



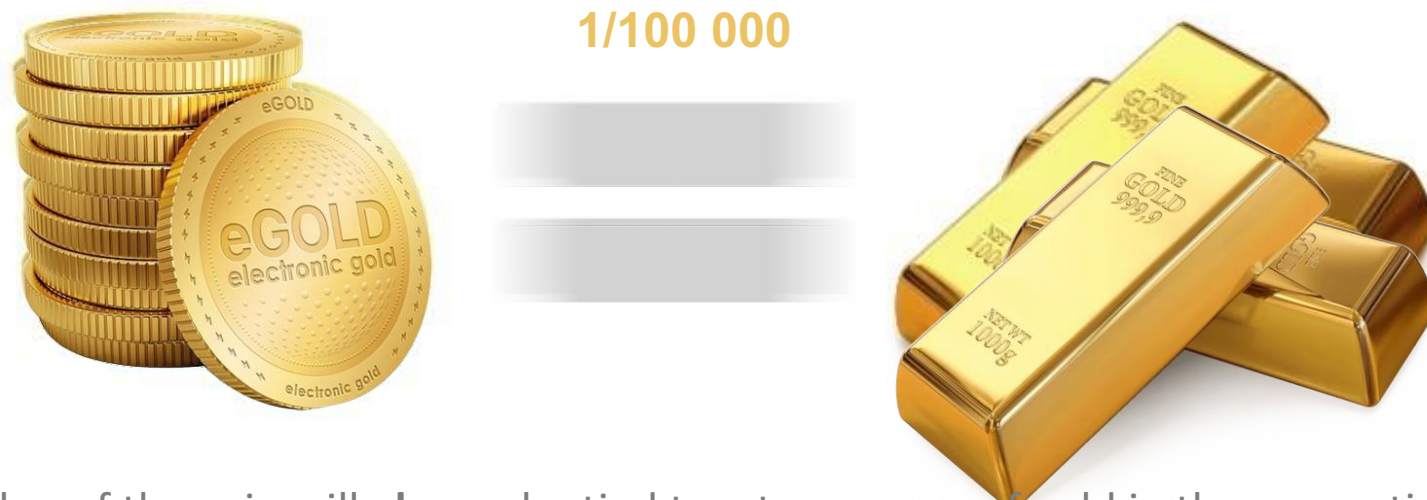
CRYPTOCURRENCY  
OF THE FUTURE



<b>Description of eGOLD coin.....</b>	<b>3</b>
<b>Comparison of payment systems .....</b>	<b>4</b>
<b>Bandwidth.....</b>	<b>6</b>
<b>Unique advantages of eGOLD.....</b>	<b>7</b>
<b>Roadmap.....</b>	<b>12</b>
<b>Why eGOLD?.....</b>	<b>14</b>
<b>Frequently Asked Questions.....</b>	<b>23</b>
<b>Working with the node API.....</b>	<b>36</b>
01. Request to the node without parameters.....	36
02. Request [type=wallet] - data on the wallet.....	36
03. Request [type=nodas] - list of available nodes.....	38
04. Request [type=history] - transactions history.....	38
05. Request [type=referrals] - transactions history for referrals.....	40
06. Request [type=referralwallets] - output of a list of referrals' wallets for this wallet.....	41
07. Request [type=referralresults] - output of the total number and volume of coins by referrals level.....	42
08. Request [type=height] - obtaining the height of the wallet and the ability to send a transaction Used for an auxiliary request before the transaction.....	43
09. Request [type=send] - carrying out of a transaction where can be the change of public and private key, the change of a password for managing contacts and E-mail notifications, the registration of a new wallet.....	43
10. Request [type=synch] - node's synchronization.....	44
11. Request [type=synchwallets] - addressing to a synchronizable node.....	44
12. Request [type=contacts] - displays the wallet's contacts on the node from which it makes transactions.....	45
13. Request [type=email] - E-mail notifications for wallet transactions on the node from which it makes transactions.....	46
14. Request [type=balanceall] - the total balance of all wallets, excluding accrued interest.....	46
15. Request [type=walletscount] - the total number of wallets.....	47
16. Request [version] - displays the current version of the node of the file egold.php.....	47
17. Example of making a transaction in PHP.....	47
18. Example of creating a wallet in PHP.....	48
19. Example of changing the private key in PHP.....	50
20. Example of accepting funds in PHP.....	52


## Description of eGold coin

**eGOLD** (electronic gold) is an electronic cryptocurrency that received its name for its intended purpose, as well as E-mail (electronic mail). E-mail messages have firmly entered our lives and replaced the usual letters because of their simplicity, ease of use and instant delivery speed without any additional costs. E-mail messages, which is quite important, are friendly to the environment and do not require the use of natural resources. eGOLD is also positioned as an eco-friendly alternative to physical gold and paper money with significant advantages over them that are similar to the advantages of E-mail letters over paper letters.









The value of the coin will **always** be tied to a troy ounce of gold in the proportion of 1/100 000. For example, at the current rate on June 11, 2020, **100** coins of **eGOLD** = **\$1.72** or **118.02 RUB**.

# Comparison of payment systems

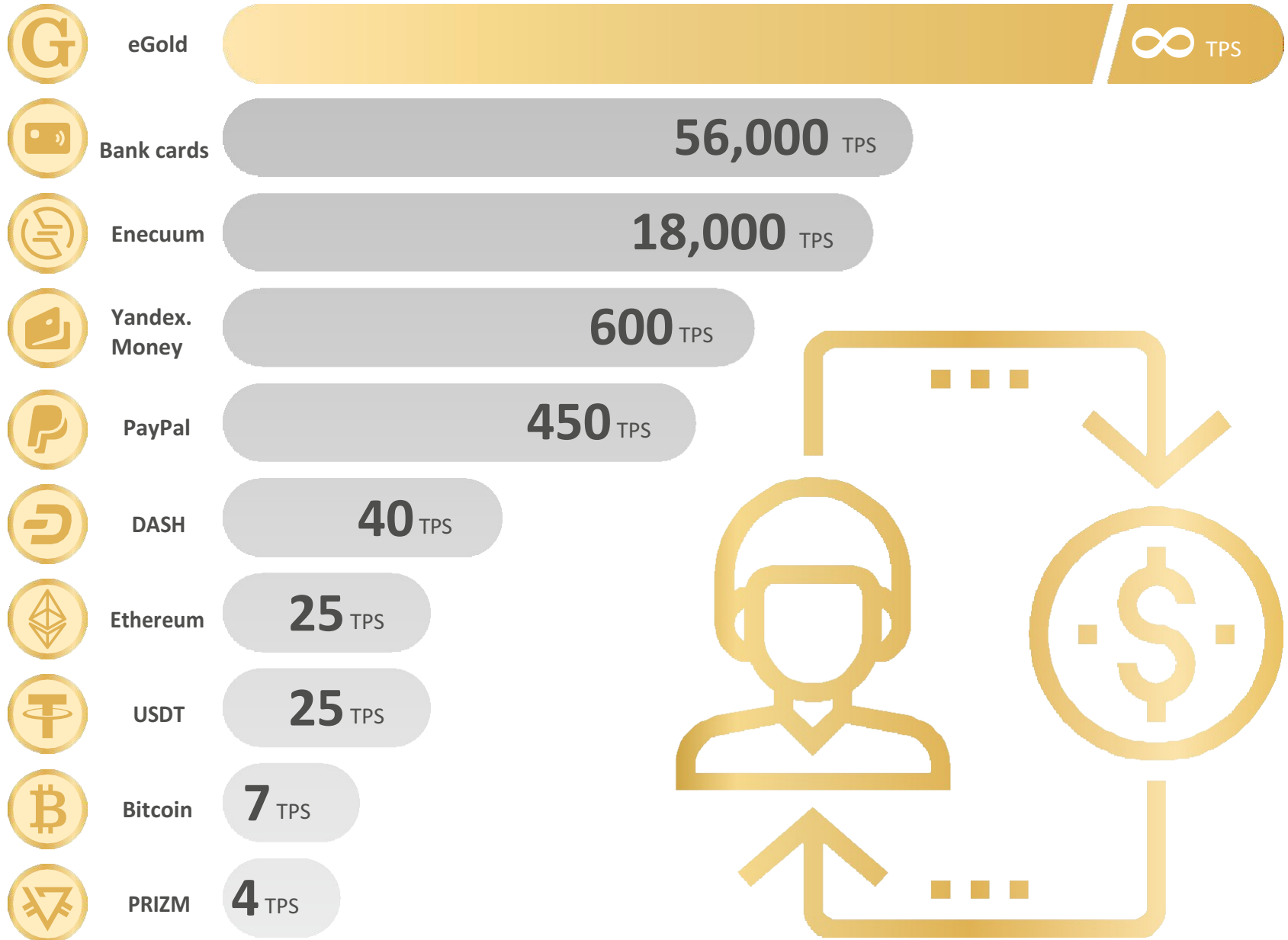
	Payment system	Transfer commission	Transaction speed	Maximum number of transactions per second	Volume of the database	Lack of freezing	Resistance to hacking by quantum computer	Monthly income on the balance	Remuneration for promoting or referral program	Stability of rate of the payment system funds	Monthly average change of the funds price over the last 6 months against the dollar
	eGold	\$0.035	less than 4 seconds	unlimited thanks to parallel transactions	the volume is customizable and unchangeable thanks to the horizon of the transaction history	+	+	from 4% to 5%	from 0.25% to 1.25% monthly from the balance of wallets created personally and by referrals of up to level 3	+	+ 4.94%
	Bitcoin	from \$0.1 to \$10	10 minutes	from 2 to 7	increases by 130 MB a day and is already more than 330 GB	+	-	0	0	-	+ 4.52%
	Ethereum	from \$0.1 to \$3	13 seconds	from 15 to 25	increases by 240 MB per day and is already more than 340 GB	+	-	0	0	-	+ 13.96%
	USDT	from \$0 to \$2	4 seconds	from 2 to 25	increases as Ethereum and Bitcoin depending on the algorithm	+	-	0	0	+	0
	PRIZM	from \$0.0008 to \$0.17	1 minute	4	increases by 30 MB a day and is already more than 27 GB	+	-	from 1.16% to 14.85% now and from 0.2% to 1.31% with the target PARATAX from 97% to 98% depending on the personal balance and referrals' balance up to level 88 and is credited until the ultimate volume of the coin issue	increase in income from the personal balance depending on the balance of wallets created personally and by referrals of up to level 88	-	- 15.70%
	DASH	\$0.003	about 2 minutes 40 seconds	40	increases by 12 MB a day and is already more than 23 GB	+	-	about 1% - at the same time, the balance should be 1000 coins - it's \$72,000 at current rate	0	-	+ 13.16%

# Comparison of payment systems

	Payment system	Transfer commission	Transaction speed	Maximum number of transactions per second	Volume of the database	Lack of freezing	Resistance to hacking by quantum computer	Monthly income on the balance	Remuneration for promoting or referral program	Stability of rate of the payment system funds	Monthly average change of the funds price over the last 6 months against the dollar
	Enecuum	?	15 seconds	18 000	increases	+	-	about 2.3%	+10% from mining for referrer and referral	-	- 8.52%
	Bank cards	from 0% to 5%	from 1 second to 7 working days	56 000	increases	-	+ / ?	from 0% to 0.5%	0	+	0
	Banks	from 1% to 10%	up to 5 working days	?	increases	-	+ / ?	from 0% to 0.55%	0	+	0
	PayPal	from 0% to 4%	several seconds	450	increases	-	+ / ?	0	0	+	0
	WebMoney	0.8%	several seconds	?	increases	-	+ / ?	0	0	+	0
	Yandex. Money	from 0% to 3%	several seconds	600	increases	-	+ / ?	0	0	+	0



## Bandwidth



## Unique advantages of eGOLD



eGOLD runs on the non-volatile Proof-of-Stake algorithm which checks transactions by confirming with more than 50% balance of the available nodes. That allows to keep a node on a device of almost any power.



