TWIST: Revolutionising Blockchain Accessibility

TWIST Developers
April 18, 2018

Note: TWIST is a work in progress and this project is development driven in nature. This paper will be continually updated and new versions will appear at https://twist.network. For comments and suggestions, contact us at contact@twist.network

1. Introduction

TWIST is a blockchain based decentralised platform and ecosystem which aims to increase the accessibility of the blockchain through innovative services and features targeted at casual cryptocurrency users and programmers interested in utilising blockchain technology. Features such as TWIST ID strive to simplify the payment process and make it more familiar and friendly to create and send transactions. TWIST DATA aims to provide a simple and clear interface to allow users and programmers to read and write encrypted data to the blockchain. TWIST API will provide a modern, language agnostic programming interface which can be utilised by developers to integrate their applications with the TWIST blockchain. Further features and services are planned and will be developed as the project progresses.

The ultimate goal for TWIST is to achieve widespread adoption and to create a thriving self-sustaining ecosystem where TWIST coins are used to pay for both blockchain based, and off-chain features, the fees of which are allocated as rewards to node owners for providing services and for stabilising and securing the network.

2. TWIST Coin

2.1. Specification

The TWIST coin is a Proof-of-Stake (PoS) crypto-currency based on the STRAT coin by Stratis. The initial supply of TWIST was 200,000,000 coins which were premined and predominantly distributed via airdrop. TWIST has a fixed target block-time of 30 seconds and a uniform block-reward of 20 TWIST per mined block. Given 2 blocks are mined every minute, with 1440 minutes in a day, and 365 days in a year, approximately (2 * 1440 * 365) * 20 = 21,024,000 new TWIST are mined every year, which represents an annual inflation rate of about 10.5%.

2.2. Distribution

Of the 200,000,000 TWIST coins that were premined, 20,000,000 (10%) were reserved as a development fund, 10,000,000 (5%) were reserved as a bounty fund, and the remaining 170,000,000 (85%) were airdropped to BitcoinTalk forum members. To be eligible to sign up to the airdrop, forum member accounts had to have an account that had existed for more than 2 weeks, with a post count of at least 15. 1938 accounts signed up to the airdrop and each account received an even share of the coins (170,000,000 / 1938) = 87719 coins per participant.

3. Encoding Auxiliary Data into Transactions

Blockchain based Twist services will rely on data being written to and retrieved from the TWIST core blockchain. A protocol has been defined to facilitate this.

3.1. Protocol Specification

Every TWIST transaction contains a non-zero number of output addresses. A TWIST output address is an identifier of 33-34 alphanumeric characters, beginning with the number 1 that represents a potential destination for a TWIST payment. Normally these addresses are displayed in their Base58 format; however each address is in essence just a 20 byte binary string. Since 20 bytes of arbitrary information can be stored in a single address we can use output addresses as simple and transparent ways of encoding data onto the blockchain. Simply convert the data to be included into valid Base58 addresses and use them as outputs in a transaction. Then after the transaction has been broadcast to the network, all clients have the ability to decode the addresses and retrieve the raw data in UTF-8 encoding.

The encoding to Base58 process is well documented online, so for brevity the steps are not outlined in this paper.

3.2. Advantages and Drawbacks

Encoding data into output addresses has the advantage of not needed a lot of complex code to extract the payload - simply iterate though each of the output addresses in a transaction and pass them through a Base58 decode function. This also means that existing tools such as block-explorers can be easily adapted to process and display the auxiliary data.

However, the extra addresses may clutter transactions and confuse casual users. Fortunately, block explorers and wallets can be easily programmed to hide the excess addresses to maintain a simplified user experience when necessary.

4. TWIST ID

A TWIST ID is a public identifier which can be assigned to any valid TWIST address. An ID is a 3 to 18 character string consisting only of alphanumeric, dash, and underscore characters (a-zA-Z0-9-_). An address can only have one ID assigned to it at a time, and IDs must be unique (case insensitive). The purposes of TWIST IDs are to simplify and familiarise the payment process for users, allowing them to send and receive transactions from recognisable recipient identifiers rather than the standard Base58 cryptographic addresses.

