SkyeKiwi: A Decentralized Secret Sharing Network Based on Polkadot

Draft Version v0.2

Author: Song Zhou (song.zhou@skye.kiwi)

July 11, 2021

1. Introduction

Secret sharing refers to cryptographic methods of distributing secret messages to groups of participants. Data ownership is at the very core design of the SkyeKiwi Network and is an act of having legal rights over a single piece of data element so that the rightful owner implements the use, acquisition, and distribution policy. SkyeKiwi Protocol is a decentralized secret sharing protocol that is built to combine economic incentives, data ownership, and programmable interfaces to distribute arbitrary types of secrets through a decentralized network. It takes the form of a Substrate based blockchain of secure execution runtime enabled nodes and smart contracts model.

2. Background

In this section, we will give an overview of the cryptographic methods we used and how blockchain smart contracts operate in the mainstream paradigm, along with a client-side protocol that the SkyeKiwi team has built.

2.1 Basic Cryptographic

The two basic, most implemented and well-researched types of cryptography schemas are the symmetric and elliptic-curve based asymmetric encryptions. For an symmetric encryption schema, we defined it as(E_s, D_s) for **asymmetric encryption method E_s** and the **symmetric decryption method D_S**. An elliptic-curve based asymmetric encryption schema is defined as a triple:(G_a, E_s, D_s), for **a key generation algorithm G_a, an asymmetric encryption method E_a** and **the decryption function D_s**.

- E_s works as $c \leftarrow E_s(m, sk)$, by taking a message and a secret key to generate a cipher
- D_s works as $m \leftarrow D_s(c, sk)$, by taking a cipher and secret key to recover the original message
- G_a is a probabilistic function and works as $(pk, sk) \xleftarrow{R} G_a()$ that generates a key pair
- E_a is also a probabilistic function and works as $c \xleftarrow{R} E_a(pk, m)$
- D_a is deterministic and works as $m \leftarrow D_a(sk, c)$ where m is either the original message or a special **reject** message.

2.2 Threshold Secret Sharing

Threshold secret sharing is a cryptographic algorithm that is defined as a triple (G, R) and

- G is a probabilistic function that generates the shares in such way that $[s_1, s_2 ... s_n] \xleftarrow{R} G(m, n, t)$, where m is the secret message, n defines the number of shares to be generated and t is the threshold needed to recover the message
- R recovers the secret by $m \leftarrow R([s_1, s_2 ....])$ and m is either the original message or a wrong message if the shares supplies are corrupted or not meeting the threshold.

The most commonly used schema is proposed by Adi Shamir and is called the Shamir Secret Sharing (SSS). In SSS, the dealer construct a polynomial: $q(x) = m + \sum_{i=1}^{t-1} a_i x^i$ and the dealer will securely share each participants P_i with $s_i = q(i)$. For parties $P_{i_1} ....P_{i_t}$, their $s_1 ...s_t$ are sufficient to recover the secret according to the Lagrange Interpolation Formula.

$$q(x) == \sum_{j=1}^{t} y_i \prod_{k!=j} \frac{x - x_k}{x^j - x^k}$$

SSS is a simple and elegant way of sharing secrets but there are a couple of significant drawbacks of the original SSS.

1. Single point of failure: function G needs to be called on one device and later reconstructed by calling R on a single device. If the device is compromised at either of these steps, the secret is at risk of being compromised as well.
2. No Share Revocation: if the structure of the sharing needs to be changed, all shares need to be brought together and reconstruct the secret, generate a new array of shares, distribute and replace the old share with the new one.
3. Inability to verify share integrity: for holders of the shares, there is no way for them to verify that their shares are not corrupted.

To solve these drawbacks, a combination of techniques was implemented by the SkyeKiwi Protocol. As for cryptography, the SkyeKiwi Protocol adopts Feldman's scheme to generate verifiable shares.

When a dealer of the secret use the SSS to share m, they construct the polynomial $q(x) = m + \sum_{i=1}^{t-1} a_i x^i$, In addition to sending the shares to the recipients P_i. Whenever P_i receives a share s_j, they check if $g_j^s = \prod_{i=0}^{t-1} y_i^{j^i}$. The verifiable secret sharing schema (VSS) is a triple as $\{Shares(m,t,n), Recover([s1, s2...st]), Verify\}$

2.3 BLS Signature

The BLS Signature Schema is a simple elliptic-curve based digital signature algorithm. It's a standardized schema with implementations from leading engineering teams and is included in the specifications of ETH2.0 and is also included in the SkyeKiwi Protocol as an alternative to ECDSA in various parts of the design.