The highest security is achieved by using a quantum-resistant encryption FALCON and by receiving by nodes of transactions already signed by the private key on the client's side.





In cryptocurrency, complete anonymity is achieved by using only a convenient 18-digit wallet number. For example: G7355- 87879-8875-80955.



There is a unique opportunity for cryptocurrency – a private key change. This can be done thanks to the unchanged wallet number, in which the new private key is signed with the old private key, like any transaction in accordance with the rules of the most cryptocurrencies. In this case, only the hash of the new public key is stored until the first transaction after the private key change, which further increases the security of using the wallet.

## Unique advantages of eGOLD



  Instead of blockchain, eGOLD uses a multi-level algorithm for parallel transactions of the next post-blockchain generation. Using parallel transactions allows to achieve the unlimited throughput, unlike the standard blockchain where each transaction on the network is recorded in strict sequence. The speed of one transaction reaches 4 seconds.



Creating eGOLD coins is implemented by stacking technology with a reward of 4-5% per month of the total number of coins, which allows to earn coins for the material investments, directly from the wallet balance. Charges to the balance go every second in automatic mode, without the need to turn on the wallet or the node for this. This is done to ensure a fair distribution of coins among all participants of the system. And the initial issue was made only to start the coins distribution. At the same time, in accordance with the algorithm of the cryptocurrency, the developer cannot add coins to the system from outside. Each node holder is a guarantor of compliance with the rules laid down in eGOLD.



The remuneration is charged every second at a compound interest and the entire balance can be withdrawn at any time without any additional transactions.

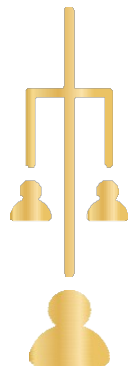


## Unique advantages of eGOLD



For the first time, cryptocurrency has a unique fair 3-level referral system with a constant income from the balance of created wallets, as a reward for the coin's popularization, accrued for any outgoing transaction from a referral:

- ✓ 1% per month from the balance of the 1st level referrals you've invited;
- ✓ 0.5% per month from the balance of the 2nd level referrals from those they will invite;
- ✓ 0.25% per month from the balance of the 3rd level referrals.



If the referral has a node, the reward increases by 25%:

- ✓ 1.25% per month from the balance of the 1st level referrals with the node you have invited;
- ✓ 0.625% per month from the balance of the 2nd level referrals with the node of those they will invite;
- ✓ 0.3125% per month from the balance of the 3rd level referrals with the node.



The minimum constant transaction cost for any amount of transfer is always equal to 2 coins (today it is about 2.4 RUB), and with its node it is equal to 1 coin.

## Unique advantages of eGOLD



The absence of traditional mining is compensated by remuneration to the holders of nodes equal to 1 eGOLD coin from each transaction, and the charges of 1% of the total number of coins on the balance monthly.



There is a service address of the G-1 cryptocurrency itself for the coins deleting.



The innovative technology of horizon of the transaction history allows you to keep unchanged and configure on your own the volume of database of the transactions on the nodes.



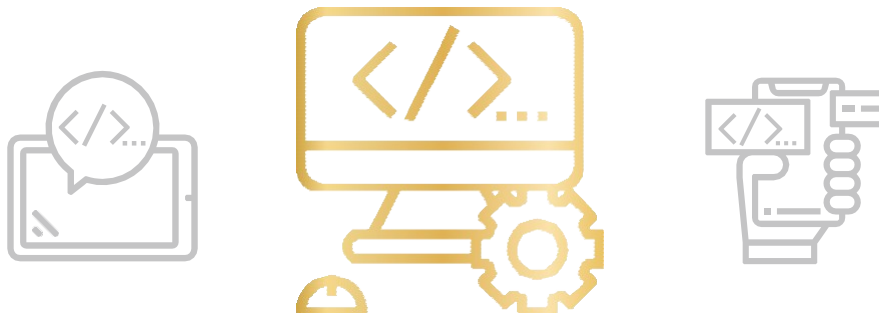
Easy integration into any web projects is possible using a simple API with JSON responses and POST and GET requests, as well as a direct use of PHP and MySQL.



All source code is open.

## Unique advantages of eGOLD

Another distinctive feature of eGOLD is its wallet. This is just one HTML file working on any device. This file contains all the necessary scripts and pictures.



- ✓ The wallet has full functionality, where all transactions, referral receipts and contacts management, as well as E-mail notifications for incoming and outgoing transactions are displayed.
- ✓ There are filters used to find the transaction by date or by digital comment tag that is a pin code.
- ✓ All actions are protected by many algorithms, which eliminates errors and hacking. Even creating a new private key has its own system of protection against accidental user actions.
- ✓ The wallet has the function of a private key change and of a new wallet creation.
- ✓ The wallet management is intuitive even for a person who has never dealt with cryptocurrencies.



- 📍 **1st half of 2018** - collecting information, developing the concept of eGOLD cryptocurrency and selecting applied technologies and technical implementation.
- 📍 **2nd half of 2018** - development of the node and choice of the applied technology for the official wallet, launch of a P2P exchanger of another cryptocurrency and gaining experience for creating a new eGOLD coin exchanger based on this exchanger.
- 📍 **1st quarter of 2019** - completion of the node development and addition of the API for interacting with the node.
- 📍 **2nd quarter of 2019** - development of the design and operating principles of the standalone HTML wallet and making adjustments to the node's API work to adapt to necessary requests and wallet data.
- 📍 **2nd quarter of 2019** - development of a standalone HTML wallet and of the official P2P exchanger eGOLD.pro, testing and debugging of the node.

01

02



- 📍 **1st quarter of 2020** - completion of the development of a standalone HTML wallet and of the official P2P exchanger eGOLD.pro.
- 📍 **2nd quarter of 2020** - the launch of eGOLD cryptocurrency.
- 📍 **3rd quarter of 2020** - creation of a white paper, materials for promotion and the beginning of promotion of the eGOLD coin.
- 📍 **4th quarter of 2020** - the integration examples development and the assistance to simplify the integration of coin into various systems.
- 📍 **1st half of 2021** - distribution of copies of the official P2P exchanger under the franchise program and active promotion of the coin to other exchangers.
- 📍 **2nd half of 2021 and til 2023** - full decentralization of the coin due to its natural distribution among all participants and, if necessary, to achieve decentralization, the destruction of superfluous coins on the first issue wallets by sending the excess to the G-1 service address.
- 📍 **After decentralization, the coin becomes a comprehensive medium of exchanging goods and services.**

03

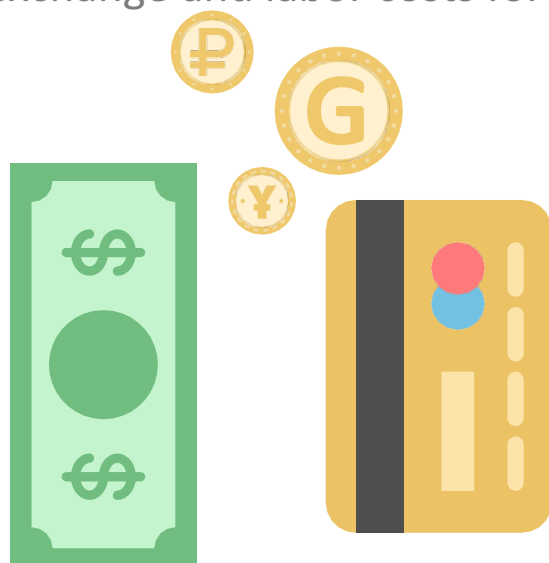
04



## Why eGold?

Any money is a community's belief in its exchangeable value compared to goods or services. And the true price of any goods and services is determined only by the labor invested in them, that's why now the air we breathe has no price, but the air that we can breathe somewhere far away in the mountains or at sea will already have a price, because in order to get there you'll need to make an effort and even pay the carriers. That's the price for the air. It's the same with other goods and services: how much effort and money we spend on it, so much it should cost. Removing costs in the form of unnecessary expenses, we increase the value of money by increasing the number of goods and services purchased with the same amount of funds. Cryptocurrencies do the same, they remove costs. Like plastic bank cards that remove the cost of currency exchange and labor costs for this exchange. For example, it removes the cost of the

bus to get to the bank and the time spent on it. Costs are reduced even when the percentage is spent on servicing of non-cash payments that banks withdraw for the use of plastic cards, and this price is embedded in the product. After all, the best money is convenient money without additional costs with instant secure payment while saving their unchangeable value.

