The Cloud

The Cloud is a cyber security company that designs private cloud solutions for file storage and systems against industrial and government espionage

The company is looking for international partners to expand its business area and establish relationships with entities interested in the development of technological innovation

Contact: +39 328 614 7882
Website: http://tc0.it/

Some realized software projects:

- **Secure Storage**

  The necessity and desire to secure personal information is one thing that everyone shares around the world in the recent times, ranging from businesses to governments to military structures. Data security is critical whether it is being stored, sent, or delivered. Data breaches, hacking, and lost or stolen devices can have catastrophic financial and reputational costs. The need for a Library to protect data generated and handled by applications arose from a desire to protect not only public structures, but also individual citizens, who are even more at risk if their freedom of expression, gender, religion, and any data relating to their person and loved ones is not protected.

  Any application that does not secure the data it generates and manages carries the risk of revealing sensitive information that can be used to profile users, scammers to invent scams, and hackers to carry out their plans to pirated programs. The information created by the applications can easily be gathered and marketed on the dark web.

  SecureStorage is a library that provides effective encryption to the apps that use it, making the data generated by it inaccessible and inviolable.

  Any application creates a large quantity of data; some of it serves just as a warning, while others are essential to the application's operation and users, and some of it, if interfered with, can allow the application and its content to be hacked.

  To protect yourself from malicious hackers and organizational data breaches, encrypt all data generated by the application and prevent it from being saved in a way that may be read externally. In the case that unwanted access is permitted to a computer network or storage device, other apps on the same device, or system applications designed with fraudulent purpose by the device's maker, encryption provides an extra level of protection. The hacker will be unable to access the application data encrypted through SecureStorage.

  What is encryption?

  Simply said, encryption transforms data entered into a digital device into gibberish-like pieces. The encrypted data becomes more unreadable and indecipherable as the encryption technique becomes more complex. Decryption, on the other hand, restores the encrypted data to its original state, making it readable again. Unencrypted data is referred to as normal data, and encrypted data is referred to as encrypted data.

Software vs Hardware encryption

Software encryption encrypts data on a logical disk using a number of software packages. A unique key is created and saved in the computer's memory when a drive is encrypted for the first time. A user passcode is used to encrypt the key. When a user enters the passcode, the key is unlocked, allowing access to the drive's unencrypted data. The drive also stores a copy of the key. When data is written to the drive, it is encrypted using the key before it is physically committed to the disk; software encryption works as an intermediate between application read / write data on the device. Before being given to the software, data read from the drive is decrypted using the same key.

Hardware - level encryption is possible on some devices: Hardware - based encryption is used in Self - Encrypting Drives(SEDs), which takes a more comprehensive approach to encrypting user data. SEDs include an AES encryption chip that encrypts data before it is written to NAND media and decrypts it before it is read. Between the operating system loaded on the drive and the system BIOS is where hardware encryption takes place. An encryption key is generated and stored on NAND flash memory when the drive is encrypted for the first time. A custom BIOS is loaded when the system is first booted, prompting for a user password. The contents of the drive are decrypted and access to the operating system and user data is provided once the pass is entered.

Self-encrypting drives also encrypt and decrypt data on the fly, with the built-in cryptographic chip encrypting and decrypting data before it is written to NAND flash memory. Because the encryption procedure does not use the host CPU, the performance penalty associated with software encryption is reduced. The encryption key is typically placed in the SSD's built-in memory at system startup, which complicates recovery and makes it less vulnerable to low-level attacks. This hardware-based encryption solution provides strong data security in the event that the device is lost, cannot be disabled, and has no performance impact. However, it is a type of low-level encryption that is completely transparent to the device that uses these storage units, as well as to all software programs that run on the device. As a result, this type of encryption does not protect the data of individual applications and users from other resident programs that can see all of the data stored in clear text.

SecureStorage provides an additional layer of security for individuals who utilize primary hardware encrypted devices, rendering the data unreadable outside of the single program that created and is using it.

The Advanced Encryption Standard (AES) is a cryptographic technique that is based on the Rijndael family of algorithms. It is now one of the most widely used encryption and decryption techniques. Vincent Rijmjen and Joan Daemen created the Rijndael algorithm, which is a block cipher. It's a symmetric-key algorithm, which means it encrypts and decrypts data with the same key. As a consequence of the NIST Advanced Encryption Standard competition, the Rijndael algorithm was chosen as an Advanced Encryption Standard and the successor to the Data Encryption Standard (DES). The competition was held in order to produce a new cryptographic standard as a replacement for the obsolete DES. Because to the modernization of computer technologies, the Data Encryption Standard's key length (56 bits) was insecure at the time. The

Rijndael family of functions is represented by three algorithms in the AES standard. They have varying key lengths of 128, 192, and 256 bits, but they all use the same 128-bit block length. More variations of encryption algorithms, cyphers, and other cryptographic functions are included in the Rijndael family of hashing functions than in AES. The Advanced Encryption Standard was designed to work equally well in software and hardware implementations. With the deployment of the substitution–permutation network design, it was possible. This network design is similar to the Feistel network, which was utilized in DES, but it is faster to compute on both hardware and software, which was critical given DES's software implementation inefficiency.

Our cryptography is the same as that used in Bitcoin, which has been put to the test by hackers all around the world without ever being broken: Breaking this form of cryptography would give you access to coins stored in wallets, which no one has ever done before.

The Advanced Encryption Algorithm (AES256) is an AES algorithm with a key length of 256 bits.The computational difficulty of the decryption is affected by the length of the AES version. The key recovery for AES 256-encrypted data requires more computational power than the 128 and 192-bit variants. The biclique attack, for example, can decrypt AES128 with a computational complexity of 2126. The computational complexity of biclique attacks on AES 192 and AES 256 are 2189.9 and 2254.3, respectively.However, for every key length, real execution of the attacks on the AES-protected data is currently impractical. All of the AES attacks are hypothetical. Every known AES attack would take millions of years to complete, regardless of the algorithm's key length.

- **Encrypted Messaging**

  Encrypted communication library to create applications similar to Telegram or Signal but with greater attention to IT security and privacy

  This project uses the Communication Channel project as its underlying, which creates a shcket communication channel for transmitting and receiving encrypted data upstream from the library. Communication Channel underlies the encrypted messaging protocol, we have separated the two parts because the idea is to provide an universal communication protocol, which can work on any type of communication medium and hardware. Communication Channel creates a tcp socket communication channel, but this underlying one can be replaced with an analogous communication channel working with GSM data networks (without using the internet), or with rs232, rs485 ports, or any other communication devices either digital and analog. Just change the underlying encrypted communication protocol and we can easily implement encrypted communication on any type of device and in any scenario. If necessary, we can create implementations of new communication channels on different hardware, on commission.