The BLS Signature Schema relies on pairing elliptic curves, which has some sweet properties, for points P, Q, R :

$$e(P, Q+R) = e(P,Q) \times e(P,R)$$
$$e(P+S, Q) = e(P,Q) \times e(S,Q)$$
$$e(xP, Q) = e(P+P...+P, Q) = e(P,Q)^x = e(P, xQ)$$
$$e(\sum^n P_i, Q) = \prod^n e(Pi, Q)$$

Similar to other elliptic-curves algorithm, our public key $P = sk \times G$, where G is a generator point, different from other digitize signature algorithm, BLS utilize a different types of hash function that hashes a point to curve. Signature is constructed as $S = sk \times H(m)$. A valid signature satisfies $e(P, H(m)) = e(G, S)$, because $e(P, H(m)) = e(sk \times G, H(m)) = e(G, sk \times H(m)) = e(G, S)$.

Aggregating signature is also pretty straightforward: for $S_1, S_2 ...S_n$, an aggregated signature is as simple as $S = \sum_{i=1}^{n} S_i$, to verify the signature: $e(G, S) = \prod^n e(P_i, H(m_i))$.

2.4 Smart Contract Execution Environment

Typical smart contracts are state machines that taking a state transition function f that takes input parameters and the last state $s_n - 1$ to produce the latest state s_n. It can be represented by

$$s_n = f(s_{n-1}, ...parameters)$$

For public blockchains, the states, the state transition functions are all public. We will come back to this later at the "Secret State Transition & Consensus" section.

2.5 TEE & Intel SGX & Remote Attestation

Trust Execution Environment (TEE) is a special security enclave area in some processors that provides isolated execution, code integration, and state confidentiality. There are many TEE implementations including the Intel chip-based SGX and the open-source framework-based TrustZone by ARM, while Intel SGX is more widely used.

Remote Attestation is a protocol that ensures the execution finished as expected inside an enclave. An Attestation quote based on information of hardware, firmware, and executed code inside the enclave will be generated and signed by the hardware, and it will be sent to the Intel Remote Attestation Service.

2.6 SkyeKiwi Client-Side Protocol & Secrets Sealing/Unsealing

In short, the SkyeKiwi Protocol can be represented by the following schema:

- Given ES representing an encryption schema, that contains the number of shares, the number of thresholds, and a list of the public keys of the recipients.
- Also given a byte stream $Source$ of size N, with a chunk size(i.e highwater mark CK_{size}), a function to consume the stream $Read(source)$
- An IPFS uploading function $CID_i \leftarrow Upload(content)$
- A client will generate a random 32 bytes sealing key denoted as slk in distinguish with the secret key in a public-key encryption schema sk.

$$\begin{gathered}
Read(Source, CK_{size}) = [CK_1, CK_2 ... CK_i] \\
E_s([CK_1, CK_2 ... CK_i], slk) = [ECK_1, ECK_2 ... ECK_i] \\
Upload([ECK_1, ECK_2 ... ECK_i]) = [CID_1, CID_2 ... CID_i] \\
Upload(encode([CID_1, CID_2 ... CID_i])) = CID_{list} \\
G(slk + CID_{list}, ES.shares, ES.threshold) = [s_1, s_2 ... s_n] \\
E_a([s_1, s_2 ... s_i], [ES.pk_1, ES.pk_2 ... ES.pk_i]) = [cs_1, cs_2 ... cs_i] \\
Upload(encode([cs_1, cs_2 ... cs_i])) = CID_{result}
\end{gathered}$$

While recipients with the sk_i associate with a pk_i and the CID_{result} is capable of recovering the original byte stream by reversing the process described above.

> One thing to note is that the pk_i of recipients are never recorded in the resulting metadata because that won't be necessary. The secret recipient will try to decrypt all $[cs_1, cs_2 ... cs_i]$ with sk_i and for all those successfully decrypt, the recipient will try to recover the decrypted $[... s_i]$ over the TSS recover function $R([... s_i])$.

Furthermore, the slk is the same size as a private key of Curve25519 and it is a shared secret between all recipients. Therefore, the slk is also multi-functional and acts as a shared private key. Therefore, a simple decentralized contract signature platform can be easily constructed:

- Given a file as $Source$ and a sufficient chunk size to include the whole file in one chunk.
- Given a list pk of contract receivers as $[pk_1, pk_2 ... pk_i]$, and compose into an encryption schema ES
- Given a smart contract assign a number to a contract as $ID_{contract}$

The initiation process can be expressed as the following pseudo-code:

```
// @pk_contract is a compressed 32bytes public key derived from the %slk%
// Note: for convience, @pk_contract might be encoded as a standard ETH address
// @CID_result is the output of the above cryptographic processing

const [pk_contract, CID_result] = await SkyeKiwi.upstream(Source, ES);
const contract_id = await smartContract.createContract ({
  cid: CID_result,
  public_key: pk_contract
});
```