4.1. Encryption Keys

When a user registers a TWIST ID to an address, an Elliptic curve Diffie-Hellman (ECDH) key pair is generated and the private key is encrypted with the registering address's private key. The registration information, public key, and encrypted private key are all written to the blockchain in a single transaction and are publicly visible. Utilising asymmetric key cryptography facilitates utilities such as encrypted communication between two TWIST ID users.

4.1.1. Key Generation

When a user wishes to register a TWIST ID, an ECDH asymmetric key pair is securely computed using the secp256k1 curve. Both key components are then formatted as hex strings. For obvious reasons the private key component must first be encrypted before it is written publicly to the blockchain.

4.1.2. Key Encryption

Private keys for every TWIST ID need to be stored on the blockchain so they can be retrieved automatically by the owner of the ID. However, since the blockchain is a public ledger accessible to anyone with the tools to read it, the private key must first be encrypted in such a way that only the owner of its associated ID can decrypt it.

Ownership of a TWIST ID is defined as having possession of the private key of the address the ID was registered to. Therefore, by encrypting the ID private key with the registering address private key, only the owner of the ID is able to decrypt it.

4.1.3. Encryption Procedure

A 32 byte encryption key is derived using a password-based key derivation function with the address private key as the password, address as the salt, and a SHA512 digest algorithm with 10000 iterations. A 16 byte initialisation vector is also derived using the same approach, except 5000 iterations of the digest algorithm is used. These derivations are deterministic in nature – the same result will always be derived for a given password and salt.

Industry standard 256-bit symmetric encryption (AES-256 in Cipher Block Chaining mode) is then used, utilising the above key and initialisation vector, to securely encrypt the ID private key. Finally, the encrypted key is converted to a Base64 string, ready to be written on to the blockchain.

Deriving an encryption key rather than using the address private key directly to encrypt data has a rather arcane advantage of allowing for ID data to be encrypted/decrypted, with the knowledge that if the encryption key is compromised, funds stored in the associated address are still secure, since the derivation function is one-way.

4.2. Registration Procedure

A TWIST ID can be registered by broadcasting a transaction to the TWIST blockchain with the following properties:

1. All inputs are Unsigned Transaction Outputs (UXTO) spendable to the registering address – this ensures that ID registration transactions have provable ownership to the registering address, since the registering address's private key is required to sign the transaction.
2. A registration fee (50 TWIST at the time of writing) is divided and sent to a series of 17 data-encoded addresses. The fee acts as a deflationary measure for the TWIST currency since the coins are in effect being burned. The fee also acts to deter individuals from registering a selfish amount of IDs and thus restricting the IDs availability to other users.
3. Exactly 17 data-encoded addresses in a determined order. The first address is fixed and acts as an indicator to let the network know to examine this transaction for a possible ID registration. The second address, when decoded from Base58 contains the ID to be registered. The following 4 addresses contain the public ECDH key which was generated by the user before registering. The remaining 11 addresses contain the encrypted private EDCH key.

