Golden Gate

Interchain Infrastructure Protocol

Matthew Doty Viktor Ihnatiuk Yehor Butko
Artur-Yurii Korchynskyi Timofei Sokolov
Danyil Poliakov

1 Introduction

The future of the blockchain ecosystem is multi-chain. We are at the cusp of a Cambrian explosion of multi-chain transactional activity. Market forces are driving a centralized ecosystem to a decentralized one. To date, BitCoin and Ethereum make up $> 60\%$ of the market. At the same time, the two blockchains average around 15 transactions per second. The two pieces of software are also notorious for their slow release cadences. These factors have led to a torrent of innovation in the form of Layer 1 and 2 chains. However, this wave of invention has challenges.

The growing internet of blockchains requires practical communication tools for liquidity routing. Furthermore, emerging chains are not accessible to the vast majority of the community. This inaccessibility creates significant market friction in the form of *price delay*. Price delay measures the speed the market reacts to information[20]. While price delay is in decline in cryptocurrency[22], certain driving factors remain. In particular, there is a clear correlation between price delay and liquidity.

The chain-to-chain communication problem is of massive economic importance. Billions of dollars worth of cryptocurrency are locked in cross-chain custodial contracts[32]. Industry interest has led to extensive research with many remaining open questions[25].

Golden Gate is a novel Layer 0 that solves the interchain communication problem. It allows for cross-chain orchestration via smart contracts. It uses zk-SNARKs for efficient cross-chain synchronization and communication with other chains through industry-standard messaging protocols. Golden Gate leads the industry by bringing formal methods from aerospace to the blockchain. Finally, Golden Gate scales as a relay chain and prioritizes peering with other relay chains.

2 Communication Protocols

2.1 Overview

There are four mechanisms for communication used by Golden Gate and its connected smart contracts.

1. Incentivized Message Delivery Protocol (IMDP)
2. The LayerZero Protocol
3. Cross-Consensus Message Passing (XCMP)
4. Interblockchain Communication Protocol (IBC)

The mechanisms differ in implementation prerequisites, guarantees provided, and gas pricing. Golden Gate supports many connections via different mechanisms to a specific external chain.

2.2 Incentivized Message Delivery Protocol

Golden Gate uses a novel *Incentivized Message Delivery Protocol* (IMDP) for efficient cross-chain communication. IMDP uses a network of *couriers* to deliver messages. Any computer may run a courier node. Courier nodes run Golden Gate light clients. Courier nodes pay gas fees for communicating messages from Golden Gate. In exchange, they earn rewards through fees paid by users. While validators could run courier nodes if they wished, the blockchain and courier network are designed to be distinct and decentralized.

The lifecycle of a message demonstrates the role of the different Golden Gate components play. We depict this lifecycle in Figure 1.

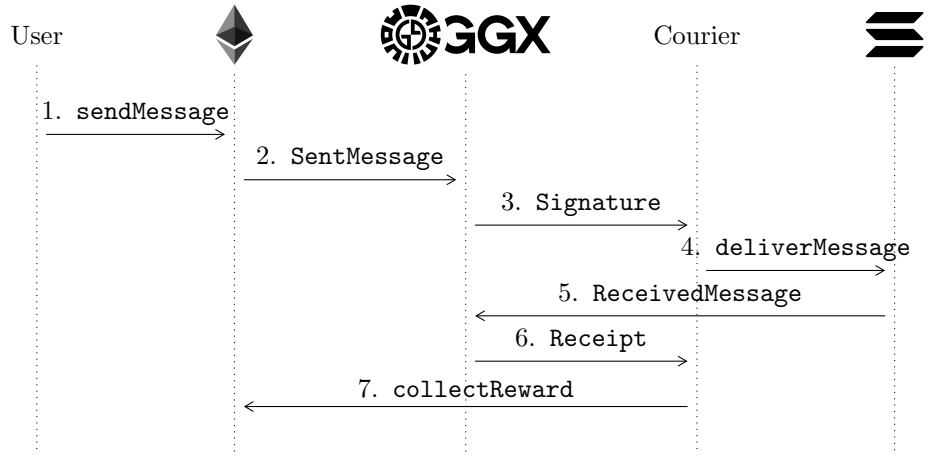

Figure 1: Lifecycle of a User's call to Golden Gate's smart contracts

1. To start, a User sends a message by invoking the `sendMessage` method on a designated smart contract on blockchain ♦ (note that ♦ is distinct from Golden Gate GGX). The User specifies a destination chain and address when they submit a message. In this case, the destination chain is denoted by ≡. The User deposits a reward to be collected in Step 7.

