Introduction

There have been many iterations of a decentralized digital currency over the last 30 years. Some have been more successful than others, but until now, all have failed in becoming a non-fiat, decentralized digital currency that can be used by the mainstream public for every day small to large irreversible purchases, in an efficient and cost effective way.

As of this writing, for example, Bitcoin is very popular, as are many other block chain currencies, however, each and every one of them has failed at every point listed above.

They are all 100% fiat, they are not usable by the mainstream public, and they cannot provide small or large irreversible purchases in an efficient and cost effective way. While there are some that do a much better job than Bitcoin, they do have security/ reliability concerns, they are still 100% fiat, and they are still not suitable for use by the mainstream public.

We propose Anocoin as the final solution.

Anocoin is not fiat, because Anocoin is the only currency accepted on the Anopod Network. Miners that produce the Anopod Network are paid in Anocoin only. If we were to view the network as an individual, then the network can exchange Anocoins for network. Imagine you are a train, and you exchange train coins for track maintenance/ building. Train coins would be worth what track maintenance/ building is worth. So, Anocoins are worth what the Anopod Network creation/ maintenance is worth.

Anocoin is a decentralized digital currency that is acceptable for mainstream use. In fact, we propose that Anocoin is simpler to use and more secure than an online bank account.

Anocoin is suitable for mainstream users, is more efficient and cost effective than current banking systems, and provides irreversible payments.

We compare the efficiency and cost to current mainstream payment solutions such as Visa and Mastercard, because there is no point in comparing to block chain digital currencies as these are preposterous in both their transaction times, capacity, and cost to run.

A digital currency that cannot compete with the current mainstream payment processing systems, will never be able to replace these systems. Anocoin is more than an improvement over block chain, it is an improvement over the current payment systems of our time.

Although in some moments repetitive, we will explore every element of the entire Anocoin ecosystem, as each is either an introduction of a new technology or an improvement over an existing one.

To simplify things, we will only focus on Anocoin 1.0 and its ecosystem as it currently exists today. So, I will avoid saying things like, "currently it will work like this, but...", "currently ...", "in the future ..." or "for now this is the ...". We do however, in the future, expect major changes to both the ecosystem in which Anocoin runs, and Anocoin itself.

Although unpopular to do so today, we will use the term "user", to refer to anyone who uses Anocoin. If you prefer a different term, then you can imagine that term in your head in place of "user" as you read through this white paper. You are also welcome to print this white paper, and replace the word "user" with the term of your choice.

Also, a final note on the style in which this white paper will be written. Most white papers, in my opinion, seem to be written under the understanding that no one will ever read them, other than maybe the intro, and a quick glance over.

We've written this white paper in the hopes that to a willing ardent reader, they will be able to comprehend, understand, become enlightened about new things, and follow along with this white paper. So that it can be both educational, fun, and enlightening to its audience.

So, let's begin with a view from the user's UI side of things:

User Facing

Simplicity and security to the user is the main focus of Anocoin. There are two ways a user can interact with Anocoin:

Choice 1: An Anopod hosting company's internal account management.

This is the easiest, safest, and most secure way for a user to interact with the Anocoin system.

Anopodhost.com, anohost.com, and anopod.com are all anoPod hosts. Users can rent miners from them, and they can also manage their anocoin accounts through them, including creating an account, trading coins, etc... These hosts also have stand alone wallets, to easily buy/sell things with Anocoin and/or trade Anocoin.

The hosts make it very easy to set up an account, establish a public ledger ID, and handle security issues, such as issuing a "public key", and storing and proper usage of the "private key". We'll discuss further along in this white paper what the user's private and public keys are. For now, think of it as a password and proof of password.

While accessing Anocoin through a host, no person or group will ever be aware of what the "users" private key is. Not the host, not the user, not the banker, not the owner of the company or person which hosts the private key. This makes storage of the user's private key very secure.

Choice 2: Independently.

Anocoin can be interacted with independent of any host, via anything that can connect to the internet.
This could be an independently produced wallet, the terminal on your mac or the command line on a windows, etc...

In this way Anocoins accounts can be created, Public Ledger IDs can be established, coins can be traded, etc...

A major drawback to any of the options under this method is that the user or creator of the independent system is responsible for managing the private key and other security concerns.

We believe that less than 0.1% of users will choose this method, yet we feel it's important to enable, in order to prove decentralization and to appear more legitimate.