  Encrypted Messaging is a data exchange library between any type of device, usable for both desktop, mobile and internet of things applications:

  - Support of digital signature on packages, and security level in military standard.
  - This library, in terms of functionality and type of use, currently has no analogues.

  Examples of use:

  - Encrypted communication to create applications similar to Telegram or Signal but with greater attention to computer security and privacy.
  - Data transmission between device and cloud.
  - Acquisition of telemetry data produced by equipment.
  - Connecting the Internet of Things and wearable devices to the cloud.

  The library works correctly under all kinds of circumstances and data lines, and is ready for production scenarios,

  Our mission is to exacerbate the concept of security in messaging and create something conceptually new and innovative from a technical point of view. Top-level encrypted communication (there is no backend , there is no server-side contact list, there is no server but a simple router, the theory is that if the server does not exist then the server cannot be hacked, the communication is anonymous, the IDs are derived from a hash of the public keys, therefore in no case it is possible to trace who originates the messages, the encryption key is changed for each single message, and a system of digital signatures guarantees the origin of the messages and prevents attacks "men in the middle"). We use different concepts introduced with Bitcoin technology and the library itself: there are no accounts, the account is simply a pair of public and private keys, groups are also supported, the group ID is derived from a hash computed through the public keys of the members, since the hash process is irreversible, the level of anonymity is maximum). The publication of the source wants to demonstrate the genuineness of the concepts we have adopted! Thanks for your attention!

- **Communication Channel**

  Communication Channel underlies the encrypted messaging protocol, we have separated the two parts because the idea is to provide an universal communication protocol, which can work on any type of communication medium and hardware. Communication Channel creates a tcp socket communication channel, but this underlying one can be replaced with an analogous communication channel working with GSM data networks (without using the internet), or with rs232, rs485 ports, or any other communication devices either digital and analog. Just change the underlying encrypted communication protocol and we can easily implement encrypted communication on any type of device and in any scenario. If necessary, we can create implementations of new communication channels on different hardware, on commission.

- **CloudBox**

  Multi-purpose multi-platform library to create both server and client cloud systems on the fly, this library allows you to virtualize a cloud system on the fly.
  This library has been created to provide all the functions of both client and server cloud storage, both for small embedded systems and for large infrastructures: The possibility of being able to mount cloud units on the fly (hot create and destroy) allows you to create powerful infrastructures cloud as well as simple cloud si minilali systems.
  The prerogative of this library is that it is a symmetric library for server and client, i.e. this library is valid for both server and client cloud functions, as the platform was designed to be symmetric, i.e., the same code allows to create instances of cloud client and cloud server, giving rise to great flexibility as, for example, machines that using this library can act as both clients and servers with little intervention on the part of the developer.

CloudBox through the underlying CloudSync library implements a low-level data synchronization system between client and server which saves on transmitted data, as the packets are minimalistic, it is a purely binary protocol which does not add anything to the bare essentials for transmission and data synchronization, which does not happen, for example, for protocols that encapsulate data in json or xml structures.
The underlying encrypted messaging libraries add to this library an extreme security based on symmetric key protocols deriving from bitcoin technology and from bitcoin derive all the concepts in terms of security and trustless: Both client and server machines, when instantiated, do not require any user account, therefore there is no place where user databases and authentication data are kept, the client and the server are instantiated with exactly the same system used for bitcoin wallets (and to do this we use the same technology), i.e. at when the instance is created, a passphrase is randomly generated which allows account recovery, and which acts as a generator for the cryptographic keys (public and private), the private cryptographic key will never leave the device while the public one is used by the client for be able to access the server with a PIN. In this type of technology, the private key as generated also represents a sort of inviolable digital identity, which also has a digital signature to authenticate documents and data packets, in fact, during the synchronization procedure, the data, in addition to being encrypted, sees the addition of the digital signature to ensure maximum certainty on the origin.
This type of technology is known and appreciated by users of cryptocurrency cold wallets such as Ledger and Trezor, who are considered among the most secure IT solutions for storing digital coins: Since the violation of digital wallets represents a great reward for hackers who succeeded, and therefore it is reasonable to believe that this is the best technology to secure the data.
In practice, the server generates a QR code that represents its public key which allows a client to establish a cryptographic connection with the server and communicate with it in a secure manner.
To ensure that the private key can never leave the device, we have developed a special library (SecureStorage), which uses the best possible technologies to prevent private data from being accessible, including the hardware components present in modern devices to save key-data pairs.
The communication protocols implemented by the cloud through this library are two, a binary socket that represents the best possible technology in terms of performance, and one derived from the Rest API technology, to which we have added an encryption level that would not be present in standard technologies which relies on https requests to secure data, a bad solution because it allows the machine receiving the https requests to see the data in clear text, risking the security of everything that passes through: Https protects the point-to-point traffic on the internet but not inside the machine that receives the data or sends them through, having added an encryption layer we are going to create a protected tunnel that is not simply from machine to machine, but from client application to server application and vice versa, none of everything in the middle can intercept in plain text what passes through.
The main functions provided by the library are:

- Instantiate cloud clients and servers on the fly.
- Establish connection between client and server using a software router as a hub.
- Provide real-time data on synchronization status, data transmission and network problems.

    - Automatically manage synchronization by means of a specific underlying library.
    - Administrative functions of instanced clouds (server and client).
    - Create sub clouds for internal areas.
    - Digital signature functions on documents using the private key that represents the digital ID of the instance.

  Notes: This library, to work as a server, needs the CloudServer library which adds small aspects that are not necessary in the client for obvious reasons, such as for example the generation of thumbnails, the exposure of encrypted APIs and the management of a proxy to directly expose the machine to internet while using the API.