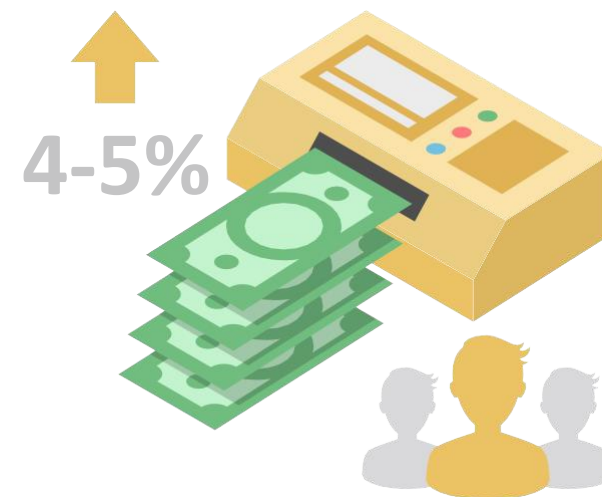




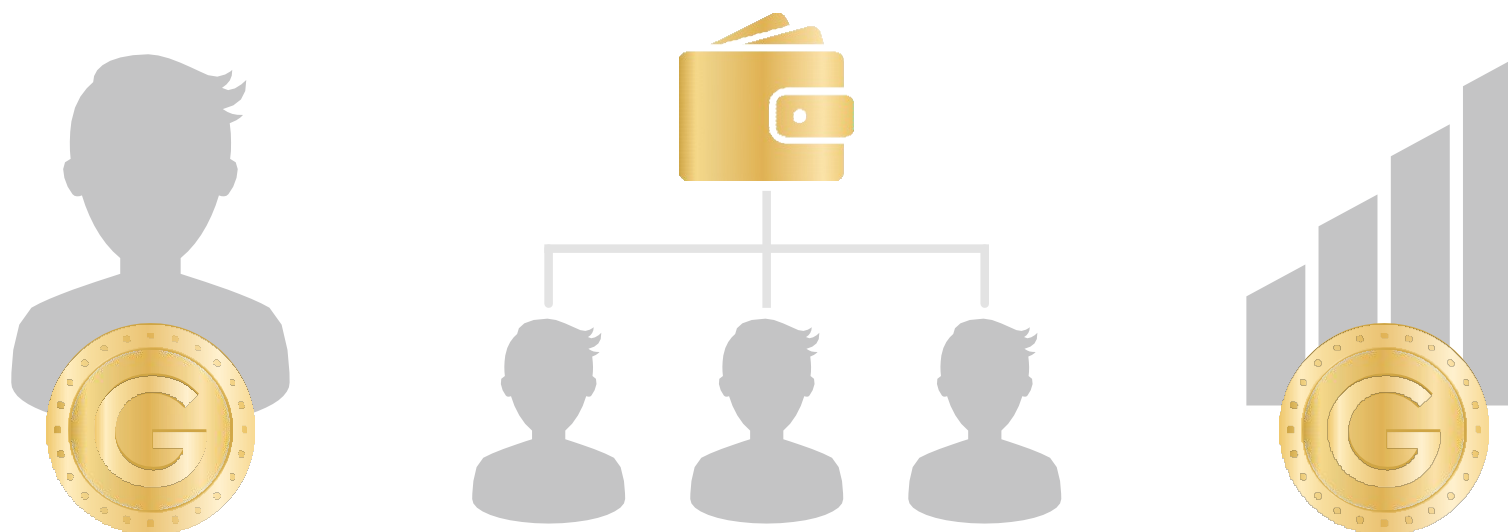
## Why eGold?

With the increase of the coin popularity, it will inevitably lead to the need to increase the coin value or to increase its number in circulation. If the price of a coin increases, it will lose its capacity to pay, because its rate will become floating, which will cause some inconvenience in calculations. Artificial increasing of the volume of coin on the market by its creators will lead to an imbalance and concentration of resources in the hands of a single person that will inevitably undermine the authority of the coin and will contribute to inflation. Therefore, as in some

other PoS cryptocurrencies, eGOLD has a built-in **seigniorage** in the form of 4%-5% profit per month for all users holding coins on their balance or otherwise. In cryptocurrencies this is called earning on **staking**, and such users are called **holders in the blockchain environment**. At the same time, the interest in eGOLD is credited every second and you do not need to hold coins for a long time to get an increase of coins of 4% per month on your balance. Moreover, a bonus of **1%** is credited to the node holders, increasing to 5% the total return from their balance, and there is a referral system for the fair distribution of resources and the encouraging of labor costs for servicing, promoting and using the coin. The initial issue of eGOLD is limited to a small amount relative to its price, in order to give a start to all people who want to evenly distribute coins for their subsequent honest generation and complete decentralization, unlike many cryptocurrencies, where only about 10-30% of all coins are released to the market, and the rest is held by its creators.



## Why eGold?

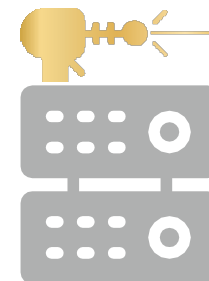


If you can't buy coins and you don't want to hold a node, but you want to help in its development and to earn good money in unlimited quantities at the same time, you should start promoting coins and creating wallets for the other people. In this case, the profit is equal to 25% of seigniorage of those people who were invited (1% per month of the balance of the 1st level referrals without node and 1.25% of the balance of referral with node), and 12.5% from those (0.5% per month of the balance of the 2nd level referrals and 0.625% of the referral with node) who they'll invite, and 6.25% from those (0.25% per month of the balance of the 3rd level referrals and 0.3125% of the referral with node), who will be invited by those who give 12.5%. Spending your energy and money on the eGOLD cryptocurrency, even the money that came in the form of interests, will be more than compensated in time. Popularity, distribution and use of the coin, and with it, the revenue from it, is completely in the hands of the entire community and of each eGOLD coin holder!

## Why eGold?



**eGOLD** is an eco-friendly **PoS** cryptocurrency based on the post-blockchain technology of unlimited throughput of parallel transactions using the fast **quantum-resistant** encryption algorithm **FALCON** and, more importantly, the ability to change encryption keys (public and private key) while preserving the wallet number, while the new public key is closed under **SHA-3** hashing and by its specially designed hashing until the next transaction, making it impossible to determine the private key on the basis of a public key due to its full concealment. In working with a wallet, only a changeable private key and a permanent wallet number are used. The eGOLD cryptocurrency does not have a blockchain, but instead a more advanced multi-level next-generation algorithm for transactions synchronizing similar to the **DAG (directed acyclic graph)** is used, with a significant difference that close parallel synchronization problems - sending a message about a successful transaction to all nodes along the chain, if this transaction has not come before and it has passed validation checks, including the loyalty check and the verification that the signature belongs to the sender's wallet. For the work of the node, it is necessary to have a regular Internet hosting with PHP and MySQL. The time of acceptance of one transaction is about **4 seconds** (transaction processing is about 1 or 2 minutes). Protection against double spending is made using a time delay and is similar to the protection of the **NANOCOIN** cryptocurrency. There is also a check for missed transactions in the nodes and a constant check of the wallets validity for more than 50% of the balance of available nodes. eGOLD has a built-in system for database self-cleaning by introducing a **horizon of the transaction history**, which allows to keep its volume unchanged for the same number of transactions.

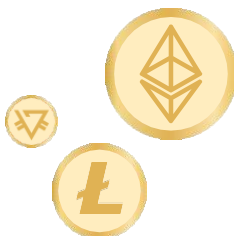


01011010101101010  
1011 **FALCON** 01  
0100 0  
1 **SHA3** **DAG** 1  
0 **PHP** MySQL

## Why eGold?



0%



For example, for the current month (June 2020) **Bitcoin** has on average about 300 thousand transactions per day, and the volume of the blockchain database is constantly increasing and has exceeded 300 gigabytes. In eGOLD, the volume of database is configurable and remains unchanged thanks to the horizon of the transaction history and other self-cleaning methods. There is no mining, but there is a reward for node holders in the form of 1 coin for each node transaction + 1% per month (hereinafter the month is equal to 30 days)

to the node holders from the balance of the wallet attached to the node. All wallets receive 4% of coins per month to the wallet balance at a compound interest when charged once per second - this is 4.08% per month and **61.59%** per year, and the node owners receive an additional +1% when making transactions from their nodes from the current balance per month - this is 5.13% per month and **82.21%** per year at a compound interest, in addition to remuneration from each transaction sent from his node by any wallets. All coin owners receive interest as a **seigniorage** (the income received from the money issue), meaning that all participants are the coins creators. The balance is credited automatically without creating additional transactions. The total initial issue is **1,000,000,000** coins.

Coins can be destroyed by sending them to the G-1 address. G-1 is a professional address that does not have a wallet. There are no other similar service addresses and you can only send coins to wallets addresses. After pre-mining, coins are generated on the user's wallets in accordance with the marketing rules, there is no restriction on the generation of coins. Users can always view the total amount of coins in the node using the request: [http://\[IP of the node\]/egold.php?type=balanceall](http://[IP of the node]/egold.php?type=balanceall), where balanceall in the response is the total amount of coins on all wallets, excluding interest accrual from the moment of last transaction.

5%

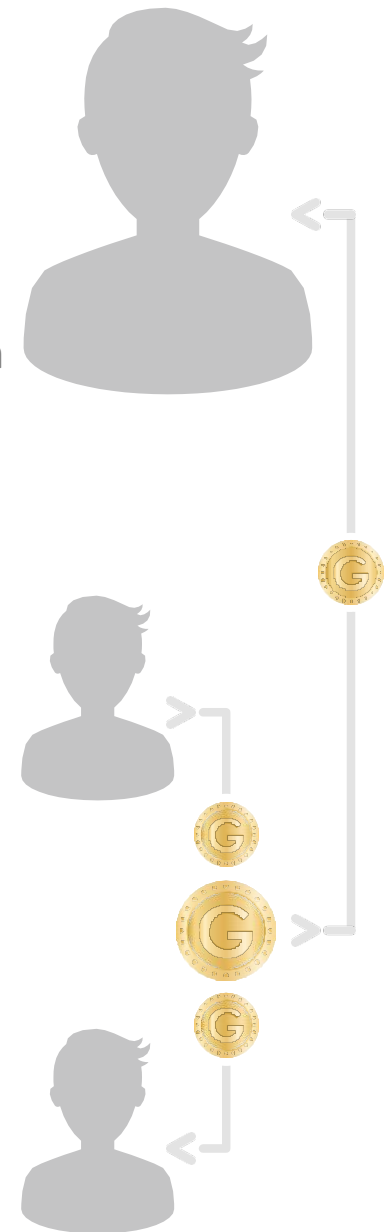




## Why eGold?



In eGOLD, only whole numbers 1,2,3... without hundredths are used as units. The commission for all transactions is 2 coins and does not depend on the transaction amount. At the same time, 1 coin always goes to the owner of the node to which the wallet is connected for transactions, and 1 coin is debited by the system to protect it from attacks using multiple transactions. When a user completes a transaction and adds a password to access the node, he can use the services of the node through which this transaction was made: the contact book (all contacts are encrypted from the node's owner) and E-mail notifications about transactions. The coin has a built-in 3-level automatic referral system. Each time with any transactions of their referrals, the owner of referrals receives a reward from the accrued interest of his referrals, that depends on the balance of their referrals, the time that elapsed between referrals transactions and the level of referrals: 25% (1st level), 12.5% (2nd level) and 6.25% (3rd level). For example: the 1st level referral received for a month 408 coins in the form of interest from 10,000 coins on his wallet (4% per month at a compound interest accrued every second). The person who owns this referral will automatically receive 102 coins, even at zero balance (the referral receives separately 408 coins and the owner of the referral receives separately 102 coins) after any transaction of the referral (transfer of funds from his balance). If the 1st level referral was the 2nd level referral, the owner would get 51 coins.