Interacting with Anocoin through a host walk through and technical details:

Let's follow Sam as he creates an account:

Sam makes an Anocoin account at anopodhost.com.

He does this by opening the link to his wallet, and then either clicks a button to create his wallet.

I will refer to any written program simply as 'the code' to ensure clarity for all readers, including those who are not programmers.

After clicking the button, the code creates a private key which is stored in a secure server, this private key is associated with Sam by his user ID on anopodhost.com.

This key is an Elliptic Curve P-384 - SHA 384 Digest key, and after its creation the code on anopodhost.com sends the public key of Sam's private key to the "Bankers" so that it can be stored on his Public Ledger, which if you imagine a table with rows and columns, this is stored under a column labeled, A410.

I introduced some new words in the last paragraph, "Bankers", "Public Ledger", and A410.

"Bankers" and Public Ledger will be explored shortly in this white paper.

A410 is just what we call Sam's Public Key derived from his private key.

This A410 will be used to verify that it is Sam, when his identity needs to be confirmed. For example, when making a transaction, or buying something with Anocoin.

This mostly wraps up everything that happens from the user's side.

Anopodhost.com stores Sam's private key on a secure server, separate from any other anopodhost.com server. Although anopodhost.com stores this, no one at anopodhost.com knows or has any way to find out what Sam's private key is.

It is possible that the system could be hacked so that his private key could be used, but it's not possible to find out what the key is.

Security processes are in place to prevent server take over.

Bankers

Bankers could be to Anocoin what miners are to Bitcoin.

Bankers count coins, and keep records.
A banker is a person somewhere in the world, whose identity is known, is a member of the private Anocoin Bankers Association club, and who has a 50,000 Anocoin stake.

No one banker has a right to be in the Banker's club.

If 80% of bankers vote out a banker, then they are out. Their stake will be returned to them 90 days after they stopped being a banker.

Banker's, and their 80% vote threshold, control all of Anocoin.

A banker has ownership over a computer, which is running the Anocoin Banker Software, and is joined to the Anocoin Banker Network.

The Anocoin Banker Software is built around several Database records and systems. Let's explore each.

The Public Ledger.

A table of all Anocoin accounts, which is stored as a series of "append only" records.

Append only refers to a system of file keeping where records can only be added, they cannot be altered or deleted. Going forward in this white paper, we will refer to this as a WORM DB, which stands for write once read many database. This means things can only be added to the database, but they can be read as many times as needed. Let's move on.

Other decentralized digital currencies have failed to protect their users from losing their coins. Something as simple as forgetting a long private key can cause permanent loss of their coins.

The Public Ledger Id is Anocoin's answer to this problem.

By taking basic personal information and sha 256 hashing it - I won't go into what a sha 256 hash is but feel free to search it on the internet - a unique Public Ledger Id is created. In the event of loss of coins, a user is able to prove ownership, because only they possess the personal details that equate to that specific hash.

Please imagine the Public Ledger as a table of rows and columns, as a banker in the olden days might have kept in a book.

Rather than updating the columns in a row, a row with the updates is simply added, as this is a WORM DB.

Much like the banker of old added a new row for each change in the account, rather than attempting to erase and update each line.

Each new row could be added as a snap shot to the WORM DB, but it's not practical to add a snapshot of the entire DB every single time there is an update.

Therefore in addition to the Public Ledger Table there is a Pending Public Ledger Updates table.

Once every 24 hours, after verification, this pending table is added to the Public Ledger snapshot and a new snapshot is made.

Next, let's talk a bit more about what a WORM DB is.

WORM DB

The WORM DB exists in many forms today, but this can lead to confusion because many DBs that claim to be WORM are not. While these dbs do tend to be WORM for regular use, they do not hold as WORM when faced with a malicious attacker.

The Anocoin WORM DB is impossible to delete or modify by anyone, including the Banker, the core developers, or any intruder.

Each banker hosts their Anocoin DB on a google cloud storage bucket, with delete protection turned on, and locked.

This means the banker's project cannot be deleted, Google cannot delete or alter the contents of the bucket, the banker cannot, core developers cannot, and no intruder can either.

The DB in this bucket is considered "Golden Data", and is used as a second redundancy to be compared with the Garlic Chain data.