2. Next, GGX detects the user's message via a light client for ♦, listening for an `SentMessage` event. Golden Gate follows each of the blockchains in its network via a separate light client.

3. GGX generates a signature for the Users' message through MPC protocol. Golden Gate's blockchain keeps a permanent record of the signed messages for retrieval. It puts a signature on an event stream. A Courier picks up the `Signature` on Golden Gate's event stream.

4. The Courier delivers the signed message to the destination chain by calling `deliverMessage` on ≡.

5. GGX detects a `ReceivedMessage` event from ≡ using its light client.

6. $\frac{2}{3}$rds of the validators for GGX sign a `Receipt` and put it on GGX's event stream. The `Receipt` is detected by the Courier.

7. The courier returns the receipt to the origin chain ♦ by calling `collectReward` to claim the reward the user deposited in Step 1.

2.3 LayerZero Protocol

LayerZero[23, 40] is a recent emerging communication protocol intended to work across multiple blockchains. The protocol involves two actors: *oracles* and *relayers*. In the context of Golden Gate, the blockchain serves as a decentralized oracle. Oracles forward block headers from one chain to another. Relayers communicate Merkle proofs of events that are checked using headers notarized by the oracles. The courier network serves as the relayer network. Because Golden Gate controls its virtual machine, as discussed in §6, decentralized oracle support is accessible to all users.

Golden Gate extends the LayerZero protocol in two ways while maintaining backward compatibility. In addition to block headers, Golden Gate also notarizes:

- Contract States
- Emitted Events

By notarizing this other state beyond just notarizing block headers, Golden Gate improves LayerZero's inter-chain communication efficiency.

As in §2.2, we demonstrate the workflow of Golden Gate's LayerZero protocol below and depict how message flow in Figure 2.

1. Golden Gate monitors ♦ continuously, listening for `epoch update` events. These occur when the ♦ blockchain rotates validators. This way GGX is always up-to-date on the current set of validators.

2. The User transacts with some smart contract on ♦, which emits an event `TransactionEvent`.

3. The Courier network (acting as a LayerZero relay) listens to ♦ for a `TransactionEvent`.

4. The Courier network then requests for the block, contract data, and an associated Merkle proof from ♦ for the smart contract.

5. The ♦ blockchain responds with the requested data.

6. The Courier network uses the data from ♦ and makes an RPC call `requestSigs` to GGX. The payload of the RPC call contains the necessary data to produce a cryptographic validation of the smart-contract state.

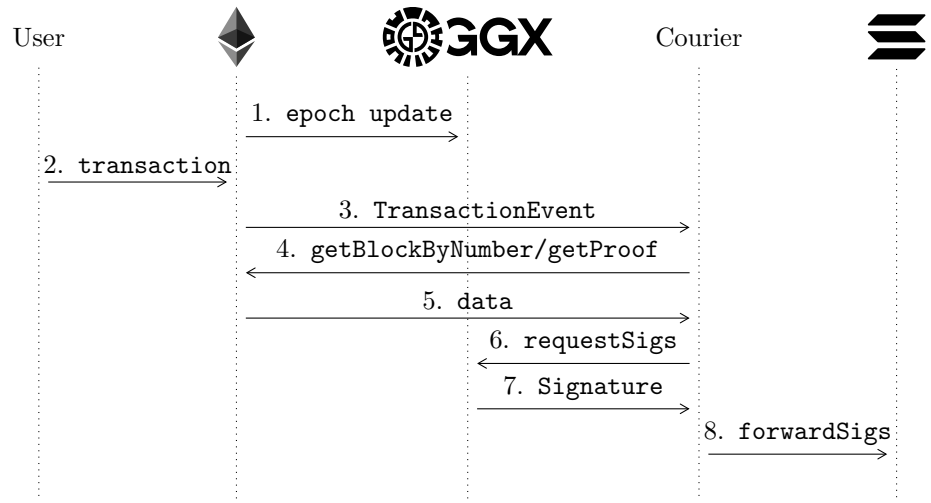

Figure 2: Golden Gate's LayerZero workflow

7. Golden Gate processes the block signatures and Merkle proofs. It then signs the data and responds to the Courier with a `Signature`.