## Why eGold?



Wallet creation costs 5 coins that already includes 2 coins of the commission of the system, and 3 coins remain on the newly created wallet for the possibility of changing the private key. In this case, only the owner of a previously created wallet can create a wallet. Creating a wallet occurs along with setting a secret phrase for the new wallet, that must be changed later (not necessarily). This is done to promote the coin and protect it from attacks that use multiple transactions.

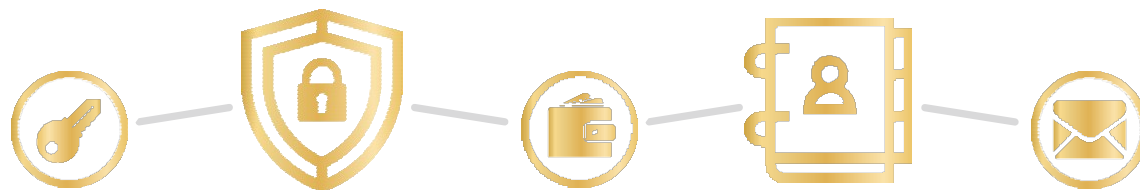
The coin has only a whole part and has no hundredths, like cents for the dollar and kopecks for the ruble. The calculation uses 100 coins instead of 1.00. The price of gold is chosen for many reasons. One of them is the relative stability and growth of the rate in comparison with fiat money (over the past few months the price of gold in US dollars increased almost by 20%).

The stability is due to the limited quantity of gold in nature and its centuries-old value. Not without reason, the most stable money is the money tied to the gold reserve (gold and foreign exchange reserves). Although eGOLD is not backed by gold, all participants are committed to take responsibility for following the set course, just as the global community is committed to value the gold.



## Why eGold?

The node's code is written in PHP (requires a PHP 7.1 version or higher, PHP library - BCMATH, GMP, and CURL) + MySQL (it's recommended to use the latest versions). The official wallet is written in HTML using JavaScript, jQuery, and Ajax and is made as a single HTML, which makes it possible to run it on any device. FALCON encryption using a private key takes place directly in the HTML file, and already signed transaction is sent to the node. This transaction fully protects the secret phrase from theft, by intercepting it at the moment of sending to the node. A pair of private and public key is also generated on the HTML wallet.



For the convenience of users, the node has built-in modules for storing contacts and sending E-mail notifications about transactions, similar to SMS notifications of banks about crediting or debiting funds. The names of contacts in the node are stored in encrypted form. The access to the contacts and the control of E-mail notifications on the node is provided using a password. The password is set and changed during any transaction. For convenience, the wallet has a separate function of setting a password. Contact services and E-mail notifications services work on a specific node and are available to a participant only after making a transaction on this node, which encourages users to use certain nodes, and the node's owner is rewarded with 1 coin for each transaction. For the node to work, the balance of the wallet linked to the node must be at least 100 coins.



The eGOLD node has the ability to self-clean from unused data, which protects it from excessive use of memory and system resources, keeping the same volume of database for the same number of transactions by deleting:

- ✓ history of transactions older than 30 days (this parameter is set by a variable `$history_day` in the settings file `egold_settings.php`)
- ✓ node from the wallet, if there were no transactions on the wallet and this node was not used for more than 30 days
- ✓ accounts to use for sending E-mail notifications and wallet contacts, if the wallet has not used this node for more than 365 days
- ✓ wallets with a balance of less than 10 coins if they have not been used for more than 90 days
- ✓ wallets, if they have not had incoming and outgoing transactions for more than 30 years

The source code of the coin is open and simple, so the coin will be popular with developers. Everyone will be able to make for oneself or for the community the modules that solve some of the problems. And the site owners will be able to embed nodes in any of their web projects and accept eGOLD coins directly. Public and private key generation, signature of transactions (payment) are also possible from the node, which allows to automate services and withdraw funds to the wallets of participants. Node's settings are stored in the file `egold_settings.php`. Requests to nodes are made using GET and POST requests to the file `egold.php`, and the node's responses are presented in the form of JSON.





## 01 **B** How do I log in to my wallet?

**0** To log in to your wallet, you must download and run the file **eGOLD.html**. This file is the official wallet of the eGOLD cryptocurrency. After running the file, you will see the wallet login page. You must enter the 18-digit number of the wallet and the IP of the node you will connect to. You can get a wallet from other people or generate it yourself on the site **eGOLD.pro**. How to do this is described on the wallet page, which can be accessed by clicking on the button **Wallet** located at the bottom of the site. You can also find the IP address of trusted nodes, or you can ask other people for the IP address of the nodes.

## 02 **B** How do I change my private key?

**0** The private key changes when you click on the button of the key with arrows, which is located on the top, to the left of the current wallet number. When you click on the button, a page opens where you need to click on the form and to move the mouse cursor or your finger on the screen until the progress bar reaches 100%. After this, an animated drawing of running dots will appear. The generation of a new private key may take several minutes. When the generation is complete, a new private key will appear thay, you need to save it.



## Frequently Asked Questions

Below is the form where you need to enter the current private key and click the button Send. When the picture with a check mark appears on the screen, the private key is changed. The old private key should be saved for some time, for example for 5 or 10 minutes, because if the transaction is not accepted by the nodes, you'll be able to use the old private key. After changing the private key, it will not be possible to restore the old private key. It's a cryptocurrency. Therefore, check your new private key before transferring large funds on it, to avoid an error you could make when saving it. Changing the wallet's private key costs 3 coins.

03

### **B** How do I transfer funds?

**0** To do this, enter the 18-digit number of the recipient's wallet and the amount in the upper part of the wallet and click the button which is to the right of the amount. You will see the form for entering the private key and the pincode - the comment on the payment with the name of input form MARK. Then you need to click the button to the right and the transfer will be made. At the same time, the screen will darken and a check mark will appear in the center of the screen. This is always the case when accessing the node and if its response is positive. If a red cross appears, it means that the node is not available and you need to repeat the transfer again. If this cross appears frequently or always, you need to use another IP of the node. You can change it in the settings, or you can exit the wallet and enter it when you log in. The transaction fee is always 2 coins.



## 04 **B How do I register a new wallet?**

**0** To do this, click on the "wallet with the plus" button in the menu and do the same as when changing the private key (point 2 of FAQ), but at the end you will see the number of the new wallet with its private key. You should pass this wallet with the key to the new user and tell him to change the private key. A new wallet registration costs 5 coins.

## 05 **B Why is it impossible to log in to a new wallet or to transfer funds to it immediately after its creation?**

**0** The creation of a new wallet is registered in the transaction and the new wallet starts working after the nodes accept the transaction for more than 50% of the balance of node's wallet in accordance with the PoS algorithm. It takes on average about 2 minutes and the transaction line in the official wallet eGOLD.html is no longer orange.

## 06 **B The password is set in the settings of the official wallet eGOLD.html. What is it for?**

**0** This password is used for accessing additional node services. At the moment, every node can store the user's contacts and if the node has a domain, and it is registered in the node settings egold\_settings.php,


the node can send E-mail notifications about incoming and outgoing transactions on the wallet in accordance with the established thresholds for the number of coins in transactions. If you change the password, your contacts will be deleted. The node provides access to its services from the moment of the first transaction sent to it and until the transaction is sent to another node. To start using the services of the node, you need to enter the password, and if this password has not yet been set, the form for its creation will appear. To do this, enter any password of 6 characters or more, write your private key, and click Save. To delete a node's password, erase it in the settings and click Save. To change it, write any characters in the form of the password and enter a new password. The cost of creating or changing the node's password is 3 coins.

07

**B** Why can't I put my wallet on a website for easy use by following a link so that I do not have to store it on my computer?

**O** When working with a wallet from a local computer, the private key remains on the computer and is not transmitted anywhere. Only a signed transaction is sent to the node, making it safe to work with any node. Using a third-party resource, you can't be sure if the wallet was modified to steal the private key or not. Therefore, after downloading the archive with the wallet, check if it is modified using MD5 and comparing the line **MD5** on several trusted resources at once.

## Frequently Asked Questions



It will prevent phishing or wallet spoofing. You can find the information about it and how and where to do it in any web search engine. In short: you send the file and receive the line with MD5 immediately. You can also check MD5 online. But for security reasons, you need to check it on several resources at once.

**08** **B** Does the developer have a possibility to issue additional coins?

**0** Technically, this is impossible, since after launching of the nodes, you will need to completely restart everything and to stop all nodes, to add coins from outside.

**09** **B** How much does eGOLD depend on the developer? If, God forbid, something happens to the developer, what will happen to the project?

**0** If something happens to the developer... the project will live on its own. The source code is open, some of the coins are sold out. The initial issue of the coins will be less, but since other people have nodes, everything will continue to work.

## Frequently Asked Questions



10

**B** Let's assume that all issued coins will be bought out in half a year, or vice versa, there will be a sharp increase in the dollar and many people will want to fix the profit, how is it planned to restrain the rate? After all, the fiat money will not be enough to buy back previously sold coins, or there will be a shortage of coins.

**O** The financial reserve is sufficient, and it is decentralized for the re-purchase of coins, and there are no prerequisites for its negative change, since users themselves buy coins from each other. A 50% repurchase of fiat assets is an additional protection.  
If all the coins are sold, the users will manage the system themselves, and this is a successful scenario, because they'll begin to value the coin even more, but its price won't change. Others will have to wait to buy it for themselves.

11

**B** Is there a possibility to complicate the generation of new coins and is it planned to change or complicate the mining in the future?

**O** This possibility is not provided. There is a strict work with the specified algorithm without changing the conditions. Everything will work in the future as it was launched. The open source code and each node holder ensure compliance with these rules.



## Frequently Asked Questions



**12** **B** Will there be an increase in the number of sites where coins are exchanged? Are these sites reliable?

**0** The number of platforms will gradually increase and moreover, there will be a franchise of the exchanger **eGOLD.pro**. It will be distributed later to trusted persons.

**13** **B** It is quite difficult to master it, so the distribution will be difficult.

**0** It only seems that way because there is a lot of new things. In fact, everything is quite simple, especially for those who just use the wallet. You just need to understand and get used to working with the coin. Over time, using eGOLD will be as easy as riding a bike. Moreover, unlike other projects, there are a lot of explanations and all the features of the work are disclosed.

**14** **B** When accessing the node for making transactions, is the passphrase (private key) transmitted?

**0** It is not transmitted and it is completely secure if you run the file of the wallet eGOLD.html from a local computer, having previously matched the MD5 of the archive from which the wallet was taken, with several trusted resources at once. A signed transaction is sent to any node.

## Frequently Asked Questions

The private key is only used in the wallet. Therefore, you can use any node. The most that can happen, the transaction won't pass. The private key signs the transaction in the wallet and is immediately deleted.

**15** **B** So, is it impossible to connect to the node illegally and steal the passphrase (private key)?

**O** Yes. This will give nothing to the attacker. He will just get the signed transaction.

**16** **B** Please tell about interest accrual. Is it happening every second? And if you have 100 coins in your wallet, then in a month there will be exactly 104 coins?