It would not be sufficient to depend on this Google Cloud server alone to protect our data, however, this WORM DB is only 1 of 3 redundancies.

Let's look at the second redundancy, the Garlic Chain, after which we will explore the 3rd redundancy, the XA transaction.

The Garlic Chain

Let me first introduce some new vocabulary.

1. Banker's private key: much like a user has a private/ public key, each banker also has one, under the same conditions, that even the banker is not aware of what their private key is.

2. Hash: a string of what appear to be random letters and numbers. There are many types of hashes that are produced from different code, commonly we use sha 2 384, which takes, for example, a sentence, and then turns it into a hash.

3. ECDSA signature and verification: the elliptic curve digital signature algorithm is a piece of code that can take something, such as a hash, and provide a hash of its own, which we refer to as a signature. This signature can later be verified by the public key.

So, imagine a new account is created, a private key is created and kept secret, however, this private key is able to produce a public key. This public key can be viewed by anyone, it in no way gives any clue as to what the private key is. However, the public key can be used to verify a signature from a private key.

So, let's say I'm banker 1, and I want to claim that banker 2 has confirmed the transaction that I am processing as correct and true. Banker 1 needs banker 2 to sign off on this transaction. It's also important that banker 1 not change the transaction after banker 2 has already signed off on it.

So, banker 1 may hash the transaction data, have banker 2 sign it, which banker 2 does by taking that hash and then signing it with their super secret private key, and then handing that signature back to banker 1.

Through banker 2's public key, banker 1 or anyone for that matter, can run a simple piece of code to determine whether banker 2 actually signed off on that transaction, or if it were instead a forgery.

Now let's introduce a new piece of technology, the garlic chained DB:

Intro:

The Garlic Chained DB ensures data integrity and protects it from malicious attackers.

There are only two ways the data could be corrupted by a malicious attacker, one of which would be to make the garlic chained data invalid, or the other would be to change the entire garlic chained database.

The only way to make a valid, undetectable change in the DB, is to know the private key of the transactor, the private key of the transacting Banker, and the private keys of all the other bankers that have signed off on this transaction.

The Garlic Chain

Protects a WORM DB from being altered without becoming either invalid, or completely changing the entire db.

Each new transaction appended is followed by a "garlic" transaction.

This example table is wide, so I'm going to add it here in pieces:

ID	Type	transactor public key	recipient public key
...			
11	Transactor	gghei	idfknhp
12	SNH	null	null
13	Transactor	ewfiun	asdfe
14	SNH	null	null
15	Transactor	tyri823	lhirje348
16	SNH	null	null
17	Transactor	afwek4	asfl4u7f

anocoin amount	banker public key	other bankers public key
3.11	ji4823hgf	oiasj023u4, oasfo2389, cnve9238, cnisd332
null	ji4823hgf	oiasj023u4, oasfo2389, cnve9238, cnisd332
5.32	ji4823hgf	oiasj023u4, oasfo2389, cnve9238
null	ji4823hgf	oiasj023u4, oasfo2389, cnve9238
0.09	ji4823hgf	oiasj023u4, oasfo2389, cnve9238
null	ji4823hgf	oiasj023u4, oasfo2389, cnve9238
3.4	ji4823hgf	oiasj023u4, oasfo2389, cnve9238

hash	banker signature
jgkjwek(columns a-h, all of 9, columns f-g of 12)	sotnboegg
wef(columns a-g, and all of 13)	ilubiawef
awef (columns a-g, all of 11, columns f-g of 14)	iuh9
8sd (columns a-g, all of 15)	wefiun
lije4g (columns a-g, all of 13, columns f-g of 16)	hu8324
9dglr (columns a-g, all of 17)	wlefmn
afk2 (columns a-g, all of 15, columns f-g of 18)	ljaf97

banker signature	transactor signature	other bankers signatures
sotnboegg	mrn3293rj	asdfasdf , ...
ilubiawef	null	asdfasdf , ...
iuh9	iuh9hp9	asdfasdf , ...
wefiun	null	asdfasdf , ...
hu8324	eioj349	asdfasdf , ...
wlefmn	null	asdfasdf , ...
ljaf97	oj5kgjo	asdfasdf , ...

Date/Time submitted	Date/ Time processed
some date/ time	also some date/ time
....	.....
....	.....
....	.....
....	.....
....	.....
....	.....
....	.....