8. The Courier network sends a `forwardSigs` smart contract call to ≡, completing the cross-chain interaction.

LayerZero complements IMDP. While IMDP permits a smart contract to communicate state across blockchains, LayerZero allows a smart contract to receive notifications from another contract, even when that state is not being sent directly.

2.4 IBC Protocol

The *Inter-Blockchain Communication Protocol*[16] (IBC) designed to facilitate communication of sovereign replicated ledgers that share only a minimum requisite common interface. IBC handles authentication, transport, and ordering of opaque data packets relayed between modules on separate ledgers can be run on solo machines, replicated by many nodes running a consensus algorithm, or constructed by any process whose state can be verified. The protocol is defined between modules on two ledgers, but designed for safe simultaneous use between any number of modules on any number of ledgers connected in arbitrary topologies. IBC requires certain functionalities and properties of the underlying ledger. It requires *finality signatures*, cheaply-verifiable consensus transcripts, and a simple key/value store. On the network side, IBC requires only eventual data delivery — no authentication, synchrony, or ordering properties are assumed.

While IBC may be implemented on any blockchain with smart contracts[21], in practice the gas costs are too high. This is because IBC-supporting blockchains such as Cosmos use Ed25519 signatures[1]. Ethereum does not have a precompiled contract for checking these signatures[26]. As a consequence, it costs 500,0000 gas to check an Ed25519 signature, and millions to run a light client.

Golden Gate includes IBC support via the IBC substrate palette[37]. Golden Gate parachains can also opt into IBC support via the same palette.

IBC does not provide specific protocol-level provisions for compute-level or economic-level flow control. The Courier network is expected to have compute throughput limiting and flow control mechanisms of their own such as gas markets.

We demonstrate the workflow of Golden Gate's IBC protocol below and depict how messages flow in Figure 3.

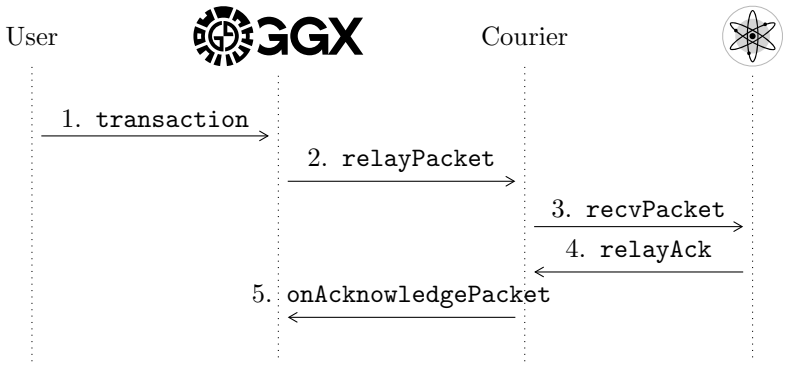

Figure 3: Golden Gate's IBC workflow

1. To start, the User makes a `transaction` calling a designated smart contract on GGX, which emits an event `relayPacket`. GGX saves all information about the connection, channel and message. The User specifies a destination chain and address when they submit a transaction. In this case the destination chain is denoted by ✳.
2. The Courier listens to GGX for a `relayPacket` event.
3. The Courier sends message to a designated smart contract on ✳ invoking a `recvPacket` event.
4. Smart contract on ✳ relays an acknowledgement to the Courier using a `relayAck` method.
5. The Courier[2] sends an acknowledgment to GGX calling an event `onAcknowledgePacket` on a designated smart contract.

[1] Golden Gate also uses this signature scheme and others, see Appendix A.1.

[2] note: in this step the Courier may have changed and the protocol would still function properly.

2.5 XCMP Protocol

Golden Gate supports the *Cross-Consensus Message Passing* (XCMP) protocol. XCMP is a protocol designed by the Web3 Foundation for communication between parachains[10, 39]. The implementation of XCMP relies on a shared relay chain between the communicating chains. Given this restriction, XCMP guarantees trusted and ordered delivery as long as the receiving chain keeps producing blocks. Furthermore, it enables shared code execution across blockchains via *Shared Protected Runtime Execution Enclaves* (SPREE). SPREE allows parachains to use shared code and enforce that they execute this piece of code and no other. In particular, using a SPREE module would allow for *asset teleportation*. In this scheme, a token is burned and minted on two different chains simultaneously, relying on the shared execution environment to mediate the computation.