**O** The interest is accrued every second on the entire amount and on the amount that has already been added during the previous second. That is, there is an accrual at a compound interest. If there were 100 coins on the balance, in a month there will be 104. And if there were 10,000 coins, then, according to the algorithm for the accrual of the compound interest, the balance will be 10,408 coins. At the same time, no additional actions for interest crediting are required, and you can manage the entire balance taking into account the accrued interest.

## Frequently Asked Questions



- 17** **B** And if the wallet has not been used for a long time, will the interest be accrued?
- 0** If there were no incoming or outgoing transactions on the wallet or node of the wallet for more than 3,650 days (almost 10 years), the interest will not be accrued on the balance anymore. The aim is to accelerate the node's operation.
- 18** **B** When does the referrals accrual take place?
- 0** Referrals accruals takes place when referrals make an outgoing transaction. At the same time, accrual through the referral program is rounded down to an integer. In other words, if you do not get at least one coin through the referral program, the accrual does not occur. The accrual is made at a compound interest rate and depends on the referral's balance at the time of the previous transaction of the referral, the referral's level, and the time elapsed since the previous referral's transaction. Crediting is made on the cumulative referral balance, which is credited to the main balance for any outgoing transaction.
- 19** **B** I see a transfer on the page of receipts from referrals, but it is orange and there was no accrual from it on the balance. When will the funds be credited to my balance?
- 0** Immediately after receiving funds from referrals, they are credited to the cumulative referral balance and the transaction for referrals in the wallet turns orange until the coins from the referral balance are credited to the main balance.

## Frequently Asked Questions

After the outgoing transaction in the wallet all coins are transferred from the referral balance to the main one. It is made so that the interest accrued on the wallet does not burn if the wallet has a lot of referrals. The credited interest is rounded down to whole coins.

20

**B** When does the new private key start working after changing the old one, and do I need to store the old private key?

**O** The new private key starts working immediately after changing the old private key. From now on, the old private key doesn't work any more and it will not work in the future. But if for some reason the transaction was not accepted by the nodes for more than 50% of the balance of the nodes' wallets in accordance with the PoS algorithm, it can be deleted and the new private key will not work, because it will disappear from the nodes. Therefore, do not delete the old private key for at least 5 minutes. The technology of saving a new private key is similar to the technology of saving new transactions and is written in the transaction itself, by signing the transaction with the new private key by the old private key.

21

**B** Which time zone is used in the official wallet eGOLD.html and what time is displayed in it on the bottom to the left?

**O** The current time zone of the device running the wallet eGOLD.html is displayed at the bottom left corner, in the section Transactions and

## Frequently Asked Questions

accruals from referrals. You can also see there the time of the data receipt from the node for transactions and separately your time for referral accruals. Each transaction has its time and date according to the time zone of the device running the wallet eGOLD.html.

**22 B Does the node owner have access to the contacts or the email stored on it?**

**O** All wallet numbers and their names in contacts are encrypted with a node's password, and the node owner is only given a hash of the password to access his node. Therefore, you can't view contact data without the password itself. The node owner can see the mail itself. When saving the email, it is once sent to the node in encrypted form and is not sent back in any form. Therefore, instead of E-mail, the asterisks are indicated in the wallet settings.

**23 B What means the second letter G with a plus and a number that is displayed to the right of the balance?**

**O** This is a bonus of **1%** of the current balance of the wallet for the node holders, which is accrued for any incoming and outgoing transaction, as well as for crediting rewards from referrals. Moreover, it is calculated at a compound interest with the difference between 5% and 4% over the past period, which is much more than just 1%.





- 24** **B** My own node is installed. There were no incoming and outgoing transactions on the wallet, and the G+ bonus was equal to 2 coins. Now G+ is equal to 0. Why?
- O** The node had a transaction from another wallet. Since coins has accumulated on the node's bonus balance, which is equal to +1% of the current funds on the wallet, and more than 24 hours have passed since the last accrual of interest on the wallet, the interest has been credited to the wallet balance. In other words, the node's algorithm provides periodical crediting of interest to the balance of the wallet linked to it, for transactions on the node immediately after receiving of 1 coin for the transaction made on it.
- 25** **B** How do I change the wallet linked to the node?
- O** To do this, you need to set the IP of another node in the wallet linked to your node, on the wallet's settings page [eGOLD.html](#), make any outgoing transaction, wait for the transaction to be processed and the node will unlink from the wallet, register a new wallet in the settings file `egold_sengs.php` and make a transaction with it through your node. After processing the transaction, the node will link to the new wallet. The node can be unlinked from the wallet automatically in 30 days and will stop working if there are no transactions on it and on the node's wallet during this time. As soon as the transaction is processed, the node will be linked to the wallet. If you lose access to the wallet, the node will unlink from it in 30 days and will stop working, if during this time there were no transactions on the node and on the wallet linked to it.
- 26** **B** What devices run the node and what are the system requirements?
- O** The node is incredibly light, and it works on any operating systems and devices of almost any power. The size of the database can be adjusted by

yourself up or down by changing the storage time of the transactions history. At the moment, I recommend you to allocate at least 1 GB for data storage.

The main requirements are: a static IP address and an installed PHP, version 7.1 or higher, with MySQL. All this can be installed on almost any operating system, including Windows, MAC OS, Linux, and even on mobile devices. The information about how to do this can be found on the Internet.

27

**B** The node doesn't work. What should I do?

**O** First, you need to install the latest version of the node (download and place the files in a folder, replace the old files if necessary). Then, if it doesn't work, you need to eliminate the causes of its incorrect work by configuring the environment in which it works. Maybe something is not installed in this environment, or there's the wrong version of the required components of the environment, or something is not working properly in the environment itself. You can do this yourself, either by using the technical support where the node is installed, or by asking other people, for example in a chat. Messages about errors that occur in the node itself are displayed immediately when accessing it, or you can look in more detail at the log of the system where it is installed, and to comment the first lines for logging with the parameters `ini_set`. If there are errors in the node, the log file `egold_error.log` will appear in the folder with node file `egold.php`. It opens in any text editor.

28

**B** I indicated the domain `mail.ddns.net` in the node settings. After a while, I stopped receiving emails with notifications about transactions. Why?

**O** Your domain must be paid and not higher than level 2. This domain is free and it has 3 levels (3 words and 2 dots between the names). Often, mail servers filter messages from these domains.



## 01

### Request to the node **without** parameters

**Example of accessing the node using the link:** [http://\[ip ноды\]/egold.php](http://[ip ноды]/egold.php)

**Example of page response:** `{"time":"1586582875", "noda":"91.106.203.179", "owner":"G-1000-00000-0000-00000", "transactionscount":"12", "datelasttransaction":"1586553129", "email_domain":"egold.pro", "noda_site":"https://www.egold.pro"}`

- **time** - the time on the node in UNIX format
- **noda** (this parameter is necessary for the node to work and it is written in the settings file `egold_settings.php` under the name `$noda_ip`) - IP of the current node
- **owner** (this parameter is necessary for the node to work and it is written in the settings file `egold_settings.php` under the name `$noda_wallet`) - the wallet of the node owner and it is needed for the node to work
- **transactionscount** - number of transactions since the installation of the node
- **datelasttransaction** - time of the last transaction for statistics and checking for node updates in UNIX format
- **email\_domain** (optional parameter that is specified in the settings file `egold_settings.php` under the name `$email_domain`) - the mail from which the node user receives notifications about transactions.
- **noda\_site** (optional parameter specified in the settings file `egold_settings.php` under the name `$noda_site`) - the domain of the node's site where you can access the node

## 02

### Request [**type=wallet**] - data on the wallet

**Example of a GET request:** [http://\[IP of the node\]/egold.php?type=wallet&wallet=G-1261-21479-](http://[IP of the node]/egold.php?type=wallet&wallet=G-1261-21479-39731)

**Example of a JSON response:** `{"time":"1567159479", "noda":"[IP of the node]", "email_domain":"[domain for sending notifications to email set in the settings]", "wallet":"G-1261-21479-1061-51551", "ref1":"G-4613-11305-1110-01414", "ref2":"G-1514-07813-0081-39731", "ref3":"G-1000-00000-0000-00000", "nodawallet":"91.106.203.180", "nodawalletuse":"91.106.203.180", "balance":"11578494", "percent_4":"121", "percent_5":"151", "balance_ref":"11578494", "date_ref":"1567243045", "percent_ref":"0", "height":"311", "date":"1567243045", "signpubnew":"", "signnew":"", "signpub":"[public key of the wallet]", "sign":"[signature of the last transaction]"}`

Every request must specify its type: **type**. The example uses request type **wallet** with an additional parameter **wallet** that is equal to the wallet number.

- **nodawallet** - node that belongs to this wallet
- **nodawalletuse** - the node from which this wallet made the last outgoing transaction
- **balance** - current balance
- **percent\_4** - accrual of seigniorage of 4% in the general
- **percent\_5** - accrual of seigniorage of 5%, if you use your node and the next payment will be done from it. If the payment is not made from its node, the node will stop working and the seigniorage will be equal
- to 4%

# Working with the node API



- **date\_ref** — date since the last outgoing transaction
- **percent\_ref** - the accumulated interest from referrals that is credited during an outgoing transaction
- **height** - the height of the last wallet transaction
- **date** - time and date of the last transaction in UNIX format
- **signpubnew** - new public key under the hash (if the parameter is not empty, the private key was changed) **signnew** - signature of the transaction with a new private key (if the parameter is not empty, the private key was changed) **signpub** - the public key of the wallet
- **sign** - signature of the transaction with the private key
- **balancetransactioncheck** - the balance of the transaction

\* If there are unprocessed transactions, you can not immediately make a payment for interest. The parameter **balancetransactioncheck** with the amount of unprocessed transactions is displayed, and the entire balance is reduced by this amount, taking into account the commission of 2 coins.

\*You can additionally view in the request the account settings on the node for sending notifications. To do this, the password parameter **password** is sent with hash SHAKE256 of 256 characters long (1024 bits) from the previous password, which was already written on the node under the hash SHAKE256 of 128 characters long. That is, first the hash SHAKE256 of 128 characters is taken from the previous password on the client's side and is sent to the node to set the password. Then, to verify the password on the node, the hash is taken one more time from the existing hash, but of 256 characters long, using the function `gen_sha3([password on the node],256)`, and is sent from the client and checked on the node. If there are two hashes, we count everything correctly and show the settings: whether there is email and with what thresholds.

\*If the password is changed (if it will differ from the set password), all saved contacts will be deleted, since they are encrypted for the old password.