- **CloudSync**

  The Cloud Sync library is a highly specialized and scalable library dedicated to synchronizing data between local and remote storage, where the local device is intended as the client cloud and the remote one is the cloud server. The library is multipurpose, i.e. it is the same for both client and server systems and its utility is to monitor local and remote files and synchronize changes in real time so as to always have a redundant copy of the data you are working on , on the cloud.
  The synchronization library is placed at a higher level than the CommunicationChannel Library and the EncryptedMessaging, and uses the characteristics of the latter to be able to expand with a binary command language, minimalist in data transmission and which therefore allows optimal use of data via the Internet by optimizing the use of bandwidth. EncryptedMessaging also offers a level of encryption and digital identity based on Bitcoin technology, making the communication protocol secure, with a type of encryption transparent to the developer that integrates the commands in the Cloud Sync library. The CommunicationChannel instead works as the underlying at a lower level to manage the TCP socket connection via the internet, theoretically we can replace this library with one specialized in GPRS, or RS425 communication and with relative simplicity we can adapt the Cloud Sync library in scenarios where instead of the internet as a communication channel, we have instead a GSM line or a serial communication or a transmission via modem. Having therefore unpacked the cloud technology into multiple libraries, we can relatively easily adapt a cloud project and different scenarios for which different hardware and technical solutions will be implemented.
  The Cloud Sync library is technically an underlying of the CloudBox library and adds to this an efficient and fast data synchronization protocol which takes place by means of a set of commands at the binary level which are transmitted between client and server, to check the synchronization status and proceed with updating the files if necessary. Another task performed by the library is the management of three types of events: Events on the current state of synchronization, events on file transfer, input output events (i.e. events that are triggered upon receipt and sending of communication between client and server), events about file errors, and events that the antivirus system could trigger during the synchronization of presumed infected files. This series of events can launch code that is executed on a possible app with user interface, which uses the library, in order to show the user in real time what is happening during synchronization and inform him of any problems found and on the synchronization progress.
  This library also deals with the management of credentials useful for user login (when

connecting the client to the server), the management of roles, and provides a class with all the utilities for managing access pins and other things useful for synchronization .
Synchronization takes place through data analysis both on the client side (user) and on the server side (the cloud), and an algorithm optimized for the transmission of the least amount of data between machines, as soon as it is clear what the differences between client and server are, puts in scheduler all the operations to be performed for synchronization, which will be immediately performed, and resumed in the event of an error or drop in the communication line.
In any case, if the client should not be online, the synchronization will start automatically as soon as the connection will allow to calculate the differences between client and server and what needs to be updated.

- **Trustless cloud client**

  The project is centered with a military safety standard.

  Easy to use Cloud Client + Backup Drive (Bonus)!

  Cross-platform desktop cloud client, for compatible clouds (works in conjunction with cloud server to sync files).
  This project has dependencies with other libraries which you can find open source in the same account as separate projects as the underlying libraries are in common with other projects:
  Therefore, the projects not included can be added by searching for them on the github repository, some are also available as nughet packages (you can add them to the solution instead of the missing projects).
  If you don't have the Cloud Server, alternatively you can use this software with a network address as a remote repository (the path must be set in the "git" parameter under the backup settings), for example you can set a pen drive connected to your router (in this case, the samba network path of the pen drive must be entered in "git").

  Description

  To use this software you need to have the relevant Private Cloud and compatible Cloud service (otherwise where does the cloud connect?).
  This program is an open source desktop cloud client to automatically synchronize, encrypted and with military grade security, files from your PC to your private cloud or cloud service.
  The synchronization algorithms are very fast and the software with hundreds of thousands of files does not go into crisis as it happens with similar products.
  This is an open source product and is published here in exactly the same source format version as you find here: https://github.com/Andrea-Bruno/CloudClient without any additions or modifications.
  Respect for your privacy is total, and the military level of security protects you and your data from hackers who would like to sneak into the cloud to access your personal data and information.

  Safety:

  We have transmitted our experience in the development of "non custodial wallet" bitcoins to this application so the underlying uses the same concepts and the same libraries, which are the foundations of the trustless technology used for the blockchain (the maximum current concept in terms of security) . The application generates a passphrase which creates a pair of cryptographic keys (public and private), which represent your digital identity and you can also sign documents with it. This digital identity is used by the server to recognize you, it is the underlying for encrypted communication. Using the passphrase you can restore your account on the client side, just like cryptocurrency wallets. Just like with bitcoin wallets, your account is client-side only, there is no website where you need to register or a place where your account is kept, which makes this application conceptually superior to all similar ones.

As a bonus we have added some extra features:

- Automatic virus detection (occurs at the same time as cloud file synchronization).
- Daily automatic backup: To take advantage of this function you need to have an additional physical HD since the backup done on the same disk is useless because it does not protect against physical disk failure. The backup uses hard link functions so it doesn't take up much space, new backups only take up the difference of what has changed since the previous backup was done.
- Versioning: A new backup is created every time a file is modified in order to keep its previous losses and allow rollback (very useful function for software developers).
- Synchronization of the cloud area on the pen drive or disk attached to the router: You can specify the network path of your device connected to the router's USB port and have a real-time synchronized copy of all your data so if your computer were stolen or should you lose it, you will still have a copy of all your data.
- Digital signature of documents, i.e. the software creates a digital identity with which you can sign documents and validate the signature affixed by others.

The software needs to run in administrator mode for the following reasons:

- Automatic date and time adjustment in your computer (if the date is wrong the files will be recorded with wrong dates and could be mistakenly mistaken as older than versions contained on the cloud).
- Create hard links for backups (this saves a lot of space during backups).

Privacy Policy

The application does not collect or send personal data, all communication is exclusively with your private cloud or cloud service.
Since the privacy policy is trustless (you don't have to trust us but it is the code that demonstrates honesty), the source code irrefutably demonstrates that our work is sincere and loyal and that your data is protected, and the communications between client and Cloud server are impossible to intercept because they are covered by military-grade encryption systems with digitally signed packets to prevent "man in the middle" attacks in a preventive manner. The versions that we publish on the store are the same (without any modifications) that you can find here in source format.

- **CloudServer**

  Scalable and cross-platform cloud server technology

  It is a library that allows one or more cloud systems to be instantiated on the fly, creating a scalable structure. Since this is a library, it has no graphical interface which must be developed by the developers based on the specifics of the required scenario. Having decoupled the client