For usage in Golden Gate, however, the requirement of having a shared relay chain restricts the usage of XCMP to parachains within the Golden Gate ecosystem. IBC or Incentivized message delivery protocol connections are needed to communicate with blockchains not connected to Golden Gate's relay chain.

3 Multi-Party Computation

3.1 Protocol Overview

Golden Gate employs *Multi-Party Computation* (MPC) to generate signatures for applicable connected chains[3]. Multi-Party Computation is a subfield of cryptography that introduces methods for parties to compute a function over private inputs jointly. We use MPC to distribute signature generation and prevent dependencies on a single private key for transaction authorization.

Golden Gate implements *Fast Multi-party Threshold ECDSA with Fast Trustless Setup*[12] for secure distributed key generation using MPC. In this scheme, a secp256k1 signature is generated jointly by nodes designated by *Nominated Proof of Stake* (discussed in §4). If $t+1$ nodes successfully participate in MPC, a valid secp256k1 signature is produced. Even if the attacker controls t nodes, he cannot forge a legitimate signature. The attacker needs to subvert at least $t+1$ nodes to create a legitimate signature. Golden Gate sets the parameter t to equal $\frac{2}{3}$rds of the number of nodes. The $\frac{2}{3}$ fraction reflects the liveness assumptions of PBFT-based consensus. Note that because Nominated Proof of Stake effectively minimizes validator variance, there is no need to assign weights to signatures in the threshold scheme.

Fast Multi-party Threshold ECDSA with Fast Trustless Setup uses additively homomorphic encryption, a commitment scheme, verifiable secret sharing, and zero-knowledge proofs. In additively homomorphic encryption, one can sum encrypted data without decrypting it. Specifically, we let $E : \mathbb{Z}_N \to \mathbb{Z}_N$ for some

[3]Currently, most (over 80%) of encrypted digital currencies adopt the same secp256k1 ECDSA signature algorithm as Bitcoin and Ethereum, which is the signature scheme supported by our MPC implementation.

prime $N = PQ$ where P and Q are primes (and N is hard to factor), and require $E(x+y) = E(x)+E(y)$ for all x and y. A commitment scheme is a cryptographic primitive that allows one to commit to a chosen value (or statement) while keeping it hidden from others, with the ability to reveal the committed value later. Commitment schemes are designed so that a party cannot change the value or statement after they have committed to it (i.e., commitment schemes are binding). Verifiable secret sharing is an extension of Shamir's secret sharing in which auxiliary information is published that allows nodes to check that their shares are consistent and define a unique secret. Finally, zero-knowledge proof is a method by which one party (the prover) can prove to another party (the verifier) that a given statement is true, while the prover avoids conveying any additional information apart from the fact that the statement is indeed true. Commitment scheme is defined by 2 functions:

1. Commit - $\mathsf{Com} : \mathsf{PrivKey} \times \mathsf{Message} \to \mathsf{Commitment} \times \mathsf{Decommitment}$ - a function takes the private key and the message to commit. Outputs are a pair of $(C, D) = \mathsf{Com}(pk, M)$ which are commitment and decommitment strings - $C : \mathsf{Commitment}$ and $D : \mathsf{Decommitment}$, respectively.

2. Verify - $\mathsf{Ver} : \mathsf{PubKey} \times \mathsf{Commitment} \times \mathsf{Decommitment} \to \mathsf{Message}$ - a partial function that takes the public key, commitment, and decommitment strings as inputs. The output is a committed message $M = \mathsf{Ver}(Pk, C, D)$ in the case of a successful verification or $\perp$ otherwise. We further assume that $\mathsf{Ver}(Pk, C, \_)$ is injective[4].

We define our MPC approach over a generic (EC)DSA signature algorithm. It works over any group G of prime order q generated by an element g. It uses a hash function H defined from arbitrary strings into $\mathbb{Z}_q$, and another hash function H' defined from G to $\mathbb{Z}_q$. The secret key is x chosen uniformly at random in $\mathbb{Z}_q$, with a matching public key $y = g^x$. To sign a message M, the signer computes $m = H(M) \in \mathbb{Z}_q$, chooses k uniformly at random in $\mathbb{Z}_q$ and computes $R = g^{k^{-1}}$ in G and $r = H'(R) \in \mathbb{Z}_q$. Then she computes $s = k(m+xr) \mod q$. The signature on M is the pair (r, s) which is verified by computing

$$R' = g^{ms^{-1} \mod q} y^{rs^{-1} \mod q}$$

... in G and accepting if $H'(R') = r$.