When a recipient tries to generate a proof-of-agreement to a contract, they will call the following function on the smart contract, with pesedo-code as:

```
function generateProofOfAgreement ( signature, contract_id ) {
  Hash content_hash = generateHash ( msg.sender, contract_id );
  address signer = ECRECOVER ( signature, content_hash );

  // stores the public key derived from the sealing key %slk%
  require(signer == public_key[contract_id], "fake signature");

  // mint the origin of the TX an NFT w/ content of the contract_id hash
  mintNFT ( msg.sender, hash(contract_id) )
}
```

So far, we have discussed a client-side implementation of the SkyeKiwi Protocol to transmit secrets over public blockchain & storage networks and a realistic use case to use it for contract signature.

3. The SkyeKiwi Nodes

The following section will describe a blockchain network built with a system-level implementation of the SkyeKiwi Protocol. At the very core, the SkyeKiwi Network is a network of node operators with TEE-enabled hardware that operates a series of encrypted databases.

3.1 Node Types

There are three types of blockchain nodes in the SkyeKiwi Network:

- **Users**: users can deploy or interact with the vaults via Blockchain. They can also verify and challenge the cryptographic components of the vaults.
- **Validators**: run the NPoS consensus engine as Substrate based blockchain nodes and are responsible for the basic consensus of the network. They will also run all non-secret related computations of the network. Validators can register

themselves as Secret Keeper candidates.

- **Secret Keepers**: validators who also run specialized TEE software that is authorized to write and read the vaults. **Secret Keepers** are the most important members of the network and they are required to stake a large amount of the SkyeKiwi token (**SKW**). They are rewarded for processing vaults requests and slashed in case of misbehavior.

3.2 Decentralized Vaults

The vaults are the encrypted state of an application in form of a JSON file or key-value database. A vault with ID $Vault_i$ will be stored encrypted according to the SkyeKiwi Protocol. The SkyeKiwi Network will accept transactions from the open network for requests to interact with vaults.

Vaults can be small or large. The SkyeKiwi Protocol is designed and implemented to reduce the storage overhead as a result of encryptions. However, Secret Keeper nodes can still offload vaults that they considered less frequently accessed to the storage network at their interest, in scarifies for speed. Therefore, on each Secret Keeper Node, there will be secrets that are cached (i.e. stored locally) and not cached (i.e. stored on a storage network). By default, we recommend nodes to use the Crust Network for cheap and temporary storage but the nodes can also configure themselves to store secrets on Arweave for permanent storage or any centralized blob storage provider. The risk of in-accessibility of the storage network at events of network faulty should be taken by the nodes.

Vaults are created by deploying smart contracts through a "secret-contract-pallet", with a specialized smart contract language. (i.e. a modified version of the ink! environment), where there will be a dedicated section of the storage declaration marked as “secret” and messages marked as “secret message”. When these secret storage and secret messages are called, they will be redirected to be only handled by Secret Keepers, while other smart contract requests will be handle by either regular validators or Secret Keepers.

3.2 Limited Resource & Secret Slot Auction

Handling secrets is complicated and computing & networking intensive. Therefore, we are limiting the number of secrets available to the network. Vault slots are leased and they will be initially registration-based while requiring a small amount of staking. For common good secrets, the on-chain treasury can assign secret slots on an application basis. As secret slots are running low, an auction mechanism similar to the Polkadot parachain auction will be in place. We will discuss the Secret Slot auction in more detail in Section 5.4.3.

When an address successfully obtained a vault slot, they will be assigned a $SecretSlotId$ that will expire on par with the lease terms. The valid address that holds the $SecretSlotId$ can be an external address or a smart contract address and is considered the owner of the secret and is entitled to the future value generated by the secret as detailed in the Economic section.

3.3 Secret Keeper Node registration

Secret Keeper nodes need to be first registered to be validators or validators candidates. Then, they will generate a public key pair (pk, sk) and generate a remote Attestation quote q. After submitting and signed by the Remote Attestation Service with signed quotes q_{signed}, the blockchain will verify and accept the registration into a Secret Keeper registry. After each era, secret keepers will be elected or re-elected from the candidate list stored in the registry.

3.4 Secret Initialization & Indexing

An address that is authorized to use a $SecretSlotId$ can initiate a vault by creating an initial state of the Vault with the SkyeKiwi client-side protocol and associate a secret smart contract as a collection of state transition functions associated with the secret. The ES of such encrypted with public keys of all current the Secret Keepers from the registry through the SkyeKiwi Protocol.