**Example of a GET request:** : `http://[IP of the node]/egold.php?type=wallet&wallet=G-1261-21479-1061-51551&password=[password under hash SHAKE256 of 256 characters long from the previous one under hash SHAKE256 of 128 characters long]`

**Example of JSON response:** `{"time":"1567159668","noda":"[IP of the node]","wallet":"G-1261-21479-1061-51551","ref1":"G-4613-11305-1110-01414","ref2":"G-1514-07813-0081-39731","ref3":"G-1000-00000-0000-00000","nodawallet":"91.106.203.180","nodawalletuse":"91.106.203.180","balance":"11578494","percent_4":"121","percent_5":"151","height":"311","date":"1567243045","useremail":"[email]","useremailup":"10","useremaildown":"20","usersemaildateupdate":"1567159580","signpubnew":"","signnew":"","signpub":"[public key of the wallet]","sign":"[signature of the last transaction]"}`



## 03

### Request [type=nodas] - list of available nodes

Displays a list of nodes with requests statistics and number of wallets connected to them. Uses additional optional

parameters such as:

- **balancestart** - minimum balance of the node wallet
- **balancefinish** - maximum balance of the node wallet
- **nodauswalletstart** - minimum number of wallets using the node
- **nodauswalletfinish** - maximum number of wallets using the node
- **order** - a parameter for sorting the history output in direct order from the smallest date to the largest date with the parameter =asc, by default in reverse order by the date of last use. You can also display nodes by balance from larger to smaller with order=balance.
- **start** - what transaction will be the first in the output. Here and further in other types with multiple display: when =0 we start without skipping, when =1 we shift the output by 1, and so on.
- **limit** - the number of nodes displayed from the parameter start. By default, here and further in other types with multiple display =100.

**Example of a GET request:** `http://[IP of the node]/egold.php?type=nodas`

**Example of a JSON response:**

```
[{"noda":"91.106.203.181","wallet":"461311305111001414","balance":"17516750","walletsuse":"1","datelastuse":"1586553129"}, {"noda":"91.106.203.180","wallet":"126121479106151551","balance":"14365293","walletsuse":"3","datelastuse":"1586553129"}, {"noda":"91.106.203.179","wallet":"100000000000000000","balance":"18402445","walletsuse":"10","datelastuse":"1586587986"}, {"noda":"91.106.203.202","wallet":"151407813008139731","balance":"16955544","walletsuse":"2","datelastuse":"1586531746"}, {"noda":"91.106.206.144","wallet":"125811814158141411","balance":"20661642","walletsuse":"3","datelastuse":"1586086496"}]
```

- **noda** - IP of the available node
- **wallet** - the wallet to which the node is connected
- **balance** - the balance of the wallet to which the node is connected
- **walletsuse** - the number of wallets using the node
- **datelastuse** - date and time of the last transaction on the node in UNIX format

## 04

### Request [type=history] - transactions history

Uses additional optional parameters such as:

- **all** - when it is set =3 - it displays all the types of history, including transactions not accepted or rejected, all=2 - displays the transactions not accepted and accepted by all=1 displays only the transactions that have not been accepted, all=0 - displays only accepted transactions. By default, all=0





- **history** - displays all incoming and outgoing wallet transactions and must be equal to the wallet number
- **pin** - any number of up to 18 digits (to determine the transaction). The full match is shown with the equal sign
- **wallet** - number of the wallet where one can see only outgoing transactions
- **recipient** - the number of the wallet where one can see incoming transactions
- **date** - the date from which transactions are viewed in UNIX format
- **dateto** - the date until which transactions are viewed in UNIX format
- **height** - the height from which transactions are viewed
- **nodause** - the node that was last used by the wallet
- **order** - a parameter for sorting the history output in direct order from the smallest date to the largest date with the only possible parameter =asc, by default in reverse order
- **history\_exception** - parameter for excluding of history output with an array of wallets and heights for excluded transactions. Used for synchronization of transactions between nodes
- **start** - what transaction will be the first in the output. Here and further in other types with multiple mapping: if =0, we start without skipping, if =1, we shift the output by 1, and so on
- **limit** - the number of transactions displayed for the parameter start. By default, here and further in other types with multiple display =100, but with history by default =25

**Example of a GET request:** `http://[IP of the node]/egold.php?type=history&wallet=101245147116351512&height=2&start=0&limit=2&order=asc`

**Example of a JSON response** (here and further the responses "signpub" and "sign" are omitted):

```
[{"wallet":"10000000000000000000","recipient":"260749343133458182","money":"3","pin":"0","height":"2","nodawallet":"10000000000000000000","nodause":"95.169.185.90","nodaown":"1","date":"1590647968","signpubreg":"076b17561d742...","signreg":"656465383964653361613038...","signpubnew":"","signnew":"","signpub":"07f217eb1ad9100c0902290600e2030419b719d122e505bc223809612d1f2f091d96...","sign":"326239...","checkhistory":"1"},{"wallet":"10000000000000000000","recipient":"932333570717013652","money":"3","pin":"0","height":"3","nodawallet":"10000000000000000000","nodause":"95.169.185.90","nodaown":"1","date":"1590648025","signpubreg":"07dd1ba61b731c561b6000862bee1...","signreg":"33316531386335633864656132306633...","signpubnew":"","signnew":"","signpub":"07f217eb1ad9100c0902290600e2030419b719d122e50e...","sign":"323432633...","checkhistory":"1"}]
```

- **wallet** - the wallet of the sender
- **recipient** - the wallet of the recipient
- **money** - transfer amount with no fee charged
- **pin** - pincode of up to 18 digits in clear text
- **height** - the height of the transaction on the wallet
- **nodawallet** - the node of this wallet
- **nodause** - the node that the wallet used for this transaction
- **nodaown** - whether the node from which the transaction was made belongs to this wallet
- **date** - date and time of the transaction in UNIX format
- **signpubreg** - the public key of the new wallet under the hash (if the parameter is not empty, it means that a new wallet was created with the number in the field 'recipient')
- **signreg** - signature of the part of transaction for creating a new wallet (if the parameter is not empty, it means that a new wallet was created with the number in the field 'recipient')
- **signpubnew** - new public key under the hash (if the parameter is not empty, the private key was changed)
- **signnew** - signature of the transaction with a new private key (if the parameter is not empty, the private key was changed)
- **signpub** - the public key of the wallet
- **sign** - the signature of transaction with a private key
- **checkhistory** - transaction status: 0 - transaction is not processed, 1 - transaction is accepted, 2 - transaction is rejected.

\* If a request without the parameter **all=3** with the parameter **'history' equal to the wallet**, the number is output at the end of the array indicating the total number of records in the history for this request without pagination. This is done in order to know how many pages with requests there are in total.

**Example of a GET request:** `http://[IP of the node]/egold.php?type=history&history=101245147116351512`

**Example of a JSON response** (here and further the responses "signpub" and "sign" are omitted): `[{"wallet":"101245147116351512", "recipient":"1", "money":"1", "height":"2", "nodawallet":"100000000000000000", "nodaused":"91.106.203.179", "nodaown":"0", "date":"1562182374", "signpubreg":"","signreg":"","signpubnew":"","signnew":"","signpub":"","sign":""},{ "wallet":"101245147116351512", "recipient":"1", "money":"1", "height":"3", "nodawallet":"101245147116351512", "nodaused":"91.106.206.144", "nodaown":"1", "date":"1562182398", "signpubreg":"","signreg":"","signpubnew":"","signnew":"","signpub":"","sign":"","checkhistory":"1"},10]`

## 05

### Request [type=referrals] - transactions history by referrals

Uses additional optional parameters such as:

- **wallet** - number of the wallet that made transaction and to whom it credited referral bonuses **ref** - number of the wallet where one can see all its referrals of all levels
- **ref1** - the number of the wallet where one can see all of its 1st level referrals, if there is no parameter **ref**
- **ref2** - the number of the wallet where one can see all of its 2nd level referrals, if there is no parameter **ref**
- **ref3** - the number of the wallet where one can see all of its 3rd level referrals, if there is no parameter **ref**
- **height** - the height from which one can see the transaction
- **nodaused** - the node that the wallet used for a transaction for the last time **date** - the date from which transactions are viewed in UNIX format
- **dateto** - the date up to which transactions are viewed in UNIX format
- **order** - the parameter for sorting the history output in direct order from the smaller date to the bigger date according to the date of the last transaction with the only possible parameter =asc, by default in reverse order
- **start** - what transaction will be the first in the output.
- **limit** - the number of transactions displayed for the parameter start.

\* At the end of the array, the number is output indicating the total number of entries in the history of referral accruals for this request, without pagination. This is done in order to know how many pages with requests there are in total.

**Example of a GET request:** `http://[IP of the`

`node]/egold.php?type=referrals&wallet=126121479106151551&height=2&start=0&limit=1&order=asc` **Example of a JSON response:** `[{"wallet":"126121479106151551", "ref1":"461311305111001414", "ref2":"151407813008139731", "ref3":"100000000000000000", "money1":"201214", "money2":"100607", "money3":"50303", "height":"22", "date":"1562940727"},10]`

- **wallet** - a wallet that is a follower of referrers of different levels
- **ref1** - the 1st level referrer, that is, a wallet that created the wallet **wallet**
- **ref2** - a 2nd level referrer, that is, the wallet that created the wallet of the wallet that created the wallet **wallet**
- **ref3** - the 3rd level referrer



- **money1** - the amount of remuneration of **1%** per month from the balance of the **wallet** to be remitted to the **1st level** referrer outgoing operation, and 1.25% if the referral has a
- **money2** - the amount of remuneration of **0.5%** per month from the balance of the **wallet** to be remitted to the **2nd level** referrer outgoing operation, and 0.625% if the referral has a
- **money3** - the amount of remuneration of **0.25%** per month from the balance of the **wallet** to be remitted to the **3rd level** referrer outgoing operation, and 0.3125% if the referral has a
- **height** - the height of the transaction from which the reward was
- **date** - date and time of the transaction in UNIX format from which the reward was
- at the end of the digit, this is the number of rows in this

## 06

### Request [type=referralwallets] - displays a list of referral's wallets for this wallet

Uses additional optional parameters such as:

- **wallet** - wallet
- **ref** - number of the wallet by which all its referrals of all levels are viewed
- **ref1** - number of the wallet by which all its referrals of the 1st level are viewed, if there is no parameter ref
- **ref2** - number of the wallet by which all its referrals of the 2nd level are viewed, if there is no parameter ref
- **ref3** - number of the wallet by which all its referrals of the 3rd level are viewed, if there is no parameter ref
- **height** - the height from which the wallets are viewed for the last transaction
- **nodause** - the node that the wallet last used for the transaction
- **date** - the date from which the wallets are viewed for the last transaction in UNIX format
- **dateto** - the date up to which wallets are viewed for the last transaction in UNIX format
- **order** - parameter for sorting the history output in direct order from the smallest date to the largest date - possible parameters =asc- by increasing the date, =balanceasc - by increasing the balance and by descending the date, =balancedesc - by descending the balance and by descending the date, without parameters sorting by date in the reverse direction. Sorting the balance without calculating of interest
- **start** - what transaction will be the first in the output.
- **limit** - the number of transactions displayed for the parameter start.