The ECDSA signature scheme is obtained by choosing G as a group of points on an elliptic curve of cardinality q. In this case, the multiplication operation in G is the group operation over the curve. The function H' is defined as $H'(R) = R_x \mod q$ where R_x is the x-coordinate of the point R.

The technical complication with sharing ECDSA signatures comes from having to compute jointly R (which requires raising g to the inverse of a secret value k) and to compute s which requires multiplying two secret values k and x. As

[4] In practice, one can use any hash function for Com where a collision is negligible to achieve this.

shown in [14], it is sufficient to show how to compute two multiplications over secret values that are shared among the players.

We employ [12] approach to calculate k^{-1} and kx. First, we additively share a and b among the nodes (i.e., $a = \sum_1^n a_i$ and $b = \sum_1^n b_i$ where a_i and b_i are respective private shares of the nodes). Then we generate an additive sharing of $c = ab$. We note that $ab = \sum_{i=1}^n a_i \sum_{j=1}^n b_j = \sum_{i,j=1}^n a_i b_j$, and therefore to get an additive sharing of ab it is sufficient to obtain an additive sharing of each individual term $a_i b_j$. To that end, we use a 2-party protocol that allows parties to transform multiplicative shares of a secret into additive shares of the same secret. The players engage in this protocol in a pairwise fashion to obtain an additive sharing of the product ab. Using additive sharing of 2 random values and additive sharing of their product, we construct a signature in a distributed manner.

Fast Multi-party Threshold ECDSA with Fast Trustless Setup has three stages: *initialization*, *distributed key generation*, and *threshold signature generation*.

1. In the *initialization* stage each node P_i is associated with a Pallier public key E_i for an additively homomorphic encryption scheme E based on RSA[28]. Each node keeps a private key e_i corresponding to this public key.

2. The *distributed key generation* stage consists of 3 phases

   (a) Each node P_i selects a random $u_i \in \mathbb{Z}_q$ then calculates and broadcasts commitment string C_i from $(C_i, D_i) = \mathsf{Com}(pk_i, E_i \,||\, g^{u_i})$, where $A||B$ is the concatenation of A and B.

   (b) Each node P_i broadcast decommitment string D_i. Nodes generate private shares x_i through verifiable secret sharing and jointly compute a public key $y = g^{\sum_{i=1}^n x_i} = g^x$. Note that the values of g^{x_i} are public.

   (c) Let $N_i = p_i q_i$ be the RSA modulus associated with E_i. Nodes provide zero-knowledge proofs that they each know their respective x_i using Schnorr's protocol [31], and that N_i is square-free using the proof system by Gennaro, Micciancio, and Rabin [13].

3. In the *threshold signature generation* stage, nodes generate a signature in a distributed matter on the input of m (the hash of the message M being signed).

We now describe the signature generation protocol. Let $S \subseteq [1..n]$ be the set of nodes participating in the signature protocol. We assume that $|S| = t+1$. For the signing protocol, we can share any ephemeral secrets using a $(t, t+1)$ secret sharing scheme and do not need to use the general (t, n) structure. Define w_i as a $(t, t+1)$ share of a secret w (i.e., $x = \sum_{i \in S} w_i$). The protocol consists of 5 phases:

1. Each node P_i chooses $k_i, \gamma_i \in \mathbb{Z}_q$. Nodes calculate and broadcast commitment $C_i = \mathsf{Com}(pk_i, g^{\gamma_i})$. Define $k = \sum_{i \in S} k_i$, $\gamma = \sum_{i \in S} \gamma_i$. Note that $k\gamma = \sum_{i,j \in S} k_i \gamma_j$ and $kw = \sum_{i,j \in S} k_i w_j$.

2. Every pair of nodes P_i,P_j engage in two multiplicative-to-additive share conversion subprotocols.

   (a) For the product $k_i \gamma_j$, define corresponding additive shares as $\alpha_{i,j}$and $\beta_{i,j}$ (i.e., $k_i \gamma_j = \alpha_{i,j} + \beta_{i,j}$). Each node P_i sets $\delta_i = k_i \gamma_i + \sum_{j \neq 1} \alpha_{i,j} + \sum_{j \neq 1} \beta_{i,j}$.