Upon successful initialization of a secret, a record of the initial CID_{result} will be recorded in the secret registry so that the elected Secret Keepers can recover the initial state locally on their environment. At the same time, the block height, which the initialization of the secret will be recorded as the current **checkpoint** of the secret. Authorized Secret Keepers will fetch the initial state of the secret, un-seal the secret, and re-encrypt the secret with a uniquely generated sealing key of the node sk_i that is kept within the TEE enclave. The sealing key of nodes sk_i is unique on any Secret Keeper Nodes. Upon successful initialization, the secret keeper node will write a record to the registry, so that other nodes in the network know that the Secret Keeper has access to the secret.

The SkyeKiwi Protocol stores secret into two parts: the encrypted chunks $[ECK_1, ECK_2...ECK_i]$ and an encrypted metadata of the sealing key and the CID list: $[CID_1, CID_2...CID_i]$. Therefore, while key rotations come in two forms (detailed in the Secret Keeper Keys section), secrets re-indexing also comes with two flavors for different types of key rotations:

- Recipients re-indexing: keep the currently public sealing key of the secret but update the encryption schema to match the new list of recipients. Upon finishing, write the new CID_{result} to the registry.
- Full re-indexing: generate a new public sealing key and run the whole encryption of the secret with the SkyeKiwi Protocol. Such action will take the most recent elected Secret Keepers' public identity for the encryption schema. Upon finishing the process, it will write the new CID_{result} to the registry and update the check-point to reflect the block height of the most recent execution queue.(detailed in the Secret State Transition & Consensus section).

3.5 Secret State Transition & Consensus

For requests $call_i(Vault_i, f(parameters), origin)$ passed to the Secret Keepers. Calls will be recorded in an append-only queue to be executed so that the state is deterministic and verifiable anytime later. When a Secret Keeper is elected to process a set of requests on the next block, they will initiate the processing signal and are expected to finish processing within a defined block time.

Upon each Big Rotation Cycle (detailed in the next section), a secret will reach a **checkpoint**. One Secret Keeper who has access to the secret will package the current stage of the secret to a public storage network and register the CID_{result} to the secret registry and mark the latest executed write request associate with the **checkpoint**. After packaging, depends on the economic and machine capacities, the secret keeper can choose to off-load the secret or keep the secret for the next Big Rotation Cycle.

New Secret Keepers who did not have the secret before can choose to pick up a secret once it passes a checkpoint. At a definitive time, any Secret Keepers can pick up as many secrets for execution as they want, as long as it's aligned with their machine capacities and economic benefits. They can either pick up the secret from the initial state and re-run the execution of each **checkpoint** and confirm the correct packaging of the secret at each checkpoint, or they can pick up from the most recent **checkpoint** and only catch up with execution from there on. An inaccurate packaging challenge will reward the challenger and slash the original packager.

The consensus of the secret state is achieved by comparing the end-hash of each stage of each secret.

3.6 Secret Keeper Keys

To operate the vaults, Secret Keepers will need to generate and broadcast their public keys to receive key shreds. Leakage of the private key or the sealing keys can be catastrophic. To ensure the safety of all secrets within the network, Secret Keepers will be forced to rotate keys periodically. There are two keys to rotate, the public keys of each Secret Keeper (the Small Rotation), and the sealing key of each vault (the Big Rotation). The key rotations are done within the TEE enclave and can not be interrupted at the operating system level or be tampered with.

3.6.1 The Small Rotation

The Small Rotation is the process of rotating the public key of each Secret Keeper Nodes. Secret Keepers are required to rotate keys frequently. Those who fail to do so will be slashed and temporarily left out of the next session. At the start of each key session, Secret Keepers will need to submit a signature of their new public key for the next session, signed by their current keys. Once the signature is verified by Validators, the registry of the Secret Keepers will be updated.

Upon confirmation of verification from the Blockchain, the Secret Keeper will also run the recipient re-indexing process to reflect the updated public key.

3.6.2 The Big Rotation

Secret Keepers are required to rotate the sealing keys of each vault less frequently. Because the sealing key is inherently multi-purpose, upon finishing re-sealing all secrets available, the secret keepers will generate an aggregated signature of the most recent hash of all secrets with the sealing key and broadcast a public key derived from the sealing key. Validators shall verify the signature to confirm a successful Big Rotation.

Immediately after the blockchain confirms a successful Big Rotation, the node can choose to initiate a full re-indexing.

3.6.3 Leakage Challenge

Users can challenge any secret keeper with a hash of a private key that they believe is the private key of each Secret Keeper.

4. Other Features of the Blockchain

We believe that easy-to-use and developer friendliness are equally important as security for any blockchains. Therefore, several existing technology will also be integrated to the SkyeKiwi Protocol blockchain natively. Some will be made in collaboration with other blockchain communities.