4.3. Detecting ID Registrations

Valid TWIST ID registrations are discovered on the network by the following algorithm:

```
1.  procedure detect_id_registrations()
2.    for block in blocks do
3.      for tx in block.txs do
4.        for vout in tx.vouts do
5.          if vout.address is ID_REG_ADDRESS then
6.            startIndex = tx.vouts.index_of(vout);
7.            confirm_id_registration(tx.vins, tx.vouts, startIndex);
8.            break;
9.          end
10.       end
11.     end
12.   end
13. end
14.
15. procedure confirm_id_registration(vins, vouts, startIndex)
16.   paid = 0;
17.   id, pubkey, privkey, registrants;
18.   for i = startIndex; i < vouts.length; i++ do
19.     paid += vouts[i].value;
20.     if i is startIndex then
21.       continue;
22.     else if i is startIndex + 1 then
23.       id = base_58_decode(vouts[i].address);
24.     else if i <= startIndex + 5 then
25.       pubkey += base_58_decode(vouts[i].address);
26.     else
27.       privkey += base_58_decode(vouts[i].address);
28.   end
29.   for vin in vins do
30.     for a in vin.addresses do
31.       registrants.add(a);
32.     end
33.   end
34.   if isValid(id) & isValid(pubkey) & isValid(privkey) &
35.      id ∉ idSet & pubkey ∉ pubkeySet & privkey ∉ privkeySet &
36.      |registrants| is 1 &
37.      registrants ⊄ registrantSet &
38.      paid >= ID_REG_FEE then
39.     idSet.add(id);
40.     pubkeySet.add(pubkey);
41.     privkeySet.add(privkey);
42.     registrantSet.add(registrants);
43.   end
44. end
```

Figure 1: TWIST ID registration discovery algorithm

Remarks on Figure 1:

1-13: Every transaction in every block on the TWIST core blockchain is scanned to determine whether it includes an output to the TWIST ID registration flag address. If such a transaction is found, then it is inspected to ascertain whether it contains a valid ID registration.

18-28: Each output in the transaction from the flag address onwards is parsed in an attempt to retrieve the ID, public key, and encrypted private key of the registration. The value of each output is cumulatively totalled to confirm whether the fee has been paid.

29-33: Each input address to the transaction is added to a set of registrant addresses.

34: For a registration to be valid the parsed ID, public key, and encrypted private key data must be in the correct format. The isValid() functions check the length of the fields are in the expected range and that the fields do not contain any illegal characters.

35: idSet, pubkeySet, privkeySet are global sets populated by running the algorithm in a chronological order starting from the first block. After every successful registration, the ID, pubkey, and privkey for that registration is added to its respective set. For a registration to be valid, the ID, pubkey, and privkey must not already be in use (not already a member of their respective set).

36: The registrants set must be a singleton set since an ID can only be registered to a single address and there must be no ambiguity in the address being registered. Registrations of an ID to an address require all inputs to the transaction to originate from the registering address, which ensures that the owner owns the private key to the address, since it is necessary to sign the UXTO.

37: The registrants set must not be a subset of the global registrantSet (the address must not already have an ID registered to it).

38: The total value sent to the data-encoded (burn) addresses must be greater than or equal to the ID registration fee. This requires that a determined amount of coins are burned for a registration to be successful.

39-42: Since the registration is successful, add its elements to their respective sets.

4.4. TWIST ID Transactions

An owner of a TWIST ID may send a transaction to another TWIST ID owner such that the recipient is able to see the ID of the sender, even if none of the coins spent in the transaction originated from the sender ID's registrant address. These transactions may also contain encrypted messages, decipherable only by the owners of the involved IDs. These transactions are syntactically identical to regular TWIST transactions and they are also publicly broadcast on the core blockchain.

The auxiliary data to facilitate these transactions is encoded using the same protocol as previously specified in the paper (i.e. the data is encoded into Base58 output addresses).

4.4.1. Transaction Structure

A TWIST ID transaction has the following output structure:

1. A recipient address: This is the address the recipient's ID is registered to. The value out for this address is the amount of coins being sent from the sender to the recipient.
2. [*Optional*] A change address: For returning excess coins (sum of all input values – sum of all output values in the transaction) back to the sender.
3. A flag address. This is a fixed predetermined address which acts as an indicator to the TWIST network to make it aware that this transaction may be a TWIST ID transaction, and to inspect it further. Different flag addresses are used depending on whether the transaction contains a message. This provides an easy way for the client to distinguish whether the transaction contains a message without needing to perform any decryption.
4. Sender ID address: This address contains a Base58 encoding of the sender's TWIST ID, and is necessary for the recipient to know the ID of the sender.
5. Validation/Message addresses: Each TWIST ID transaction contains a validation string for security purposes. If the transaction does not contain a message, then the validation string is encrypted and encoded into a series of addresses. If the transaction contains a message, then the validation string is prepended to the message before being encrypted and encoded into the addresses.