   (b) For the product $k_i w_j$, define corresponding additive shares as $\mu_{i,j}$ and $\nu_{i,j}$ (i.e., $k_i w_j = \mu_{i,j} + \nu_{i,j}$). Each node P_i sets $\sigma_i = k_i w_i + \sum_{j \neq 1} \mu_{i,j} + \sum_{j \neq 1} \nu_{i,j}$). Nodes prove that they know the value k_i via zero-knowledge proof.

3. Each node P_i broadcasts δ_i and reconstructs $\delta = \sum_{i \in S} \delta_i = k\gamma$. Nodes then compute $\delta^{-1} \mod q$.

4. Each node broadcasts D_i from the first phase and provides a zero-knowledge proof that it knows γ_i. Nodes then compute

$$\begin{aligned} R &= \left(\prod_{i \in S} g^{\gamma_i} \right)^{\delta^{-1}} \\ &= g^{(\sum_{i \in S} \gamma_i) k^{-1} \gamma^{-1}} \\ &= g^{k-1} \end{aligned}$$

   ... and let $r = H'(R)$.

5. Each node P_i sets $s_i = mk_i + r\sigma_i$. Note that the signature s is

$$\begin{aligned} s &= \sum_{i \in S} s_i \\ &= m \sum_{i \in S} k_i + r \sum_{i \in S} \sigma_i \\ &= mk + rkw \\ &= k(m + wr). \end{aligned}$$

   Nodes prove that they know s_i through subsequent commitments and zero-knowledge proofs. Nodes output the signature s in case of a successful proof. A sophisticated 5-round approach is taken to avoid expensive zero-knowledge proofs and the leakage of information to an adversary. See [12] for details.

The *Fast Multi-party Threshold ECDSA with Fast Trustless Setup* signature scheme requires only a small amount of storage and computational power for nodes to participate, which makes Golden Gate scalable. The computational power required is constant; only zero-knowledge proof verification scales linearly with the number of nodes (as nodes have to verify all of the zero-knowledge proofs produced by other nodes).

3.2 Distributed Key Generation Implementation

To generate a new key, nodes enter the *distributed key generation* stage described in subsection 3.1.

Before generating a key, nodes run the *initialization phase* to create a public key E_i for the Pallier encryption scheme.

Every message for a specific key generation instance contains the epoch number and a node signature.

After the initialization phase, the distributed key generation is implemented in the blockchain as follows:

1. Each node P_i selects a random $u_i \in \mathbb{Z}_q$ then calculates and broadcasts commitment $C_i = \text{Com}(pk_i, g^{u_i} + E_i, pk_i)$. If a node times out while broadcasting, they are ejected, and the protocol continues without them.

2. Nodes broadcast decommitment D_i for the commitment C_i. Whenever a node receives $\mathsf{Ver}(Pk_i, C_i, D_i) = \perp$, node P_i gets slashed. If a node times out or gets slashed, they are ejected, and the protocol continues without them.

3. Nodes generate and verify x_i using verifiable secret sharing. Whenever verification fails, the protocol restarts.

4. Nodes provide zero-knowledge proofs that they each know their respective x_i and that N_i is square-free. Whenever one of the proofs fails, that node is slashed, and the protocol restarts.

5. The threshold parameter t is set to equal $\frac{2}{3}$rds of the number of nodes. Protocol publishes y and t to the blockchain for that epoch.

3.3 Distributed Signature Implementation

To sign a message, nodes enter the *threshold signature generation* stage described in subsection 3.1, adding hash of the message being signed m to each broadcast along with a signature:

1. Each node P_i chooses $k_i, \gamma_i \in \mathbb{Z}_q$. Nodes calculate and broadcast the commitment $C_i = \text{Com}(g^{\gamma_i}), pk_i$.

2. Every pair of nodes P_i,P_j engage in two multiplicative-to-additive share conversion subprotocols.

(a) For the product $k_i\gamma_j$, define corresponding additive shares as $\alpha_{i,j}$ and $\beta_{i,j}$ (i.e., $k_i\gamma_j = \alpha_{i,j} + \beta_{i,j}$). Each node P_i sets $\delta_i = k_i\gamma_i + \sum_{j\neq 1}\alpha_{i,j} + \sum_{j\neq 1}\beta_{i,j}$

(b) For the product $k_i w_j$, define corresponding additive shares as μ_i and ν_j (i.e., $k_i w_j = \mu_{i,j} + \nu_{i,j}$). Each node P_i sets $\sigma_i = k_i w_i + \sum_{j\neq 1}\mu_{i,j} + \sum_{j\neq 1}\nu_{i,j}$. Nodes prove they know the value k_i via zero-knowledge proof. Whenever zero-knowledge proof for a node P_i fails, the node gets slashed. The protocol waits here until a threshold of $t+1$ nodes have successfully communicated the required information. Nodes continue to collect shares in case of future Byzantine behavior.