4.1 Native Meta-Transaction Support

Meta-transaction refers to the technology that allows users to interact with the blockchain without paying the gas fee. It's usually achieved by DApp developers to deploy a relayer to relay a user signature of a transaction to the blockchain, while the DApp developers can assign customized rules and relayer fees. It's considered a simpler user-on-ramp method that does not require users to do KYC and purchase cryptocurrencies from fiat currencies.

The process can be modeled as a user U send a signature S_{tx} to a relayer R. The relayer R will send the transaction $\{S_{tx}, S_{relayer}, pk_U, parameters\}$, while the blockchain will not only verify the signature of the relayer but also the signature S_{tx} against the real origin's public key. (pk_U). There should be some economic incentive for the relayers to relay user's transactions as $Price_R = Cost_{tx} + Tip$. For security reasons, the relayer R shall also implement rules for processing relay requests, like a white-listing approach.

We will allow validators to opt-in of acting as relayers and build a `meta-tx` pallet to allow meta-transactions.

4.2 Native Human Readable Identifier

We will allow users to register a human-readable name on the blockchain mapping to their public address. Unlike the native Substrate identity pallet, such identity is by default a one-way trapdoor. These naming are designed to be a common good and extremely cheap to use. By default, one can reserve a name by staking a certain amount of the native token of the network, while staking profit received will be converted into a pool of renewal funds until the staking profit is not sufficient to reserve the name anymore. Such functionalities will be rolled out in steps.

4.2.1 Single Blockchain Mapping

The mapping on a single blockchain from a hash of a name to the public address on the blockchain. For instance, `skyekiwi.ksm` could be registered as the name for SkyeKiwi on Kusama, and “song.skw” can be registered as the name for Song on the SkyeKiwi Blockchain.

4.2.2 Cross-chain Support

Support to do a selected list of cross-chain transactions directly from different TLDs(top-level domains). For instance, “skyekiwi.ksm” could directly transfer assets to “song.skw” from Kusama to SkyeKiwi.

4.2.3 Resolver/Sub-Naming Support

Expose the naming pallet with a chain extension so that smart contracts can be created to act as a resolver to resolve sub-namings.

4.3 Native Integration of Other Products by the SkyeKiwi Team

While building the blockchain, the SkyeKiwi team also builds other DApps. Once the blockchain is launched, these applications will be migrated to the blockchain as the original DApps on the blockchain.

- **KiwiSign**: is a decentralized contract signature platform. We will integrate real-world contract signature capacities to the SkyeKiwi Protocol Blockchain. It will take a vault slot of the secret database.
- **Metastable**: is a creator-centric mNFT platform. mNFTs are NFTs that have their content masked. Creators can either publish public content or pay-walled private content for fans only. Metastable features a unique profit-sharing feature that while fans can stake on their favorite content creators to get into the pay-wall of the masked contents, they can also earn rewards when others join the creator’s fan base. It creates an economic incentive to advertise for the creators they like. When the creation of a creator is sold, portions of the proceeding will also be sent to the pool of fans.

4.4 Bridges to Other Blockchains

One of the major benefits of developing in the Polkadot ecosystem is the ability to securely and easily communicate with other blockchains in the system. We plan to be a parachain of Polkadot and Kusama. While the Crust Network is a critical part of our technology that handles all private data in the network, we will also integrate other blockchains to the SkyeKiwi Protocol Blockchain.

5. Economy

The SkyeKiwi Token (**SKW**) is the native network token of the SkyeKiwi Network. This section will provide an overview of the economy of SKW.

SKW is designed to align the interests of all stakeholders for a healthy and sustainable growth of the SkyeKiwi Network.

SKW is designed to facilitate:

1. Data Ownership: developers and community members should be fairly compensated for creating and maintaining secrets managed by the SkyeKiwi Network. The creator, administrator, and stakeholders of the secrets are the owner of the secrets, and their rightful claims to the secrets shall not be infringed by anyone else.
2. New Economic Model: blockchain and token economy have brought the world a new economic and trust model. However, typical blockchains handle private data poorly and the economical models when secrets are handled by the blockchain are uncharted territory. SKW should stand its roots in facilitating these new models.

5.1 The Token (SKW) Utilities

SKW will provide the following utilities:

- Network Security: to be a validator or Secret Keeper, one must stake a certain amount of SKW. The stake will be slashed for misbehavior against the community rules.
- Governance: participate in on-chain democracy.
- Gas Fee: similar to common blockchains, mutating the blockchain states will be charged a gas fee; different from common blockchains, querying a secret will also be charged.
- Reserve Trusted Computing Resource: in form of participating in leasing Secret Slots
- Others: **SKW** will act as the token of settlements for using services provided by the SkyeKiwi Blockchain and other ecosystem participants.