functions from the graphical interface, above this library it is possible to apply a graphical interface based on the technology that best suits the specific purpose of the cloud solution to be created, for example it is possible to add a web interface to create a cloud kiosk mode, or a cloud managed by windows panels, or purely software clouds whose interface is on a remote mobile app, etc.
Essentially CloudServer is a superior layer to the CloudBox library which being both a cloud server and client library does not implement the specific features only for the server, i.e. CloudBox only has common functions between client and server, while everything that is purely specific to the server (the cloud itself), is added with a software layer thanks to the CloudServer library.
The Cloud Server library implements server-specific functions, is written to be universal, i.e. it can run on both Windows and Linux operating systems, and adds features such as file thumbnails, license management, on-the-fly instantiation of new cloud, access pin management functions, diagnostic functions, generation of various types of QR code to pass the public key to the client and allow it to connect, descriptive report on system status, API for communication with a second encrypted protocol of communication (in addition to that provided by the underlying socket-type system), which allows a client to communicate with it through an innovative revamping of the Rest API protocol, to which encryption and key exchange with asymmetric encryption has been added, yes it is a protocol for communication with our encrypted proxy technology (normally a server that receives json requests, even if these are https, would be able to clearly see everything that passes because https creates a tunnel only up to the machine with which you interact with the post and get methods, instead encrypting the data that transits, with our solution the proxi will only act as a pass through without having the possibility of knowing anything of what transits.
Why have we implemented a protocol that requires a proxy? In many scenarios, for security reasons, you may not want to expose the cloud machine to the Internet, and prefer that the use of the post and get methods with the APIs take place via a machine exposed outside the infrastructure that acts as a proxy by resending the encrypted commands that receives in socchet mode to the cloud within a company intranet, the concept is to increase security by not exposing the cloud directly to the internet. In any case, with our software, the communication with the APIs must go through a proxy, which must have a static IP, while for the cloud the static IP address is not necessary since it is he who establishes the connection first, in this way it is also possible to create cloud servers that can work without a static IP or in areas covered by a firewall and interact with it from the outside using our encrypted proxy technology.

- **Data Redundancy library**

  It is a library that allows you to synchronize files in real time on a local or remote device (data mirroring with the addition of intelligent merge functionality).
  Being a library, this is not a software component that can be used directly but must be included in a solution in order to add the functions that we will describe below:
  The idea is to keep a copy of a directory with all its structure on a remote network device, as a baskup unit and with some unique and interesting additional functions:
  In the context of working in cooperation with multiple users, especially in the context of software development, it happens that you want to work on the same files, but there is no software that allows you to do it automatically and create the merge of files in real time.
  This library was initially created as a patch for what git doesn't do, which is the automatic merging of work between multiple users. Essentially this is a library designed to work in software development but it works with any kind of software that is able to work cooperatively

with multiple users, (the merge is a function that in software platforms is always done manually because it would require a very complex algorithm to do it automatically because there are many factors to consider, we have created this algorithm!).
The concept is this: All users who wish to participate in a common and collective work must configure the same network GIT path in which the files on which they work in a shared manner will automatically be maintained and updated.
Operation is very simple as users will never have to work on files in the shared directory on the network (called GIT directory or network path), which acts as a real-time mirror of the shared files stored locally. In practice, each user continues to work locally on his own files (as he normally would) and when these are modified, AntyGitLIbrary will automatically update the remote mirror, and if necessary it will merge the document with the texts edited by multiple users. At the same time when a file in the network directory is updated, all users using the library-based application, if connected, receive in real time an update of the local file with the one just updated remotely, so that everyone works on the latest version of documents and files generated by applications. If a user is not logged in (the computer is turned off), he will still receive the update the first time. In practice, whenever someone locally adds a new document or a new photo or other, or modifies an existing document, a copy is sent to the remote GIT repository and then from there it is propagated to all users participating in the same group using the app based on the Data Redundancy library. The User can also work from a laptop workstation from another place and when he will be in the office or where the git directory will be visible, the software will synchronize the data, in practice the documents are always synchronized working in the same workplace, or they will be synchronized when possible on the first occurrence, all this automatically, i.e. without any necessary action on the part of the user.
A very versatile and powerful function that has no equal is the code merge function, expressly designed for developers who use VisualStudio, but which can work in all contexts: The software monitors when a file is modified and if it has been modified by more sources, merges the parts added by all members who have worked on the same file, basically the files automatically get the additions and changes made by other users. One could believe that this function could be unwelcome during software development as a non-definitive code could make the software non-compilable to other users, but this is not the case as the merge is performed when the software is compileable, and not during writing the code. In practice, when text is added, it is synchronized with the file on the remote git directory, only when the compilation is successful, and therefore the changes will be propagated to all users only on the really working code (the additions that do not produce compileable code, will not be propagated).
However, if you think your code might interfere with that of other developers working on the same project, you can hide it from the compiler using a custom symbol and #IF USER1 statements, to hide the effects of your code from other users. still in progress and not definitive.
In some scenarios you may want to use a git directory in a remote infrastructure (outside your intranet), in which case you can do it using a VPN.

- **Backup Library**

  The backup library is a cross-platform library that makes it easy to integrate powerful backup functions.
  The purpose of this library is to make available and automate two types of backup: the daily one and the versioning one which is activated automatically whenever a file is modified.

The library doesn't perform a brutal backup as it would take up a lot of space but it performs an intelligent backup that is completely transparent to the end user, for which it seems to all intents and purposes a full backup copying all directories and contents.
The intelligent backup works that first the software analyzes the entire structure composed of files and directories and compares it with that of the previous backup, and the new backup will copy only the modified files while for those that have not undergone changes a link will be created (in the file system) to the same file as the previous backup: This type of solution is full backups but in reality they take up little space due to the fact that only the differences actually create files in the backup structure, while for everything that has not changed there is a link to the data of an identical file that transparently looks like a file for all intents and purposes.

  - Symbolic link
    A symbolic link contains a text string that is automatically interpreted and followed by the operating system as a path to another file or directory. This other file or directory is called the "target". The symbolic link is a second file that exists independently of its target. If a symbolic link is deleted, its target remains unaffected. If a symbolic link points to a target, and sometime later that target is moved, renamed or deleted, the symbolic link is not automatically updated or deleted, but continues to exist and still points to the old target, now a non-existing location or file. Symbolic links pointing to moved or non-existing targets are sometimes called broken, orphaned, dead, or dangling.

- **CloudServerUISupport**

  It is a middleware between the graphical interface of the cloud administrative panel and the underlying cloud technology, this library gives full support for the realization of all the panels for cloud administrative use.

  It is a library that groups and manages all the underlying libraries to form an enterprise type Cloud platform, and creates an all user-friendly graphical interface utilities, basically it is an intermediary library, between those that form the cloud technology and the graphical interface, to make it easy for those who develop the front end to use cloud technology.
  In practice, although the graphical interface could directly use the underlying libraries, it was decided to create a very simple intermediary library that facilitates its use, normally this part of the code could be merged into the graphical interface but it was decided to unbundle it so that if you were to create multiple graphical interfaces with different technologies (Blazor, Windows Forms, MAUI, Xamarin, Uno Platform, Flutnet, etc..), this would result in a reusable and identical part under the hood of each graphical interface.
  The idea is to keep in the front end project with the graphic interface only the part purely useful for displaying the panels with which the user can interact, so as to have a truly minimalist and easy to understand project for the front end developer, and absolve it from managing the underlying functions that make up the complex cloud technology.
  The underlying part of this library consists of these libraries:

  - BackupLibrary

  - CloudServer Library

  - DaraReundancy Library

- MessengerStorage
- ProxyApiSupport
- RouterServer
- UISupportGeneric (is an artificial intelligence library for the automatic creation of the front end)

To see specifically what the underlying libraries do, look at their documentation.
In practice, new libraries can be added to this middleware library, in order to add new features to the cloud platform or software.