Let's look at ID 11.

This is a transaction that someone with the public key gghei has placed.

The recipient public key column lists the public key that the anocoins are being sent to.

They are sending 3.11 Anocoin.

The banker's public key is listed. This is a Google Cloud hardware KMS ECDSA p 384 public key.

The other banker's keys are also ECDSA p 384 public keys, this is supplied by each of the other bankers.

In Id 11, which is a transactor row, in the hash column, we append together columns a - g, all of row 9, and columns f-g of row 12 and sha 384 hash this and store the output in this column.

In column J we have the transacting Banker's signature, and in column K we have the transactor's signature.

In column L we have all the banker's signatures.

Row ID 12 is a SNH row, this row connects transactor row 11 and 13.

It's not possible to change one without the other.

On row 12 in the hash column has the output of the hash of columns a - g of row 12 and all of row 13.

All the same signatures are on row 12 as are on row 11, except for the transactors. This is because row 12 is created when row 13 is created, and the transactor is no longer available for signing.

Notice on row 11 we included all the public keys from row 12 in the hash. One issue that arises here is that not all bankers that were online when transactor 11 submitted their order are still online when transactor 13 submits their order. Let's demonstrate this. Let's assume that transactor 13 has one banker offline when their order is submitted, it will be the last other banker in the list.

This banker with public key: cnisd332, has gone offline.

So, let's consider a validity check from line 13 and backward.

So, a validity check in line 13, 12 and 11, in the logic, must calculate if there is a difference in the number of banker's that participated in signing. If so, then when comparing hashes, the logic must see if the hashes do match when adding the missing banker's keys.

Protection against malicious attacks

There are several potential attacks as listed below:

1. Intruder changes data.

2. Intruder changes data, and rehashes hashes in a transactor row, but does not know any private keys, therefore, the intruder changes the public keys to public keys that they have the private key to, so that they can rehash valid hashes and signatures. However, we'll demonstrate that in order to keep the entire table valid, they would have to change the entire table. This would be equivalent to just deleting the table and inserting an entirely new one.

3. Intruder knows the private keys of the transactor row, the transacting banker's private key, and all the other banker's private keys.

In this case the intruder could successfully change data and make it valid. However, they would still have to change the rest of the table going forward from where they made the change.

Short of knowing all the private keys involved, no amount of data can be changed without making substantial noticeable changes to the database.

As one redundancy of three, the garlic chain could operate as a stand alone WORM DB.

However, we do not consider it to be truly WORM, because it can be changed and/ or deleted by an intruder.

Validating Data

It is technically possible to validate the entire table before processing a transaction.

There are two types of validation.

1. Is the data correct? For example, if the public key of the receiver of one transacting row were to be changed, and only that, then this row would not match the hash of the row, which would show that this row is invalid.

2. The data is correct but the signatures do not match. For example, if after changing the transacting row's receiver's public key, the row were then re hashed, then the data would appear to be valid. However, when validating signatures, the intruder would be unable to have produced valid signatures, so this row would show as invalid.

However, validating the entire table for each transaction would become impractical as the DB grows.

But this is not necessary, as the information from the Public Ledger table must match the transactor's transactions on the garlic chain.

So, it's not necessary to validate the entire table before processing the transaction, it's only necessary to validate the transactors transactions and receives, as well as the receivers receives and transactions.

Each banker, in the background, has a continuous validator running. With 15 bankers, for example, the table can be divided up into 15 parts, and a continuous validation check can run parallel to processing transactions.

XA Processing

XA transactions are the third redundancy in the Anocoin system. Each transacting banker must get signatures from every other banker, that confirms their data matches exactly.

This prevents one banker from committing fraud, or even several working together, and it means that if a malicious attacker were to somehow successfully take over one banker's systems, they would also have to simultaneously take over every banker's system at the exact same time.

Order of transactions

When a transactor initiates the transaction, it's vital that they cannot transact another transaction, until this first one is completed. However, for the garlic chain, order is vital, as orders are chained together.

Therefore, the first step of a transaction is to add the transactor's public key to the processing order table.

When a banker adds a public key to the processing order table, it has effectively locked that public key from making another transaction before the first one is complete.

Only after all the steps of the transaction have been completed, will it then be added to the garlic chain.

And thus, without too many technical details, there we have the entire Anocoin system.

Anocoin-

developed by Cider