5.2 Supply of SKW

On network Genesis, 1,000,000,000 (a billion) SKW will be minted and they will be allocate in the following way:

- 200,000,000 for private sales (20%)
- 200,000,000 for community (20%)
- 250,000,000 for the team & advisors (25%)
- 150,000,000 for ecosystem building (15%)
- 200,000,000 for foundation reservation (20%)

- 1,000,000,000 (one billion) on genesis
- 3,000,000,000 (three billion) total supply

Later on, the supply of SKW will follow a two-phase release model to bootstrap the SkyeKiwi Network.

- Phase Alpha: approximately 253,633,616 SKW are minted and distributed to active node operators every year, for 5 years. This is equivalent to expanding the supply of SKW of 694,886.62 token in each of the first 1825 days.
- Phase Sigma: fewer tokens will be minted in each successive period and the daily minting amount decays exponentially with a half-life of 2 years.

Over the lifetime of the network, as $t \to \infty$, the total supply of the network will approach 3,000,000,000 (three billion) tokens.

5.2.1 Motivation of a Two-Phase Release

- Demand Uncertainty

  There is little certainty of the length of time between the network genesis and the development and meaningful demand from customers. New customers have to take a greater risk for an inadequate service. Consequently, on an exponentially decreasing supply schedule, when the fees market is not established enough, the node operators might be rendered entirely unsustainable with decreasing subsidies.

- Centralization

  If token supply expanding is following an exponentially decreasing schedule, on early days of the network, larger operators are granted unfair advantages over later participants and might render the network to be more centralized. Therefore, a two-phase minting schedule assists all would-be node operators who do not happen to stake right at the network genesis but decide to do so at a later date.

- Risk to pricing

  If the work required to collect subsidies does not align well with the work required to earn fees, the incentives for node operators might not be aligned with the purpose of the larger community. To make things worst, high but decreasing subsidies may compel node operators to offer unsustainably cheap service and raise prices once subsidies shrink in the early days of a network. Once developers and end-users became reliant on the services but unable to afford the higher price, the network will put their application and business in jeopardy.

These concerns are not strong claims as to "when or if" the network adoption will occur. Instead, they have acknowledged the uncertainties of the early stage of a decentralized network.

5.2.2 Parameters of Supply

- S_0 the supply at genesis, equals to 1 Billion SKW.
- S_∞ the total supply, equals to 3 Billion SKW.
- $T_{1/2}$ half-life of the supply expansion in Phase Sigma, equals to 2 years
- I_α stable issuance rate in Phase Alpha
- I_σ decreasing issuance rate in Phase Sigma

We can construct the equation:

$$S_\infty = S_0 + S_0 \cdot \int_0^5 I_\alpha(t)dt + S_0 \cdot \int_0^\infty I_\sigma(t)dt$$
$$I_\sigma = I_\alpha \cdot 2^{-\frac{t}{T_{1/2}}}$$

After plugin the numbers:

$$3 = 1 + 5 \cdot I_\alpha + I_\alpha \cdot \left[\frac{-2^{1-\frac{t}{2}}}{\ln 2}\right]_0^\infty$$
$$2 = I_\alpha \cdot (5 + \frac{2}{\ln 2})$$
$$I_\alpha = 25.4\%$$

Therefore, by applying $I_\alpha \cdot S_0$, there are about 253,633,616 new tokens issued each year for the first 5 years. This is equivalent to 694,886.62 new tokens issued every day for the first 1825 days from the network genesis. Then decreasing at a two-year half-life rate.

5.3 Staking Limit & Network Security

There are three types of participants in staking in the SkyeKiwi Network, the Validators, Secret Keepers and Nominators. Similar to the design of Polkadot system, Nominators will nominate their ideal Validators or Secret Keepers, and be rewards or slashed as the Validators and Secret Keepers.

The staking subsidies come from the new issuance of the **SKW** tokens. In case of misbehavior of the Validators and Secret Keepers, they will be slashed. The slashed amount will be positively correlated to the staked amount. Therefore, Nominators are encouraged to transfer their nomination to nodes with a smaller stake when the current node they are staking with is beyond their risk parameter.

To better align the actual workload of handling secrets and the actual market demand, a staking limit will be imposed by the hardware capacity and historical performance of node operators.

- Responsive & Stay Online - High Impact
- Historical Derogatory Marks - High (Negative) Impact
- Hardware Performance - High Impact
- Historical Responsive Record - Medium Impact
- Historical Key Rotation Record - Medium Impact
- Historical Checkpoint Packaging Record - Medium Impact
- Historical Extrinsic Execution - Low Impact

5.4 Secret Slot Auction & Gas Fee

Because the secrets in the system are composable, each secret can be regarded as a service offering one type of data to the network. While there can be an infinite number of smart contracts, there will be a limited number of secrets. One secret can have multiple smart contracts accessing it and secrets can depend upon other secrets as well. Such mechanism gives early secret builders a better economic incentive to initiate a secret on the SkyeKiwi Network. Due to frequent key rotations, processing and maintain a secret is an expansive process. Therefore, limiting the total number of secrets available to the network becomes necessary. When the network got bootstrapped, developers who build on top of it bear more risk of an early and inadequate network and the secret slots are not scarce. Secret Slots will be open to being reserved on a lease based by locking a small amount of SKW, which can be subsidized by the on-chain treasury or through the ecosystem development reserve from the foundation. When the network got more developed and secret slots are becoming scarce, Secret Slots renewal and registration will enter an auction mechanism, very similar to how Polkadot and Kusama manage their parachain auctions.