The middleware creates the functions for the creation and management of the following panels:

- Backup management panel.
- Machine boot management panel.
- Dashboard for cloud management and support.
- Panel for managing and monitoring the connection to the router.
- Panel for diagnostic functions and easy identification of problems.
- Panel for customization in the graphical interface.
- Panel for managing cloud instances on the fly and creating them
- License management panel
- Panel for the main settings
- Dashboard for cloud-based messaging software support
- Panel to couple two servers and thus have remote data redundancy
- Panel to instantiate and manage an API proxy
- Panel for managing local data redundancy within the infrastructure
- Pallello for viewing diagnostic reports
- Router management panel
- Panel for displaying system information

    - Keyboard panel
    - Utilities panel

  Each panel can be easily enriched with features giving scalability to the project, and an artificial intelligence system transforms these parts of the library into panels that are truly part of the user interface.

- **CloudServerWebUI (project)**

  This application is a graphical interface based on web technology, therefore usable via browser. All technologies used, including the underlying libraries, have been designed to work on both Linux and Windows platforms
  The graphical interface derives its functions from the underlying middleware library (WebServerUISupport), which organizes all the cloud server subsystem to create an enterprise cloud environment that can be easily scaled (expanded) using the graphical interface at the service of the administrator of the cloud infrastructure.
  Graphical interface in web technology can be easily modified by front developers using html and css code.
  This user interface has been focused to administer completely stand alone server machines, i.e. that start in Kiosk mode and this interface exposes the only panels to which the administrator has the right to access, in order to give greater security to the system by not giving full access to the operating system and other functions outside the cloud.
  To adorn the web interface there is also an agile terminal that appears inside the browser, with which commands can be given from the keyboard: Everything that can be done through the panels of the graphical interface can also be done from the terminal with commands dedicated to cloud technology.
  The naiffe aspect of this panel is the technology we have developed to create them: the graphical interface is completely automatically generated on the fly (no need for a front end developer!), via our artificial intelligence system which analyzes the assembly of the backend in real time, memory allocations, etc., decompiles it for the interested party and builds a structure useful for the automatic creation of the front end side graphical interface.

  This is a short description of some panels showing the UI, it is a partial description not complete, which includes the main features:

    1. Connections
       Monitor active connections with software router. The router acts as a clearing house between the clouds and the clients, and also provides connection to the proxy which acts as a gateway with the APIs. All encrypted communications are forwarded to the router and then redirected to the destination application or device.

    2. Diagnostics
       Set of tests and utilities for diagnostics and troubleshooting

    3. GUI
       Graphic settings of the interface with which the user interacts with the application

    4. Instances

Information panel on currently unsatisfied clouds and a function on this machine
One machine can instantiate multiple clouds, or multiple clouds can be instantiated on different machines. The cloud cannot be reached directly by the client but the communication takes place through a router, this allows you not to expose the cloud to the internet in order to have more security (only one router port must be exposed to the internet, which can be even in a remote machine, and the Cloud does not need a static IP). If the client is started correctly, it will automatically remain connected to the router via the entry point specified in the settings. Since the communication between client and cloud is encrypted, the router cannot know the transmitted data.
At the first start, each client generates key pairs for asymmetric cryptography, the QR code represents the access point and the public encryption key so that communication takes place confidentially from the beginning. Each data packet is encrypted and signed with the private key of those who generate it (client or cloud server).
The cloud supports two client communication protocols, the native one based on tpc socket with cryptography, and a proprietary REST API protocol to which we have added several point-to-point encryption methods to increase its security compared to the standard protocol.
The encrypted REST API protocol uses RSA encryption for the exchange of encryption keys, so it is necessary to use the QR code including the RSA code to connect to the Cloud through this type of client.
We have created a web client in javaScript based on our protocol that serves as an example for software producers who want to create their own compatible client. Instead, the client based on the encrypted tcp socket native protocol, is provided to developers in the form of an open source library.
Clients using the encrypted REST API protocol cannot connect directly to the cloud, but must connect via a proxy, which is connected to the cloud via the router, so that it is possible to communicate to the cloud by exposing only the proxy to the internet and leaving the Cloud in an isolated place in order to increase security and still make the Cloud reachable even in those cases where it does not have a static IP

5. Licenses
   Utility for creating OEM licenses
   The cloud to connect to the router needs a license.This tool allows you to create a license that will automatically be implemented in the router.
   The license is an encryption key that is used to digitally sign the cloud connection request to the router.

6. Main settings
   Cloud system setting information

7. Messenger
   Encrypted messaging software support
   The router used for the cloud can also function as a control unit for encrypted messaging devices (they use the same protocol as the cloud). It is therefore possible to create a completely internal corporate messaging circuit whose data is encrypted and visible only to the sender and recipient. The messaging device/software also allows documents to be sent for sharing or saving them in the cloud: The cloud, in the communication device, is a contact in the address book to which documents or messages can be sent.

8. Pairing
   Pairing allows you to pair 2 devices in remote geographical locations, one acts as master and the other acts as slave and keep the synchronized version of all data in real time.
   Pairing is the best methodology to secure data against the theft of hardware devices, seizure of machines or destruction due to accidental events such as floods, earthquakes, wars, destruction due to a meteorite fall on the place where the cloud is kept , etc..
   A company can secure the data by coupling two machines, at two offices located in different geographical locations, this technique is the only one that gives the maximum guarantee against the loss of data due to breakages or accidental events.

