third-party application is connected to their account.
- Device management. The user may review the devices linked to their account, and unlink those deemed to be illegitimate.

The web interface, as well as the network connections from the various client applications to the server, are secured using the standard SSL protocol with AES-256 encryption, using

a certificate signed by Thawte, Inc. (as of 2013). This serves to secure the user credentials and file data in transit. [2]

Dropbox claims to “use modern encryption methods to store [the users'] data” , using what is assumed to be AES-256 encryption, though the encryption key cannot be specified by the user ; in fact, we can assume there exists a single global key for decrypting all users' files, since all files can be shared with other users. Dropbox, Inc. even states that they “have a small number of employees who must be able to access user data”. Therefore, one can have good faith that the Dropbox, Inc. employees will not look at one's files, but the security model does not protect the files from law enforcement officials or malicious hackers. [6]

On 19th June 2011, a bug affecting the authentication mechanism caused all Dropbox accounts to be accessible without a correct password. The bug was fixed in about 4 hours. User data that did not use client-side encryption could be readable to anyone during that time period.

To encrypt one's data in a way that even Dropbox, Inc.'s employees or law enforcement officials will not be able to read it, one can add a layer of encryption to one's client system, though this is not officially supported by Dropbox. The list of third-party encryption layers includes EncFS, BoxCryptor, Viivo,

The client application stores all downloaded files as they are, inside the folder designated to Dropbox. If the user wishes to encrypt the Dropbox directory, they have to use a lower-level tool, such as TrueCrypt or dm-crypt, to encrypt the storage partition where the Dropbox directory resides. [6]

2.4 Summary

Dropbox is a user-friendly service that is well suited for synchronizing and sharing non-sensitive data. For sensitive data, one can rely on transfer security somewhat, but the storage security of Dropbox is dubious.

3 Peer-to-Peer Sync

While Dropbox or similar cloud-based synchronization tools [1] may be a suitable choice for the average user who values ease of use more than full control, a more skilled user might desire a tool that can be set up in different configurations, perhaps not requiring a company to store the user's data or even not requiring an Internet connection at all.

The target users of our application are people with moderate computer skills who own multiple electronic devices capable of storing data and connecting to IP networks (such as desktop and laptop computers, tablets, and smartphones). We make no assumptions about the uptime of each device or the stability of their IP addresses. In other words, the application should support not-always-on and mobile devices.

3.1 Challenges

Because there might not be a central element in our peer-to-peer system, we cannot assume the presence of any central authority. This means that modification times, for

example, cannot be reliably used for comparing file age because there might be a difference in the devices' internal clocks. Conflict resolution also becomes complicated because of this – first, when a file is modified differently on two devices, it cannot be reliably told which change happened later, and second, devices might clash when trying to resolve the same conflict on both ends. Lastly, a device can easily become out-of-date if it does not connect to the network for a long time, or if during a device's uptime, no other device is online to synchronize with. [1]

3.2 Inspiration

The main inspiration for this project comes from Git [5], the distributed version control system that could, from a certain point of view, be seen as a peer-to-peer synchronization tool for source code. A typical Git project consists of several repositories scattered over different computers, exchanging information with each other using push/pull operations. All repositories appear equal to Git and any repository can generally transmit data to any other, although software projects usually provide policies that restrict the directions data can be transferred. These policies also usually determine where and how conflicts should be resolved. [5]

To allow easy transfer of information between repositories, Git stores all its information as "objects" in a content-addressable database whose keys are SHA-1 hash values of the data stored. All objects in the database are permanent and non-modifiable. Because the collision of two SHA-1 hashes is extremely improbable, the hash value is all that needs to be known to retrieve an object from the database. [5]

There are three types of objects in a Git repository [5]:

- "blobs", containing only data,
- "trees", containing file names, directories and references to blobs, and
- "commits", containing version information, references to previous versions and to a tree.