Gas Fee will be charged on par with the typical design of all Substrate chains. For private contracts, the gas fee will be higher to compensate both the creator of the secret and the Secret Keepers. Owners of the secret can be an external address or a smart contract to abstract the data ownership into DAOs. Secret contract creators can also specify extra charges from the data users through a “payable” interface in smart contracts.

Additionally, Secret Keepers will be rewarded and or slashed after successful or unsuccessful packaging of the secret **checkpoint** state.

5.4.1 Determine Transaction Fee

Plain Substrate based blockchains define transaction fees according to the following paradigm:

- For a transaction “tx”, $Length(x)$ defines the byte length of the transaction and P defines the price per byte.
- $\Omega(tx)$ defines a function to convert the **Weight** of a transaction to fees. (I.e. Weight is defined by the complexity of the request)

$$Fee = Length(tx) \cdot P + \Omega(tx)$$

For smart contract calls that involve secret contract calls:

- For a transaction “tx”, $\Omega_O(tx)$ defines the weight conversion function of ordinary smart contract calls, while $\Omega_S(tx)$ Defines the weight conversion function of secret smart contract calls.
- While, $\Omega_S(tx) = \Omega_O(tx) \cdot S$, while S is a dynamic parameter to mark up the price for secret contract calls. S will be referred as the **secret execution fee** parameter **(SEF)**.

Therefore:

$$\begin{gathered} Fee = Length(tx) \cdot P + \Omega_O(tx) + \Omega_S(tx) \\ Fee = Length(tx) \cdot P + (1 + S) \cdot \Omega_O(tx) \end{gathered}$$

SEF is designed to compensate Secret Keepers to process requests and might differ for each secret. SEF will be determined by the capacities of the Secret Keeper Nodes. Theoretically, for an infinitely powerful machine, the marginal cost of resources to process each request should be equal. However, realistically, the marginal cost of processing requests will most likely increase for machines, from a composition of higher networking and CPU/Memory cost and high knowledge cost to correctly configure machines to achieve high throughput. Therefore, the SEF should correctly markup frequently called secrets to compensate miners. As of a testnet genesis, it will be set at a fixed ratio of 0.5, and will be adjusted through upgrades when more realistic network data is accumulated and will be eventually determined dynamically when the mainnet is launched.

5.4.2 Gas Fee Income Distribution

Unlike the typical gas fee distribution design, which distributes all income from gas fees to node processors, SkyeKiwi Network will also distribute a portion of gas-fee income to the data owner of each secret. Owners can be an individual external account or a Multi-Sig address of a community DAO. While processing requests, Secret Keepers will also be responsible for packaging **checkpoints** of secrets and should be compensated accordingly.

Therefore, income distribution will follow the following principle:

- γ_{owner} and γ_{sk} define the distribution ratio for “secret owners” and “Secret Keepers”.
- $\gamma_{treasury}$ defines the portion of the income going into the community Treasury.
- $\gamma_{checkpoint}$ is the pool of income set aside to compensate Secret Keepers who package a secret when it reaches a **checkpoint**.
- $FEE_{base} = Legnth(tx) \cdot P + \Omega_O(tx)$ and $FEE_{secrets} = \Omega_S(tx)$

And,

- Income of Secret Keepers is $(1 - \gamma_{reasury} - \gamma_{checkpoint}) \cdot (FEE_{base} + \gamma_{sk} \cdot FEE_{secrets})$
- Income of data owners is $(1 - \gamma_{treasury} - \gamma_{checkpoint}) \cdot (\gamma_{owner} \cdot FEE_{secrets})$
- Income for the first few Secret Keepers who successfully package a secret at a **checkpoint** is $\sum_i^t FEE_{total} \cdot \gamma_{checkpoint}$, and shall be distributed on an exponentially declining nature for those who completed or verify the task. (i.e. 50% for the first one who finished, 25% for the second node who seconds the result ... $(1/2)^i$ For the i who finished or prove that correctness of the packaging. For those who successfully challenge a result, income from previously rewarded income will be clawed back and an additional percentage of punishment will be imposed from those who produced the wrong packaging and rewarded 100% to the challenger.
- Income of treasury is $\sum_i^n FEE_{total} \cdot \gamma_{treasury}$