By default, sorting goes in reverse order from the largest date of change to the smallest date.

\* The referrals' balance is displayed without taking into account the interest received for the last transaction until the current time

\* At the end of the array, is displayed a number indicating the total number of entries in the history of referral accruals for this request, without pagination. This is done in order to know how many pages with requests there are in total.

#### Example of a GET request:

[http://\[IP of the node\]/egold.php?type=referralwallets&ref=10000000000000000000&height=2&start=0&limit=2&order=asc](http://[IP of the node]/egold.php?type=referralwallets&ref=10000000000000000000&height=2&start=0&limit=2&order=asc)

#### Example of a JSON response:

```
[{"wallet":"942940567813543929","ref1":"735587879887580955","ref2":"10000000000000000000","ref3":"0","noda":"","nodause":"91.106.2
```



```
03.179","balance":"478","date":"1590853829","height":"2"},{"wallet":"381636645604930508","ref1":"100000000000000000","ref2":"0",  
"ref3":"0","noda":"","nodaused":"5.181.110.217","balance":"100383290","date":"1590901712","height":"7"},62]
```

- **wallet** - the referral's wallet
- **ref1** - the **1st level** referrer, i.e. the wallet that created the wallet **wallet**
- **ref2** - a **2nd level** referrer, that is, the wallet that created the wallet of the wallet that created the wallet **wallet**
- **ref3** - the **3rd level** referrer
- **noda** - the node that belongs to the wallet
- **nodaused** - the node that the wallet used for a transaction for the last time
- **balance** - the wallet's balance
- **date** - date and time of the last transaction in UNIX format
- **height** - the height of the last transaction
- at the end of the digit, this is the number of rows in this response

## 07

### Request [type=referralresults] - output of the total number and the volume referrals' level

One of the parameters must be used:

- **ref** - number of the wallet by which all its referrals of all levels are viewed
- **ref1** - the number of the wallet where one can see all of its 1st level referrals, if there is no parameter **ref** **ref2** -
- the number of the wallet where one can see all of its 2nd level referrals, if there is no parameter **ref** **ref3** - the
- number of the wallet where one can see all of its 3rd level referrals, if there is no parameter **ref**

\* Referrals' balance is shown without taking into account the interest of the last transaction until the current time

**Example of a GET request:** [http://\[IP of the node\]/egold.php?type=referralresults&ref=461311305111001414](http://[IP of the node]/egold.php?type=referralresults&ref=461311305111001414)

**Example of a JSON response:** {"count1":"1","balance1":"13969582","count2":"6","balance2":"413713","count3":"101","balance3":"300124234"}

- **count1** - the number of the 1st level referrals of this wallet specified in the parameter of the request
- **balance1** - the balance of the 1st level referrals of this wallet specified in the parameter of the
- **count2** - the number of the 2nd level referrals of this wallet specified in the parameter of the request
- **balance2** - the balance of the 2nd level referrals of this wallet specified in the parameter of the
- **count3** - the number of the 3rd level referrals of this wallet specified in the parameter of the
- **balance3** - the balance of the 3rd level referrals of this wallet specified in the parameter of the



08

Request [type=**height**] - getting the height of the wallet and the possibility of the transaction. Serves as an auxiliary request before the transaction

An additional parameter is used: wallet

**Example of a GET request:** `http://[IP of the node]/egold.php?type=height&wallet=158011521111537971`

**Example of a JSON response:** `{"time":"1567159024","noda":"91.106.203.179","balance":99901,"height":"1","date":"1567158999"}`

- **time** - the time on the node in UNIX
- **noda** - IP of the current
- **balance** - the balance of the wallet specified in the
- **height** - the height of the wallet specified in the parameter
- **date** - the date of the last transaction of the wallet specified in the parameter

09

Request [type=**send**] - carrying out of a transaction where can be made the change of public and private key, the change of a password for managing contacts and E-mail notifications, the new wallet is registered

Uses additional mandatory parameters:

- **wallet** - the wallet from which we transfer funds
- **recipient** - the recipient's wallet where we transfer funds
- **money** - transfer amount excluding the commission fee of 2 coins
- **pin** - any number consisting of up to 18 digits (to determine the transaction). By default, it should be equal to 0
- **height** - the height of this transaction of the wallet
- **signpub** - the public key of the wallet transferring funds
- **sign** - the signature of the transaction with the private key of the wallet transferring funds is formed from the line: wallet + recipient + money + pin + height + noda + signpubreg + signreg + signpubnew + signnew. Where "+" is the addition of lines. Signreg = '30' + sha\_dec(signpubreg) (3 coins are sent for registration and the height of the new wallet is 0), signpubnew= sha\_dec(signpubnew), signnew= wallet + height. sha\_dec - function in javascript. Signpubreg, signreg, signpubnew, and signnew can be empty values.

Uses additional optional parameters such as:

- **signpubnew** - hash for the new public key
- **signpubnew\_check** - public key for checking the correctness of the new private key in the node. When using this parameter, the transaction will fail if the private key was incorrect. At the same time, the public key is used only for checking its validity and only on this node, and is immediately deleted from memory. If this parameter is not passed, the validity of the private key will not be checked
- **signnew** - signature for verifying a new private key
- **signpubreg** - public key for creating a new wallet
- **signreg** - signature for creating a new wallet



- **password** - password for managing contacts and E-mail notifications. The client gets a password of 128 characters. Which is saved in the node as it is and then its hash of 256 characters is checked. For more information see point 2 on request [type=wallet]

**Example of a GET request:** `http://[node's IP]/egold.php?type=send&wallet=10000000000000000&recipient=126121479106151551&money=5&pin=0&height=2&signpub=xxx&sign=xxx`

**Example of a JSON response:** `{"time":"1592593054","noda":"91.106.203.179","date":"1592593000","send":"true","recipient":"G-7355-8875-80955"}`

- **time** - the time on the node in UNIX
- **noda** - IP of the current
- **date** - transaction date of the wallet specified in the parameter **wallet** in UNIX
- **send** - the node's response: **true** - the transaction has
- **signpubnew\_check** - the node's response: **true** - means that the public key is checked, so the private key is correct. Displays only if the parameter **signpubnew\_check** with a public key was received
- **recipient** - the recipient's wallet

## 10 Request [type=synch] - the node's synchronization

It does not use any additional parameters and is installed in **CRON with the access of every minute**. It is recommended to install to CRON a request via a PHP query in the form «usr/local/bin/php ~/[path to the file egold.php]/egold.php synch» Where **synch** is the transmitted parameter **type** for synchronization, and the PHP request must be of version 7.1 or higher. You can also use an addressing like "http://[IP of the node]/egold.php?type=synch»

## 11 Request [type=synchwallets] - addressing to the synchronizable

Uses an additional optional parameter only in the form of a POST request: **wallets** - an array of wallets to display the node's data for synchronization.

Sorting goes in direct order from the smallest date to the largest date and displays up to 100 wallets with priority to wallets listed in the array.

**Example of a POST request with the parameter [wallets=10000000000000000]:** `http://[IP of the node]/egold.php?type=synchwallets`

**Example of a JSON response:** `{"time":"1567160232","noda":"91.106.203.179","synchwallets":[{"wallet":"10000000000000000","ref1":"310121260159274912","ref2":"0","ref3":"0","noda":"","nodauser":"","balance":"3","height":"0","date":"1560612600","signpubnew":"","signnew":"","signpub":"","sign":"","413712101472131111":{"wallet":"41712101472131111","ref1":"126121479106151551","ref2":"461311305111001414","ref3":"151407813008139731","noda":"","nodauser":"","balance":"3","height":"0","date":"1560917136","signpubnew":"","signnew":"","signpub":"","sign":""}}]}`



- **time** - the time on the node in UNIX
- **noda** - IP of the current
- **synchwallets** - an array of wallets, where the wallet number goes first and then its data corresponding to the request type=**wallet** with the same data types

## 12 Request [type=**contacts**] - displays the wallet's contacts on the node from which it makes transactions

To use this request, you must first make a transaction on the node along with setting a password for managing settings on this node. Contacts will be stored and displayed only upon request to this node. A maximum of 100 contacts can be stored

Uses additional mandatory parameters:

- **wallet** - the wallet where one can view contacts
- **password** - the password for accessing the wallet's data stored on the node is compared to the received one using the function `gen_sha3([password on the node],256)`. That is, the parameter of the password **password** under hash SHAKE256 of 256 characters long (1024 bits). The client gets a password of 128 characters. Which is saved in the node as it is and then its hash of 256 characters is checked. For more information, see point 2 on request [type=**wallet**] and point 9 on request [type=**send**]

Uses additional optional parameters such as:

- **contacts** - array of contacts passed to the node only by means of a POST request. If this parameter is not passed, the list of existing contacts is displayed. If it is passed empty, all contacts are deleted. The parameters **recipient** and **name** are passed to the server. In fact, it can be any line with English letters and numbers. The maximum length is 255 characters. The parameters for sending correspond to the parameters in the **JSON** response for displaying contacts. See the **JSON** example below.