The references (which are mere SHA-1 hashes of the objects referenced) allow the object database to be represented using a directional acyclic graph (DAG). This mitigates the need of synchronized system clocks – the age of two file versions can be compared using their parent-child relations.

By using object references, changes in files can be stored by creating new objects that reference the old ones as their predecessors. This means that synchronizing two Git repositories consists of very little more than copying over the missing objects. On the downside, this means that a Git object database is constantly growing in size. [5]

3.3 Application Design

Our application uses an object database similar to Git [5], with a few key references:

- Objects are identified by UUIDs rather than SHA-1 hashes, in order to make identifier generation faster for large files.
- Storing a copy of a file's data (called "blob" in Git) is optional.

- "Tree" objects are replaced by "metadata" objects which hold information about only one file each, including its name, path, attributes, size and modification time.
- "Commit" objects are replaced by "snapshots" that only contain a list of metadata objects and an optional parent reference.
- Versioning is shifted from the commit level to the metadata level, meaning that each file can be versioned separately.
- Multiple directories in the file system can be made synchronized "volumes", which are synchronized independently of others.
- An instance of the application monitors changes in the file system while it is running, and updates the object database accordingly. When starting up, the application scans the file system for changes that happened while it was shut down.
- Two instances of the application can form a network connection and transfer information in two ways:
- "Bulk sync" – by exchanging the latest snapshot objects, each device learns the state of the other and then iteratively downloads all missing objects.
- "Real-time sync" – when two devices are synchronized, they send each other new objects (metadata) as soon as they are created in their local database.

The network connection can be formed over either IPv4 or IPv6. For secure communication, the TLS session layer protocol is used. [2] We believe most users will not be willing to pay certification authorities for certificates for their own personal use, so the application allows using self-signed certificates, provided that their fingerprints are checked.

It should be possible to use an untrusted peer for secure storage by encrypting the objects sent to them. This would require all the trusted peers to share a common key. The untrusted peer would learn about the number, relationships and approximate sizes of the data stores, but not about the data itself.

3.4 Comparison with Dropbox

Our application has the following advantages over Dropbox:

- No external providers are required. The user's own devices can be used for storage, mitigating the risk of the provider looking at the user's files.
- No device needs to be always powered and online. As long as the devices get to "see" each other often enough, they will bee synchronized.
- Synchronization can be done over LAN without Internet connection, which means full transfer speeds (usually near 100 megabits per second).
- Data can be encrypted in storage, allowing the user to use untrusted storage providers.
- There are also drawbacks that make the application less favorable than Dropbox:
- The application is not as easy to use as Dropbox, mainly because the needs for designing a topology and distributing one's own cryptographic certificates.
- A device may become out of date if it stays long enough without synchronizing with another device.

- Conflicts must be resolved by the user, otherwise two devices trying to resolve the same conflict might clash.
- Large amounts of metadata need to be stored by each device.

Acknowledgements. The research has been supported by Technology Agency of the Czech Republic (TACR) in frame of the project SCADA system for control and monitoring RT processes, TA01010632. This project has been also carried out with a financial support from the Czech Republic through the project no. MSM0021630528: Security-Oriented Research in Information Technology and by the project no. ED1.1.00/02.0070: The IT4Innovations Centre of Excellence; the part of the research has been also supported by the Brno University of Technology, Faculty of Information Technology through the specific research grants no. FIT-S-11-1 and by the project MPO CR, FR-TI1/037.

References

1. Tridgell, A., Mackerras, P.: The rsync algorithm (1998), `http://rsync.samba.org/techreport/`
2. Stallings, W.: Cryptography and network security: principles and practice. Prentice Hall (1998)
3. Android Developers: Android SDK (2013), `http://developer.android.com/sdk/index.html`
4. Tridgell, A., Mackerras, P.: The rsync algorithm (1998), `http://rsync.samba.org/techreport/`
5. Chacon, S.: Pro Git. Apress (2009) ISBN 978-1430218333
6. Dropbox. Core API Development kits and documentation (2013), `https://www.dropbox.com/developers/core/sdk`