5.4.3 Secret Slots

Theoretically, the value of each secret slot varies widely because of the nature of each different secret. For the network, a secret who gets more called is generally more resource-consuming but more valuable. For a naive economic model for secret slots, each subsequent secret slot shall raise the average number of calls of secrets of the network. (i.e. a bar raiser for the whole network), so that the income achieved by the owner of the secret can positively compensate the raised secret slot leasing fees. Therefore, following such a naive economic model, the lease price of each secret shall increase as the network collects more fees related to secret calls. However, a such raising price model might dis-incentivize developers who are late to the game. Fortunately, the treasury income of the network is positively related to the income of fees. Therefore, it will be economically sustainable for the treasury to subsidies meaning new secret slot applicants.

Therefore, the economic model of a new secret slot at position i can be summarized as:

- Define $I_{treasury}(i-2, i-1)$ as the income of the treasury from **transaction fees** of the last auction/registration period.
- (Base) Lease price of a secret slot is $I_{treasury}(i-2, i-1) \cdot 110\%$ to lock during the lease period. It will be the base price of the lease and can be more expansive when there is more demand of the slot but should always topped at $I_{treasury}(i-2, i-1) \cdot 160\%$ to prevent high price change of the secret slots over time. Ideally, the lease price of the secrets should be very close to the actual treasury income from the gas fee within the next period.
- Developers can submit proposals of the secret and being reviewed and voted following the ordinary governance rules. When approved, they would only need to lock in less than 10% of the price required to reserve the secret slot. Such

treasury proceedings shall not be used elsewhere but only for the secret slot auction. We would always encourage developers to apply for the treasury before proceeding with secret slot auctions to receive more opinions of their offering from the community. For teams that are working on common good secrets, they can submit additional applications to the SkyeKiwi Network ecosystem development foundation funding, or through an additional treasury spending motion on top of their existing application for secret slot subsidies.

- Within each auction period, developers who have applied to the treasury secret slot subsidies will be publicly listed and voted on by council members. Developers need to lock in at least a base fee as a refundable bond as their bid up to the maximum price of each secret slot.
- Similar to the Polkadot Parachain candle auction mechanism, a snapshot of the bidding result will be randomly selected and the highest bidder will win the auction. Before the auction begins, the councils will make a decision of who to reward the treasury.
- Because there is a cap of the auction bid of each period, it will be very likely to have tied bids. To be fair, those who locked their own tokens without going through the treasury application will be of higher priority in case of a tied auction. Otherwise, tied bids will be randomly selected to be the winner.
- When an application wins the auction, the proceeding from the treasury spends will not be available to be withdrawn as their ordinary tokens in the wallet. Instead, the current average traffic of all secrets will be recorded as of finishing the bid. When the lease ends and the fund released, those developers who achieved higher secret traffic will be awarded the fund, those who do not will be clawed back by the treasury. The proceeding can be used for renewal of the secret slot as if their own funds or they can apply for treasury fundings again for the renewal.

5.5 An Use Cases Example

In this section, we are giving an example of a service model. The example is not intended to be a guideline or a proposal of any kind but a naive example to demonstrate how a new economic model can be built with the native economic model by the SkyeKiwi Network.

Consider an on-chain Fiat management service (that we named “SkyePay”) that stores a secret database of user credit card information in the secret storage and provide the service of process fiat payment for users and other smart contracts. SkyePay depends upon a KYC service offered by another blockchain, while the contract signature service offered by the SkyeKiwi Network (the “KiwiSign”) for managing the signed end-user Term of Service agreement, in accordance with applicable laws.

For a customer, upon successful KYC from the third party KYC provider, the KYC approval for a public address can be verified by the smart contract of SkyePay. When the user wants to purchase $1000 worth of SKW on-chain, the user will use the front-end of the SkyePay to package their credit-card information according to the SkyeKiwi Protocol and send the request to blockchain calling a smart contract message “purchaseWithFiat” for example, with a definitive time bound for the Secret Keepers to purge the record. SkyePay can proceed the transaction through a bank transaction API and pay 0.5% for each transaction to the third-party vendor and collect an additional 1% of fees from the end-users and associated gas fee to process the transaction. Such schema is almost identical to existing centralized fiat-on-ramp solutions.

To push it one step forward, because the smart contract can manage fiat currency (i.e. bank account), it can act as a decentralized fiat-crypto swap with the popular automatic market maker mechanism. A liquidity provider can provide a fiat currency processing API instance and a bank account along with some cryptocurrency to the pool and earn the swapping fee. The best part of this is which it will be interoperable for other smart contracts, secret smart contracts, or even smart contracts from other DeFi chains.