9. Proxy
   Panel related to the proxy for communication with the API, installed on this machine
   Proxy is software that allows clients that support the encrypted REST API protocol to communicate with the clouds connected to it, without the clouds being directly exposed to the internet.
   The proxy communicates with the clouds in encrypted tcp socket protocol, through the router, while the clients communicate with the proxy in an encrypted way through GET and POST methods, typical of REST API technology.
   The addition of encryption to the REST API protocol is our invention to increase its security and privacy, in fact the proxy is not aware of the data it transmits and there is no system to capture this data, because you create a direct encryption tunnel between clients and cloud.
   In the QR code generated by the Cloud there is the public encryption key, the client uses it to connect and to scrape a symmetric encryption key that will remain secret, and will serve to encrypt the communication.
   The public cloud encryption key will never leave your machine, our encryption and communication protocols are open source and therefore can be inspected for a security check.
   To connect to the cloud, clients must prove that they know the PIN: the PIN is never sent for security reasons, the server creates a cryptographic puzzle and if the client solves it then it will have proven that it knows the PIN without ever having transmitted it.
   This proxy is more secure than a VPN, while the VPN creates an encrypted transmission up to it and then the communication continues in the clear, this proxy only acts as a pass of encrypted data from the source to the destination.
   The only data that the proxy can see are the User Agent and the IP of the connected clients, this data is transmitted to the Cloud and recorded in the event log to keep track of the activity and access attempts.
   To avoid brute force attacks, you can make a maximum of 3 login attempts every 5 seconds.
   Each proxy (there can be more than one) has a cryptographic key pair, the cloud needs to know the public key in order to connect to it through the router. The cloud doesn't need to know the IP of the proxy to connect to it, the public key is enough, but it must be properly connected to the router. Instead the client needs to know the IP of the proxy in order to pass data through it. Proxies and routers can reside on the same machine. The location of the proxy is also indicated in the QR code generated by the client, so the client will configure itself automatically by scanning the code.

10. Redundancy
    Keep a redundant copy of the unit synchronized in real time

11. Report
    Program event reports, useful for diagnostics

12. Router
    Panel related to the router that manages the encrypted communication between the clouds and their respective clients
    The cloud is not connected directly with the client, but through a dispatcher, this dispatcher is the router, which acts as a node for everyone (for all cloud instances, and clients). Communication is encrypted and the router must be publicly exposed to the internet for clients to connect.
    The router can be started on the same machine that instantiates the clouds, or you can keep the clouds in machines protected by firewalls and not reachable from the internet, and connect the clouds to a router exposed to the internet (with public ip). The best performance is to have Cloud and Router on the same machine. The communication protocol of the router also allows the sending between the network of messages of various types, encrypted from point to point, and also group messages, containing text, audio and images, with a reading notification system. Privacy: The router does not save any type of data, has no database and does not use writing media, the communication is encrypted and the router is not aware of what is being transmitted.
    Each machine connected to the router (client, server or proxy) has an ID that corresponds to the public encryption key. To increase the level of anonymity and privacy, the public key of the connected machines is never used in the communication protocol with the router, but an ID derived from the hash of the public keys, since the hash algorithm is not reversible, from this the identifiers represented by public keys cannot be obtained.
    The communication protocol provides for the generation of a new symmetrical encryption key for each packet sent, and the digital signature of the packet using the private key of the asymmetric encryption. We have made the communication protocol public so that anyone can examine it and evaluate its safety

13. System info
    Information about your system and hardware

- **CloudUIWeb**

  (Minimalist web interface for single instance private cloud server, home consumer)

  It is a minimalistic program that launches a single instance of a cloud server using the powerful CloudServer underlying library and displays its operation via textual information.
  This software allows the creation of a private cloud, very simple to use, without administrative control tools with a simple panel with information on the status and use of the cloud, for diagnosing problems and knowing the status of the connection.
  This type of cloud server, being designed for home consumer customers, unlike the corporate one, does not require a static IP, however to function it requires a proxy and a router with static IPs that act as a bridge, and multiple IP-free "home consumer" clouds. The proxy and the router can also be on the same machine, in which case the fixed IP can be the same for both.
  The idea is to make the cloud that does not have a static IP that identifies it always reachable, via a proxy which is instead placed on a static IP that is always reachable, for communication

with it via encrypted APIs, or via a software router with static IP using socket synchronization protocol.
The Cloud has its own cryptographic digital identity, made up of a pair of keys (public and private) which is used both as a unique identification ID and to encrypt the data that is transmitted, and in this way everything that passes through the proxy does not can be captured and seen in the clear.
The connection between the cloud and the proxy does not take place directly but via a router which acts as a hub for all the clouds, and then routes the communication towards the proxy, which is also identifiable via digital identity given by the pair of cryptographic keys. Therefore, in order for the cloud to connect to the proxy, the public key of the proxy (which acts as an ID) must be set in the application settings file, in this way the router will route the packets destined for the proxy to the correct machine.
In a network with heavy traffic it is also possible to connect several proxies to the router, with different cryptographic keys, so that each proxy has a different ID and then set the clouds with the public key of the proxy that you want to use in order to divide the traffic between all clouds on different proxies.
The graphical interface of the cloud shows the following salient data: A QR code that contains the location of the router and the cloud public key, this QR code allows socket-based clients to connect to the cloud and synchronize.
A second QR code that contains in addition to the data of the first one, also a public RSA cryptographic key, which is used to connect clients based on our encrypted rest API implementation. And finally an encrypted QR code which is our latest cryptographic connection implementation that allows both encrypted rest API clients and socket clients to connect by exchanging some packets which allow them to obtain the clear QR code without it passing through the data communication . This type of QR code is more compact than the others because the data in the QR are not complete but must be fished through the exchange of some packets, the more compact QR code allows easier scanning by the client device that wants to connect.
Other salient data visible from the graphical interface are: The status of the connection to the router, the status of synchronization and the number of files sent, the amount of bytes sent, transfers in progress both in upload and download, any errors, a list of clients that have access to the machine with some brief information about them.
Since this is a user interface that instantiates a cloud dedicated to the home consumer range, it is really very minimalistic, the concept on which we based ourselves is that of creating a machine that does what it has to do without any type of external intervention, clearly then starting from source to this graphical interface if the need arises, more information can be added, or allow an administrator to interact with it as for example occurs with the graphical interface of the enterprise and scalable version of the cloud.

- **ProxyAPISupport (Proxy API Support library)**
  It is a library to add proxy functions to a software for communication between cloud client and server via API.
  What is the proxy for?
  The proxy allows you to have an access point with a static IP that is always available to clients, and to be able to keep the servers (the clouds) in a private intranet without a static IP and without exposing them directly to the internet, basically the clients that use the proxy they do not communicate directly with the cloud. The proxy also allows users of cheap private clouds to be able to install the cloud on a network without a static IP, as the clients will still have access to the cloud via the proxy which will always be available at a fixed and pre-established address.