3. Each node P_i broadcasts δ_i and reconstructs $\delta = \sum_{i\in S}\delta_i = k\gamma$. Nodes then compute $\delta^{-1} \mod q$. The set of validators S is published to the blockchain.

4. Each node broadcasts decommitment string D_i for the commitment C_i from the first phase and provides a zero-knowledge proof that it knows γ_i. Whenever a node receives $\mathsf{Ver}(Pk_i, C_i, D_i) = \perp$, node P_i gets slashed, and the protocol returns to step 3 (provided that the threshold of $t+1$ is sufficed). Whenever a zero-knowledge proof for node P_i fails, the node gets slashed, and the protocol returns to step 3. Nodes then compute $R = g^{k-1}$ and $r = H'(R)$.

5. Nodes calculate s_i and prove that they know it through a zero-knowledge proof. Whenever zero-knowledge proof fails for the node P_k node gets slashed, the protocol returns to step 3. Likewise, if (r, s) is not a valid signature, the protocol restarts; otherwise, output (r, s) as a signature.

To rotate keys, Golden Gate needs to update smart-contracts on bridged chains. To update smart-contracts on bridged chains, Golden Gate signs transactions using the previous set of nodes. Nodes publish the resulting signed transaction along with the public key. If the public key does not match the key published in new key generation phase nodes, nodes retry the smart-contract update.

If a validator times out they get a strike and the strike is recorded to the blockchain. Three strikes and they get *partially ejected*, in which case they can send a message to the blockchain to be reactivated.

4 Secure Proof of Stake Strategy

Golden Gate is a *Proof of Stake* (PoS) platform. In proof of stake, validators deposit tokens and earn rewards for running consensus. Anyone may delegate their stake to validators and also earn rewards.

Golden Gate will run *Permissioned Proof of Stake* (PPoS) to start as a security measure. PPoS only allows specific validators to operate the network. Additionally, PPoS weights all nodes the same for the purposes of consensus.

For a node to join the network, they need greater than $\frac{2}{3}$rds of the validators to grant them permission. Golden Gate employs PPoS to ensure the distribution of validators is worldwide. It also provides security against a network takeover. In particular, it protects against an economic adversary as well as physical coercion.

Once achieving *economic security* and *decentralization*, Golden Gate will transition to *Nominated Proof of Stake* (NPoS). In contrast to PPoS, in NPoS users *approve* which validators they would delegate to. Golden Gate is the first chain to implement the `Phragmms`[9] algorithm, which balances delegated stake evenly given delegator approvals. Prior to achieving the two safety goals, the blockchain will stage perspective validators. After meeting the two objectives, the perspective validators will become the actual validators.

Economic security requires users to stake most of the total supply. If an adversary gains control of $\frac{2}{3}$rds of the stake, they can compromise the network. Such an event could result in the loss of user data or funds. Requiring a lock-up of most of the tokens stops such bad actors.

Decentralization requires an even distribution of stake across the nodes. The network is fragile if only a handful of nodes control $\frac{2}{3}$rds of the stake. The *Nakamoto Coefficient*[34] is the minimal number of nodes controlling $\frac{2}{3}$rds of the stake. Golden Gate requires a high Nakamoto Coefficient to transition to PoS. As discussed, the `Phragmms` weight balancing rule helps to achieve this goal. In this weight balancing system, users delegate to multiple potential validators.

The `Phragmms` algorithm takes user's delegations and

1. maximizes the total amount at stake,
2. maximizes the stake behind the minimally staked validator, and finally
3. minimizes the variance of the stake across all validators.

`Phragmms` outputs a representative committee that is provably within a factor of 3.15 of the theoretical optimal given by equation (1).