**Example of a JSON response for saving and changing contacts:** {"time":"1567160321","noda":"91.106.203.179","contact":"save"}

**Example of a JSON response for deleting contacts (if the request was sent without contacts):**

{"time":"1567160493","noda":"91.106.203.179","contact":"del"}

**Example of a JSON response for displaying contacts:**

[{"recipient":"2000000000000000000","name":"name2"}, {"recipient":"3000000000000000000","name":"name3"}]

- **recipient** - the wallet's number saved in the contact book
- **name** - the wallet's name



## 13

### Request [type=email] - E-mail notifications for wallet transactions on the node from which it makes transactions

To use this request, first you must make a transaction on the node along with setting a password to manage settings on this node. Only nodes with their mail domain registered in the settings file can send E-mail notifications. You can find out about it when accessing the node with an empty request [http://\[IP of the node\]/egold.php](http://[IP of the node]/egold.php) if the parameter email domain is filled in

Uses additional mandatory parameters:

- **wallet** - the wallet from which we transfer funds
- **password** - the password for accessing the wallet's data stored on the node is compared to the received one using the function `gen_sha3([password on the node],256)`. That is, the parameter of the password **password** under hash SHAKE256 of 256 characters long (1024 bits). The client gets a password of 128 characters. Which is saved in the node as it is and then its hash of 256 characters is checked. For more information, see point 2 on request [type=wallet] and point 9 on request [type=send]

Uses additional optional parameters such as:

- **email** - the email address where notifications will be sent to. If this parameter is absent when email is present on the node, it will be deleted and all other parameters will be cleared.
- **up** - the integer indicating the minimum threshold for the amount of outgoing transactions at which notifications will be sent
- **down** - the integer indicating the minimum threshold for the amount of incoming transactions at which notifications will be sent

**Example of a JSON response for saving email and parameters:** {"time":"1567160588","noda":"91.106.203.179","emailwallet":"save"}

**Example of a JSON response for deleting email and parameters:** {"time":"1567160606","noda":"91.106.203.179","emailwallet":"del"}

- **time** - time on the node in UNIX format
- **noda** - IP of the current node
- **emailwallet** - response for saving contacts: **save** - contacts are saved, **del** - contacts are deleted

## 14

### Request [type=balanceall] - the total balance of all wallets, the accrued interest

\* The balance of all wallets is displayed without taking into account the interest of the last transaction until the current time

**Example of a GET request:** [http://\[IP of the node\]/egold.php?type=balanceall](http://[IP of the node]/egold.php?type=balanceall)

- **time** - the time on the node in UNIX
- **noda** - IP of the current node
- **balanceall** - the balance of all wallets without accrued interest



## 15 Request [type=walletscount] - total number of wallets

Example of a GET request: `http://[IP of the node]/egold.php?type=walletscount`

Example of a JSON response: `{"time":"1583326557","noda":"91.106.203.179","walletscount":"234"}`

- **time** - time on the node in UNIX format
- **noda** - current node's IP
- **walletscount** - number of all wallets

## 16 Request [version] - displays the current version of the node's file egold.php

Example of a GET request: `http://[IP of the node]/egold.php?version`

Example of a JSON response: `{"version": "1.6"}`

## 17 Example of making a transaction in PHP

```
<?php
include './egold_crypto/falcon.php';//enable encryption

function egold_send($params,$noda){
    $url = 'http://'.$noda.'/egold.php';
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
    curl_setopt($ch, CURLOPT_TIMEOUT_MS, 15*1000);
    $json = curl_exec($ch);
    curl_close($ch);
} return json_decode($json, true);

$noda= '[IP of the node that we will use]';
$wallet= '[the wallet from which we send only 18 digits]';
$recipient= '[wallet where we send only 18 digits]';
$moneys=
$pin= '[pin - any number consisting of 1 to 18 characters]';
$falcon_k= '[Private key of the wallet from which we send]';
```



```
$params = array(//POST parameters for sending
    'type' => 'height',
    'wallet' => $wallet
);
$height= egold_send($params,$noda);

if(isset($height['height']) && $height['height']>0){$height=$height['height']+1;}
else {print_r($params);print_r($height);exit();/*error*/}

$falcon_p= Falcon\createPublicKey($falcon_k);//creation of the public key
$str_s=$wallet.$recipient.$money.$pin.$height.$noda;//the line to sign
$falcon_s= Falcon\sign($falcon_k, $str_s);//signature of a line with a private key

$params = array(//POST parameters for sending
    'type' => 'send',
    'wallet' => $wallet,
    'recipient' =>
        $recipient, 'money' =>
        $money, 'pin' => $pin,
    'height' => $height,
    'signpub' => $falcon_p,
    'sign' => $falcon_s
);
$json_send= egold_send($params,$noda);

if(isset($json_send['send']) && $json_send['send']=='true'){echo 'true';/*coins are sent*/} else
{print_r($params);print_r($json_send);/*error*/}
?>
```

## 18 Example of creating a wallet in PHP

```
<?php
include ' ./egold_crypto/falcon.php';//enable encryption

function egold_send($params,$noda){
    $url = 'http://'.$noda.'/egold.php';
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt ($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS,
        $params);
    curl_setopt($ch, CURLOPT_TIMEOUT_MS, 15*1000);
```





```
$json = curl_exec ($ch);
curl_close ($ch);
return json_decode($json, true);
}

function bchexdec($hex){//long numbers
    $dec = 0; $len = strlen($hex);
    for ($i = 1; $i <= $len; $i++)$dec = bcadd($dec, bcmul(strval(hexdec($hex[$i - 1])), bcpow('16', strval($len - $i))));
    return $dec;
}

function sha_dec($str){return substr(bchexdec(gen_sha3($str,19)),0,19);}//generating a hash of 19 numbers

$node= '[IP of the node that we will use]';
$wallet= '[the wallet from which we send only 18 digits]';
$recipient= '00';
$money= '3';
$pin= '0';
$gen_pass= '[random line of 50,000 digits to generate a new wallet]';
$falcon_k= '[Private key of the wallet from which we send]';

$params = array();//POST parameters for sending
    'type' => 'height',
);
    'wallet' => $wallet
$height= egold_send($params,$node);

if(isset($height['height']) && $height['height']>0){$height=$height['height']+1;}
else {print_r($params);print_r($height);exit();/*error*/}
list($falcon_k_reg,$falcon_p_reg)= Falcon\createKeyPair(128,$gen_pass);//creation of a new pair of public and private key
$str_s_reg='30'.sha_dec($falcon_p_reg);//the line for signature of the new wallet with the private key
$falcon_s_reg= Falcon\sign($falcon_k_reg, $str_s_reg);//signature of the new wallet with the key
$falcon_p= Falcon\createPublicKey($falcon_k);//creation of the public key
$str_s= $wallet.$recipient.$money.$pin.$height.$node.$falcon_p_reg.$falcon_s_reg;//the line that we sign with the key of an
existing wallet
$falcon_s= Falcon\sign($falcon_k, $str_s);//signature of the line with the private key of an existing wallet

$params = array();//POST parameters for sending
    'type' => 'send',
    'wallet' => $wallet,
    'recipient' =>
    $recipient, 'money' =>
    $money, 'pin' => $pin,
    'height' => $height,
    'signpubreg' => $falcon_p_reg,
```



```
'signreg' => $falcon_s_reg,
'signpub' => $falcon_p,
'sign' => $falcon_s
);

$json_send= egold_send($params,$noda);

if(isset($json_send['walletnew']) && strlen(preg_replace("/[^0-9]/i","", $json_send['walletnew']))==18){echo 'wallet_new=
'. $json_send['walletnew']. ' private_key= ' . $falcon_k_reg; /*the wallet is created*/}
else {print_r($params);print_r($json_send);/*error*/}
?>
```

## 19 Example of changing the private key in PHP

```
<?php
include './egold_crypto/falcon.php';//enable encryption

function egold_send($params,$noda){
    $url = 'http://'. $noda. '/egold.php';
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt ($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
    curl_setopt($ch, CURLOPT_TIMEOUT_MS, 15*1000);
    $json = curl_exec ($ch);
    curl_close ($ch);
    return json_decode($json, true);
}

function bchexdec($hex){//long numbers
    $dec = 0; $len = strlen($hex);
    for ($i = 1; $i <= $len; $i++)$dec = bcadd($dec, bcmul(strval(hexdec($hex[$i - 1])), bcpow('16', strval($len -
    return $dec;
}

function sha_dec($str){return substr(bchexdec(gen_sha3($str,19)),0,19);}//generating a hash of 19
numbers

$noda= '[IP of the node that we will use]';
$wallet= '[the wallet from which we send only 18 digits]';
$recipient= '1';
$money= '1';
$pin= '0';
```

# Working with the node API



```
$gen_pass= '[random line of 50,000 digits to generate a new wallet]';
$falcon_k= '[Private key of the wallet from which we send]';
$params = array();//POST parameters for sending
    'type' => 'height',
    'wallet' => $wallet
);
$height= egold_send($params,$noda);

if(isset($height['height']) && $height['height']>=0){$height=$height['height']+1;}
else {print_r($params);print_r($height);exit();}

list($falcon_k_new,$falcon_p_new)= Falcon\createKeyPair(128,$gen_pass);//creation of a new pair of public and private key
$signpubnew= sha_dec($falcon_p_new);//generation of a hash of the new private key
$signnew= Falcon\sign($falcon_k_new, $wallet.$height);//generation of a signature for the new private key
$falcon_p= Falcon\createPublicKey($falcon_k);//creation of the public key
$str_s= $wallet.$recipient.$money.$pin.$height.$noda.$signpubnew.$signnew;//the line we sign
$falcon_s= Falcon\sign($falcon_k, $str_s);//signature of a line with a private key

$params = array();//POST parameters for sending
    'type' => 'send',
    'wallet' => $wallet,
    'recipient' =>
        $recipient, 'money' =>
        $money, 'pin' => $pin,
    'height' => $height,
    'signpubnew' => $signpubnew,
    'signpubnew_check' => $falcon_p_new,
    'signnew' => $signnew,
    'signpub' => $falcon_p,
    'sign' => $falcon_s
);
$json_send= egold_send($params,$noda);

if(isset($json_send['send']) && $json_send['send']=='true'){echo 'private_key_new= '.$falcon_k_new;/*private key is changed*/} else
{print_r($params);print_r($json_send);/*error*/}
?>
```



## 20

### Example of accepting funds in PHP

*\* It is desirable to install more than one node on different servers and to check the receipt of funds in 5 minutes from the moment of transaction receipt, and if the transaction is confirmed in 5 minutes on all nodes at once, you may credit the funds.*

*\* If the funds are transferred to a shared wallet, each user must be assigned a unique numeric number consisting of up to 18 digits. And if an incoming transaction to the shared wallet will have a pin corresponding to the user's numeric number for incoming transactions, the funds are credited to this user.*

*\* I strongly recommend to allow the user to change the pin code, so that no one could accurately determine that the funds are credited to the same user.*

*\* After accepting the transaction, record the wallet number and the height from which it came, and then skip transactions like this and process only the new ones.*

*\* Incoming transactions can be checked immediately in MySQL in the table [database prefix]\_history, since the eGOLD coin database is stored in MySQL.*

```
<?php
function egold_send($params,$noda){
    $url = 'http://'. $noda. '/egold.php';
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS,
    $params);
    curl_setopt($ch, CURLOPT_TIMEOUT_MS, 15*1000);
    $json = curl_exec($ch);
    curl_close($ch);
    return json_decode($json, true);
}

$noda= '[IP of the node that we will use]';
$recipient= '[wallet that receives funds in the form of 18 digits]';
$pin= '[pin - any numbers from 1 to 18 characters to determine the user who receives funds]';
$dateto= 5;//how many minutes should pass from the transaction receipt
$date= 24*60;// the limit in minutes from the receipt of the transaction beyond which we do not look

$params = array(//POST parameters for sending
    'type' => 'history',
    'recipient' => $recipient,
    'pin' => ($pin?$pin:0),
```

## Working with the node API



```
'date' => time()-$date*60,  
'dateto' => time()-$dateto*60,  
'limit' => 100,  
'order' => 'DESC'  
);  
$json_history= egold_send($params,$noda);  
  
if(isset($json_history[0])){print_r($json_history);/*incoming funds processing*/} else  
{print_r($params);print_r($json_history);/* error*/}  
?>
```