What does the cloud identify the proxy: In the QR code that allows the client to connect, there are the public cryptographic keys and the position of the proxy, so a scan of the QR code is enough to let the client understand how to establish the connection.
The APIs are our exclusive and innovative implementation of the REST API protocol, to which we have added encryption to make them secure in the context of cyber security.
The traditional standard APIs do not have a robust information security system, at most they are protected by an https protocol which only covers the passage of information from client to proxy, but then on the proxy machine the information is unencrypted, and can be captured by by personnel assigned to the machine or by malicious software installed. Our cryptographic implementation protects the transit of packets in a highly secure way from client to cloud storage, and only the cloud software is the only one that sees clearly everything that is transmitted to it at the level of data packets.
This library allows you to create a proxy that can be queried by a set of commands by means of POST and GET methods, typical of the REAT api protocol, with the addition of a native encryption layer on the packet, this innovative non-standard security implementation , is our addition that works like this: In the QR code that the client uses to connect, there is either a cryptographic public key or encrypted information on how to find the public key, using the asymmetric encryption public key, is exchanged between clients and servers a symmetric encryption key which will be used to secure the communication. The proxy retransmits everything it receives from the client to the router via a TCP socket communication channel and the router retransmits them to the cloud which is identified via a digital identity created with a pair of cryptographic keys (in our infrastructure, every machine, server, cloud , proxy, messaging device, generates a cryptographic key pair and a pass phrase using Bitcoin technology, this cryptographic key pair corresponds to a digital identity and is used to uniquely identify the device, to secure packets by signing digital, and to encrypt data).
The system bases its security on the trustless concept (the most modern concept in the field of information security), that is, even programmers or those who manage the network, who wanted to see the messages that pass between the client and the cloud, cannot do so as technically backdoors are not feasible.
The function of the proxy is essentially to resend the packets it receives to the client or server, and some diagnostic functions on the problems encountered which are notified to the client by means of 4xx or 5xx response errors which, based on the number, describe the problem encountered so that support personnel and developers can figure out what's wrong and how to fix it.
When the proxy is initialized, it has a pair of cryptographic keys (public and private), which also create a digital identity of the machine, and this digital identity allows the router to identify the proxy and communicate with it.
Through this library, the proxy also generates a textual description showing the connected clients, the number of packets sent and received, the machine ID, the public encryption key, the entry point of the router, the connection status, the 'host (the internet location of the proxy), and other useful information.

- **SecureStorage Library**

The necessity and desire to secure personal information is one thing that everyone shares around the world in the recent times, ranging from businesses to governments to military structures. Data security is critical whether it is being stored, sent, or delivered. Data breaches, hacking, and lost or stolen devices can have catastrophic financial and reputational costs. The need for a Library to protect data generated and handled by applications arose from a desire to protect not only public structures, but also individual citizens, who are even more at risk if their freedom of expression, gender, religion, and any data relating to their person and loved ones is not protected.

Any application that does not secure the data it generates and manages carries the risk of revealing sensitive information that can be used to profile users, scammers to invent scams, and hackers to carry out their plans to pirated programs. The information created by the applications can easily be gathered and marketed on the dark web.

SecureStorage is a library that provides effective encryption to the apps that use it, making the data generated by it inaccessible and inviolable.

Any application creates a large quantity of data; some of it serves just as a warning, while others are essential to the application's operation and users, and some of it, if interfered with, can allow the application and its content to be hacked.

To protect yourself from malicious hackers and organizational data breaches, encrypt all data generated by the application and prevent it from being saved in a way that may be read externally. In the case that unwanted access is permitted to a computer network or storage device, other apps on the same device, or system applications designed with fraudulent purpose by the device's maker, encryption provides an extra level of protection. The hacker will be unable to access the application data encrypted through SecureStorage.

What are the functions of the library: The library offers 2 types of functions, saving objects, and saving values. Objects are nothing more than instances of class, which can have different properties or sub-objects that by means of this library will be saved and frozen in encrypted form to then be able to be fished out again. This feature allows many applications to save the internal working status safely to be recovered after reboot. A practical example of using this library is using it to save contacts, items for sale, announcements, encryption keys, personal and sensitive data, and anything else that in computer science can be represented with a class and you want to make it secure and inaccessible. Internal saving takes place first by means of serialization of the objects, followed by the addition of encryption and then finally with the secure saving of the data on the internal archiving system.

The second type of saving allows you to save the value of text, numeric, bolean, and DateTime variables in an encrypted and permanent way. These variables to which a key is assigned can also be recalled after restarting the application.

Securing takes place via the encryption which can be strengthened by passing the hardware saving functions of keys and values, during the initialization of the library for use.

It often happens that several applications, although they work in a very secure way, can be violated by modifying the data they generate and manage, this is not possible if the developers use this library.

What is encryption?
Simply said, encryption transforms data entered into a digital device into gibberish-like pieces. The encrypted data becomes more unreadable and indecipherable as the encryption technique becomes more complex. Decryption, on the other hand, restores the encrypted data to its

original state, making it readable again. Unencrypted data is referred to as normal data, and encrypted data is referred to as encrypted data.</para>
Software vs Hardware encryption
Software encryption encrypts data on a logical disk using a number of software packages. A unique key is created and saved in the computer's memory when a drive is encrypted for the first time. A user passcode is used to encrypt the key. When a user enters the passcode, the key is unlocked, allowing access to the drive's unencrypted data. The drive also stores a copy of the key. When data is written to the drive, it is encrypted using the key before it is physically committed to the disk; software encryption works as an intermediate between application read / write data on the device. Before being given to the software, data read from the drive is decrypted using the same key.

Hardware - level encryption is possible on some devices: Hardware - based encryption is used in Self - Encrypting Drives(SEDs), which takes a more comprehensive approach to encrypting user data. SEDs include an AES encryption chip that encrypts data before it is written to NAND media and decrypts it before it is read. Between the operating system loaded on the drive and the system BIOS is where hardware encryption takes place. An encryption key is generated and stored on NAND flash memory when the drive is encrypted for the first time. A custom BIOS is loaded when the system is first booted, prompting for a user password. The contents of the drive are decrypted and access to the operating system and user data is provided once the pass is entered.

Self-encrypting drives also encrypt and decrypt data on the fly, with the built-in cryptographic chip encrypting and decrypting data before it is written to NAND flash memory. Because the encryption procedure does not use the host CPU, the performance penalty associated with software encryption is reduced. The encryption key is typically placed in the SSD's built-in memory at system startup, which complicates recovery and makes it less vulnerable to low-level attacks. This hardware-based encryption solution provides strong data security in the event that the device is lost, cannot be disabled, and has no performance impact. However, it is a type of low-level encryption that is completely transparent to the device that uses these storage units, as well as to all software programs that run on the device. As a result, this type of encryption does not protect the data of individual applications and users from other resident programs that can see all of the data stored in clear text.