4.4.2. Fees

As with TWIST ID registrations, a fee is required for TWIST ID transactions to be considered valid. At the time of writing, the fees are 1 TWIST for a standard TWIST ID transaction (recipient can see sender ID), and 20 TWIST for transactions including a message. These fees are completely arbitrary but provide a deflationary mechanism to the coin. A small fee is a necessity when data is being encoded into output addresses, since a non-zero amount needs to be sent to an encoded address for it to be contained in a transaction.

4.4.3. Encryption Procedure

Messages contained in TWIST ID transactions are encrypted in such a way that they only decipherable to the owners of the sender and recipient IDs in the transaction. This is facilitated by utilising the ECDH key exchange protocol as described earlier.

Messages are encrypted by first computing a shared secret key from the sender's ID private key, and the recipient's ID public key. The message is then encrypted with symmetric AES-256 CBC encryption using the shared secret key as the encryption key.

Decrypting the message follows the same procedure, except the shared secret is computed using the recipient's ID private key, and the sender's ID public key.

4.5. TWIST ID Transaction Spoofing

In each TWIST ID transaction, the sender ID is sent essentially in plaintext (encoded in the sender ID address).

Theoretically, a transaction could be spoofed to appear to be sent from any ID – simply replace the sender ID address with an address containing any encoded ID.

4.5.1. A Solution

To prevent transaction spoofing, each TWIST ID transaction is required to contain an encrypted validation string. The string is simply an arbitrary non-Base64 character (i.e. $) followed by a randomly generated, Base64 string of 18 characters, followed by the same non-Base64 character. If the transaction contains a message, then the validation string is prepended to the raw message content before encryption. Else the validation string is simply encrypted using the same procedure as for messages.

When a client receives a transaction, an attempt to decrypt the validation string is made using the sender's ID public key and the recipient's ID private key. If the decryption fails, or the decrypted text does not contain a validation string in the expected format, or the validation string has been seen before in a previous transaction, then we consider the transaction invalid and it is ignored.

This validation mechanism ensures that the sender of the transaction is the owner of the sender ID, since knowledge of the sender's ID private key is required to compute the same shared secret that will be computed on the recipient's side.

4.6. Proof of Burn

At present, coins sent to data-encoded addresses are considered to be satisfactorily burned since an infeasible amount of computing power would be required to generate the corresponding keys for the addresses, which are generated from input data.

The flag addresses are the only predetermined addresses present in TWIST ID transactions, so there may be concerns that the development team own the corresponding keys to these addresses and so can spend the coins being sent to them. To assure the community this is not the case, the addresses, when decoded from Base58, contain descriptors of their use (e.g. "TWIST ID REG"). As we could not possibly have access to the computing power necessary to cryptographically generate the keys for these addresses, the community can be satisfied that coins sent there are being burnt.

Ideally, in future, coins will be provably burned possibly via utilising the OP_RETURN script opcode.

5. TWIST DATA

The internet is undergoing a fundamental shift away from centralised services and towards decentralised open ones. The popularity and hype surrounding blockchain technology is well deserved, with the success of Bitcoin, Ethereum, and other blockchain networks proving the utility and value of decentralised, distributed ledgers. People are becoming aware of the benefits to using decentralised systems with no single point of failure.

TWIST DATA defines a simple yet robust protocol for writing small pieces of information onto the TWIST blockchain. The data is safe and secure, being maintained by a network of hundreds of computers around the world, with no single points of failure and no possibility of data losses. TWIST DATA isn't intended as a means to store large files, but rather as a tool to give users an easy way of securing and storing small pieces of information (e.g. passwords, contact information, etc.) on a completely decentralised distributed ledger with no central point of failure.