SecureStorage provides an additional layer of security for individuals who utilize primary hardware encrypted devices, rendering the data unreadable outside of the single program that created and is using it.

The Advanced Encryption Standard (AES) is a cryptographic technique that is based on the Rijndael family of algorithms. It is now one of the most widely used encryption and decryption techniques. Vincent Rijmjen and Joan Daemen created the Rijndael algorithm, which is a block cipher. It's a symmetric-key algorithm, which means it encrypts and decrypts data with the same key. As a consequence of the NIST Advanced Encryption Standard competition, the Rijndael algorithm was chosen as an Advanced Encryption Standard and the successor to the Data Encryption Standard (DES). The competition was held in order to produce a new cryptographic standard as a replacement for the obsolete DES. Because to the modernization of computer technologies, the Data Encryption Standard's key length (56 bits) was insecure at the time. The Rijndael family of functions is represented by three algorithms in the AES standard. They have varying key lengths of 128, 192, and 256 bits, but they all use the same 128-bit block length.

More variations of encryption algorithms, cyphers, and other cryptographic functions are included in the Rijndael family of hashing functions than in AES. The Advanced Encryption Standard was designed to work equally well in software and hardware implementations. With the deployment of the substitution–permutation network design, it was possible. This network design is similar to the Feistel network, which was utilized in DES, but it is faster to compute on both hardware and software, which was critical given DES's software implementation inefficiency.

Our cryptography is the same as that used in Bitcoin, which has been put to the test by hackers all around the world without ever being broken: Breaking this form of cryptography would give you access to coins stored in wallets, which no one has ever done before.

The Advanced Encryption Algorithm (AES256) is an AES algorithm with a key length of 256 bits.The computational difficulty of the decryption is affected by the length of the AES version. The key recovery for AES 256-encrypted data requires more computational power than the 128 and 192-bit variants. The biclique attack, for example, can decrypt AES128 with a computational complexity of 2126. The computational complexity of biclique attacks on AES 192 and AES 256 are 2189.9 and 2254.3, respectively.However, for every key length, real execution of the attacks on the AES-protected data is currently impractical. All of the AES attacks are hypothetical. Every known AES attack would take millions of years to complete, regardless of the algorithm's key length.

- **Anonymous Messenger**

  Encrypted communication software with trustless technology (military security level)

  We exacerbated the level of security by eliminating the backend: There is no backend that manages accounts, since there is no backend, ackers cannot enter a server that does not exist! Accounts are client-side only, just like for Bitcoin the account is a private key generated by a passphrase which is kept securely on the client side (the project derives from the Bitcoin project library and inherits its fundamental concepts).

  Our mission is to exacerbate the concept of security in messaging and create something conceptually new and innovative from a technical point of view. Top-level encrypted communication (there is no backend , there is no server-side contact list, there is no server but a simple router, the theory is that if the server does not exist then the server cannot be hacked, the communication is anonymous, the IDs are derived from a hash of the public keys, therefore in no case it is possible to trace who originates the messages, the encryption key is changed for each single message, and a system of digital signatures guarantees the origin of the messages and prevents attacks "men in de middle"). We use different concepts introduced with Bitcoin technology and the library itself: there are no accounts, the account is simply a pair of public and private keys, groups are also supported, the group ID is derived from a hash computed through the public keys of the members, since the hash process is irreversible, the level of anonymity is maximum). The publication of the source wants to demonstrate the genuineness of the concepts we have adopted! Thanks for your attention!

  A peculiarity of this software, being the low-level messaging libraries the same as our private cloud platform, companies and individuals can use the cloud software router to pass messages between users, creating an internal private messaging circuit, in the which nothing between hosts and external datacenters, as for example instead happens with telegram and whatapp. In any case, everything has been engineered so as not to make it feasible to implement backdoors by us who manage the data communication infrastructure.

It is a project composed of several layers to create a complete messaging software, which can represent the maximum in security and privacy, for this purpose see the technical documentation and the description of the underlying libraries.
The project includes several software layers which technically consist of multi-platform libraries (Linux, Android, iOS, and Windows).
In the lowest state we have the CommunicationClannel Library, a socket-type communication protocol that has sophisticated mechanisms that recover communication even in the case of mobile users where the phone can unexpectedly change the IP and cell to which it is connected, and in the libraries there's also a sophisticated packet spooler and everything needed to re-establish the precarious connection. Technically the CommunicationClannel can be replaced with other compatible ones that instead of the internet connection use the GSM modem network, or serial transmission or other means of communication, in order to use the messaging software with different hardware means of data communication.
At a slightly higher level we have the EncryptedMessaging, the low-level binary encrypted communication library that deals with the encrypted sending of packets, the management of contacts and everything needed to create a complete and sophisticated security-oriented messaging software information technology, the only thing missing is the graphical interface that will have to be created by the designers in order to customize the user experience according to one's needs. The idea behind this library was to create an encrypted binary messaging platform useful for any need and functioning in any circumstance and on any data transmission medium by replacing the CommunicationClannel which deals with the physical transmission of the packets. The EncryptedMessaging library is so universal that in addition to being used for encrypted communication software, we have also used it as a cloud underlying system for synchronizing data between clients and servers.
Finally, at the top level we have the multi-platform messaging interface (Android, Linux, iOS, Windows), which in fact is only a graphical interface that allows the user to interact with the software, and developers to create a messaging completely customized in terms of graphics and ergonomics.
Messaging software also has crypto wallet functions, and other experimental stuff.

- **Messenger Storage**

  The library for Messenger Cloud (GUI)

  This here is Anonymous Messenger helper software, it adopts trustless security. It offers an encrypted cloud storage where devices encrypt their contact list and other data and send it for safekeeping. This allows the recovery of the contacts, when the messenger account with the passprase is recovered. The device encrypts the data before sending it and therefore the cloud has no way of being able to see it unencrypted (trustless).
  Both this project and the messaging software are open source and inspectable, no aspect of privacy and security has been overlooked, this work reaches military security levels.

  You may also be interested in:

  - Anonymous messenger, messaging software with military-grade security

This project has three open source dependencies for security and functionality. These dependencies are implemented here in the form of Nuget packages, and here are the sources on GitHub (you can replace the nuget packages with the source projects if you want):

- Secure storage: it is a powerful data safe, the cryptographic keys and data that would allow the software to be attacked are kept with this tool.
-
- Encrypted messaging: it is a powerful low-level cryptographic protocol, of the Trustless type, which manages communication, groups and contacts (this software will never access your address book, this library is the heart of the application).
-
- Communication channel: is the low-level socket communication protocol underlying encrypted communication.