5.1. Protocol

As with TWIST ID, the auxiliary data is encoded into TWIST addresses and written to the blockchain in the form of outputs in a transaction. Two types of data transactions are defined: private and shareable. In private transactions, the data payload is intended to only be decrypted by the owner. Shareable transactions provide the creator with a unique key that can be shared with others to allow them to decrypt and access the data.

5.2. Private Data Transactions

5.2.1. Encryption Protocol

The user is given the option to decide which of their addresses they would like to encrypt the data with. A 32-byte encryption key is then derived using a password-based key derivation function with the encryption address private key as the password, encryption address as the salt, and a SHA512 digest algorithm with 15000 iterations (iteration numbers are arbitrary but are chosen to vary between different TWIST services to enhance security). A 16-byte initialisation vector is also derived using the same approach. Industry standard 256-bit symmetric encryption (AES-256 in Cipher Block Chaining mode) is then used, utilising the above key and initialisation vector, to securely encrypt the data.

5.2.2. Transaction Structure for Private Data

A TWIST DATA private transaction has the following output structure:

1. [*Optional*] A change address. For returning excess coins (sum of all input values – sum of all output values in the transaction) back to the sender.
2. A flag address. This is a fixed predetermined address which acts as an indicator to the TWIST network to make it aware that this transaction may be a TWIST DATA private transaction, and to inspect it further. Different flag addresses are used depending on whether the transaction type is private or shareable, allowing clients to parse the transaction correctly.
3. Encryption address private key hash. This is the first 20 characters of a hash of the private key used to encrypt the data. This allows clients to check whether the encryption address is present in the user's wallet (and so they own and can decrypt the data).
4. Validation/Data addresses. These addresses contain the encrypted data and a validation string to protect against transaction spoofing.

5.3. Shared Data Transactions

5.3.1. Encryption Key Specification

For each shared data transaction, a secret key needs to be created which is shareable to others to allow them access to the data. As such, the key should have the following properties:

- Unique – The key must only decrypt the data in the transaction it was created for.
- Secure – It should not be feasible to guess or brute-force attack the key.
- Independent – Knowledge of the key should not expose any sensitive information about the owner. e.g. the key should not be a sub-section or direct derivative of the owner's private key
- Retrievable – The data owner should not need to remember the encryption key for their data. The key for a shared data transaction should be easily re-creatable or retrievable for the owner of the data.

5.3.2. Encryption Key Generation

To create a sharable data encryption key, a Base64 string with a length of 18 characters is randomly generated. This fulfils the criteria of being unique, secure, and independent. It is also not impractically lengthy, which is something worth considering if the intention is for it to be shared. To make the key retrievable, it is encrypted with the user's private key and written to the blockchain in a transaction, allowing for automated retrieval in future.

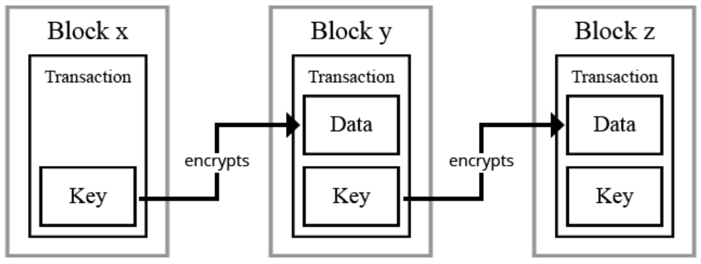

Figure 2: The chaining of encryption keys used in TWIST DATA shared data transactions

5.3.3. Encryption Protocol

The user designates which of their addresses they wish to set as the owner of the data (the encryption address). The private key of this address is used to deterministically derive a key which is then used to encrypt the shared key.

If the encryption address is not the owner of any existing shared data transactions then the encrypted shared key is bundled into a transaction containing a special flag address and an address containing the encryption address private key hash. The transaction, shown in Block x in Figure 2, is pushed to the network and after receiving at least one confirmation, the shared key is used to encrypt the data and another shared key is generated and encrypted. The new encrypted shared key and the data encrypted with the previous shared key are then bundled into a transaction which is pushed to the network. This is the TWIST DATA transaction, as shown in Block y in Figure 2.

If the encryption address is the owner of any existing shared data transactions, then the shared key is retrieved from the most recent valid transaction and used to the encrypt the new data. A new shared key is then generated and encrypted and bundled along with the encrypted data into a TWIST DATA transaction, shown in Block z in Figure 2, which is then pushed to the network.

By using this method, data owners are able to retrieve the keys for all of their shared data by following the chain of all their shared data transactions, starting from the initial transaction containing just the key. This technique does however enforce a one per block limit on shared data transactions for a given encryption address.

5.3.4. Transaction Structure for Shareable Data

A TWIST DATA shareable transaction has the following output structure:

1. [*Optional*] A change address.
2. A flag address, specific to TWIST DATA shareable transactions.
3. Encryption address private key hash.
4. Key/Validation addresses. These addresses contain an encrypted unique key for which a following shareable data transaction, with the same encryption address, will use to encrypt its data. The key also doubles as a validation string to determine the transaction has not been spoofed or duplicated.
5. Data addresses. These addresses contain the encrypted data.

5.4. TWIST DATA Transaction Spoofing

Similar to as with TWIST ID transactions, by manipulating the encryption private key hash address in a TWIST data transaction, a transaction can be made to appear to be owned by another address. While this does not present any security concerns, it could be used to spam a user and make their client display transactions they did not create. To ensure a transaction is legitimate, validation strings are included in each transaction. In shared data transactions, the validation string is simply the shared key. In private data transactions, the validation string is prepended to the data before encryption occurs.

5.5. Fees

All TWIST DATA transactions require a fee to be considered valid by the network. Fees are subject to change but at the time of writing, we propose a fee of 20 TWIST should be required per 500 characters of data.

6. TWIST API

As demand for blockchain based services increases, it is anticipated that an increasing number of applications both on a hobbyist and industrial scale will strive to utilise blockchain technology. TWIST API will provide developers with a simple yet powerful interface to access TWIST services and write and read information to and from the TWIST blockchain. The goal is to facilitate a simple way for developers to utilise the TWIST blockchain in their applications, thus incentivising usage and adoption of TWIST currency and platform.

6.1. API Functions

The API will abstract the underlying implementation for all TWIST services (e.g. ID, DATA, etc.), and only expose the actions the developer needs. Some (but by no means all) of the methods intended to be offered by the first edition of the API:

- Authenticate a TWIST ID - i.e. confirm that the user owns the private key of the address for which the ID is registered
- Register a TWIST ID
- Send a TWIST ID Transaction
- Write data to the TWIST blockchain
- Read data from the TWIST blockchain

The scope of the functions provided by the API will continue to grow as the TWIST platform evolves and more features and services are developed.

6.2. Use Cases

By exposing the ability to authenticate and register TWIST IDs, developers can integrate TWIST ID with their applications (for example, ‘Login via TWIST ID’ functionality). However, we expect TWIST DATA functionality (i.e. writing/reading data to/from the blockchain) to be the most utilised.

6.2.1. Example Use Case 1

A video game developer may choose to utilise the TWIST blockchain to keep a record of each player’s score in the game. Each player first authenticates with their TWIST ID, and then upon completing the game they may wish to submit their score to the TWIST blockchain. The data is written to the blockchain and a leader board is constructed by reading all scores from the blockchain.

Using a blockchain to store information, such as player scores, may be preferable over a database since the data is secured with no need for backups, there is zero network downtime, the data has complete immutability, and the storage is fully decentralised with no central point of failure.

6.2.2. Example Use Case 2

A university assignment submission system may wish to utilise the TWIST blockchain to verify that a student has submitted their assignment on time. When a student uploads their assignment files, each file is passed through a cryptographic hashing function and the resulting hash is written to the TWIST blockchain along with the student’s ID. This provides a decentralised and permanent record that the file existed at the time of the transaction network confirmation, taking advantage of the distributed irreversibility of blockchain technology. Students can feel safe in the knowledge that a decentralised, distributed ledger can provide irrefutable proof that their assignment existed and was submitted to the school’s servers at a given time.

6.3. Specification

The API is designed to be language agnostic, allowing programmers working in any language to make requests and receive responses through standard protocols and in common formats (e.g. HTTP/JSON). Initially, the API is intended to be built as an interface to run on top of the TWIST Toolbox desktop application. Developers running the Toolbox could consequently enable and configure the API to receive requests from external IP’s, allowing them to run an API server which can be used to serve their applications.

7. TWIST Nodes

Services such as TWIST ID, rely on writing and reading data to and from the blockchain. However, for certain purposes blockchain based data access becomes impractical as block-size limits and block-times hinder streamlined functionality.
A peer-to-peer messaging application for example would not be well suited to operating solely on a blockchain, as there would be a considerable delay between sending a message and the recipient receiving it, as well as limits to the maximum size of the message that can be sent in a single transaction. TWIST Nodes are introduced as a facilitator to provide support for off-chain functionality such as peer-to-peer messaging and data storage, allowing the TWIST platform be as comprehensive and feature rich as possible.

7.1. Requirements

Anyone will be able to run a TWIST Node provided they possess a minimum amount of TWIST (exact amount yet to be confirmed) to be used as the deposit for the node. Users wishing to run a node will need to consolidate the deposit amount into a single address and then make a registration request on the blockchain. Their node will be registered and will remain active so long as the balance of the address does not drop below the deposit amount. Nodes will be free to register and registrant’s coins are never locked and are always available to them should they wish to break their node. Requiring a minimum deposit amount ensures that each node owner has a significant stake in the TWIST ecosystem, and an incentive to keep the network stable and secure.

7.2. Functionality

Nodes will facilitate a network parallel to the TWIST core blockchain which will enable services such as TWIST CHAT to operate by acting as intermediary servers between the communicating parties. TWIST ID users will be able to send instant encrypted messages to each other relying on AES symmetric encryption and Elliptic Curve Cryptography, without the need for these messages to be stored on the blockchain. The encrypted messages will first be sent to and stored on nodes, and then delivered to the recipient when they are online and able to receive them. Blockchain based validation (e.g. upon sending a message the sender writes the hash of a message payload to the blockchain, which the recipient then uses to validate the integrity of the message they receive) could also potentially be utilised to assure the communicating parties that there were no errors or tampering involved in the transmission of their messages.

In conjunction with end-to-end encrypted messaging, nodes will also provide extensions to the TWIST DATA service. We envision a network where large data payloads and files can be encrypted and stored off-chain on nodes, consequently reducing the load on the core blockchain, while still maintaining a distributed and decentralised data storage network.

Nodes may also operate as API servers, exposing an interface to the core blockchain for application developers to connect to and utilise. As a result, developing applications upon the TWIST blockchain may seem more appealing and accessible as developers will not need to spend time or money setting up their own API servers to serve their applications.

7.3. Incentives

As a reward for running a TWIST Node, owners will earn fees in TWIST depending on usage statistics (i.e. messages served, disk space utilised, etc.) along with activity statistics such as node uptime and average connection speed. Nodes which are utilised more heavily or have longer, more consistent uptimes will earn a greater distribution of fees, incentivising a fast, stable, and secure network. It is currently envisioned that fees will be collected from all TWIST services and distributed on a weekly basis amongst all node owners. A large proportion of the coins presently being burned in services such as TWIST ID, will instead be collected and used to pay node operators. In future, a reduction in the staking reward may possibly be introduced or the reward potentially eliminated altogether. This would limit inflation, making the coin scarcer and helping the TWIST ecosystem to transition into a sustainable fee based model, where node owners have a strong incentive to secure the network and provide services to TWIST users.

8. Future Developments

As the TWIST project evolves, documentation for more features and services will be added to this document, and additional detail will be provided for existing features as development progresses.