Below we give a formal presentation of `Phragmms` following [9]. First, we assume two finite sets of V voters and C candidates. Further, each voter *approves* of a set of candidates, given by a function $\alpha : V \to 2^C$. Each voter also has a *weight* given by $s : V \to \mathbb{N}$. Assume there are k validator seats. Our goal is to elect a committee $A \subseteq C$ such that $|A| = k$ and A maximizes the minimal subcommittee of potentially Byzantine validators. To this end we look to solve the following optimization problem, called the *maximin support objective*:

$$\max_{A \subseteq C, |A|=k} \left(\min_{A' \subseteq A, A' \neq \varnothing} \frac{1}{|A'|} \sum_{v \in V : \alpha(v) \cap A' \neq \varnothing} s(v) \right) \tag{1}$$

An equivalent formulation of the above may be given in terms of *feasible* weight vectors $w : V \times C \to \mathbb{R}^{\geq 0}$. We say a weight vector w is feasible if and only if:

$$\sum_{c \in \alpha(v)} w(v,c) \leq s(v) \text{ for all } v \in V$$

In addition, define the *support* for a weight-vector to be

$$\text{supp}_w(c) := \sum_{v \in V : c \in \alpha(v)} w(v,c).$$

By abuse of notation we may extend this to committees:

$$\text{supp}_w(A) := \min_{c \in A} \text{supp}_w(c).$$

The following result characterizes the significance of supp:

Proposition 1. *For any committee A*

$$\max_{\text{feasible } w} \text{supp}_w(A) = \min_{A' \subseteq A, |A'| \neq \varnothing} \frac{1}{|A'|} \sum_{v \in V : \alpha(v) \cap A' \neq \varnothing} s(v)$$

Hence, finding the optimal committee A where $|A| = k$ that maximizes $\max_{\text{feasible } w} \text{supp}_w(A)$ is exactly the maximin support objective problem given in (1).

As discussed, we also wish to minimize the variance of the stake behind the validators. This concept is captured by the notion of *balance*.

We say a feasible w is *balanced* for A if, for for all feasible vectors w':

1. $\sum_{c \in A} \text{supp}_{w'}(c) \leq \sum_{c \in A} \text{supp}_w(c)$ and
2. $\sum_{c \in A} \text{supp}^2_{w'}(c) \geq \sum_{c \in A} \text{supp}^2_w(c)$.

Proposition 2. *Given committee size limit k, there exists an algorithm* Bal *that finds the nearest balanced vector to a vector w with an upper bound time complexity of $\Omega(\log k \cdot |\{(v,c) \in V \times C : c \in \alpha(v)\}|)$.*

Next, we define a scoring system for unelected candidates. A candidate's *score* is defined as:

$$\text{score}(A, w, c) :=$$
$$\max \left\{ t \geq 0 : \sum_{v \in V : c \in \alpha(v)} \left(s(v) - \sum_{c \in A \cap \alpha(v)} w(v,c) \cdot \min \left\{ 1, \frac{t}{\text{supp}_w(c)} \right\} \right) \geq t \right\}$$

With this definition of score we may give the `Phragmms` algorithm, which is the following iterative procedure:

1. Initialize an empty committee A and zero weight vector w.
2. Repeat k times:
   - Find the unelected candidate with the highest score and add it to committee A.
   - Re-balance the weight vector w for the new committee A using Bal.
3. Return A and w.

The key theorem behind `Phragmms` may be stated as follows:

Theorem 1. `Phragmms` *is guaranteed to yield a committee* A *where* $\max_{\text{feasible } w} \text{supp}_w(A)$ *is within a factor of 3.15 of the maximin support objective in equation* (1).

Proof. See Lemma 22 in [9]. □

Furthermore, we assume an additional constraint that each committee A must obey *Proportional Justified Representation* (PJR)[30]. A committee $A \subseteq C$ of k members satisfies PJR if and only if, for any group $V' \subseteq V$ of voters and any integer $0 < r \leq k$ we have $r \leq \left|\bigcap_{v \in V'} \alpha(v)\right|$ and $r \cdot \frac{\sum_{v \in V} s(v)}{k} \leq \sum_{v \in V'} s(v)$ jointly imply $r \leq \left|A \cap \left(\bigcup_{v \in V'} \alpha(v)\right)\right|$. That is to say, whenever there is a group of voters V' that approve of r candidates in common, and enough aggregate voting power to give each of these candidates support $\frac{\sum_{v \in V} s(v)}{k}$, then V' is entitled to r representatives they have approved (although perhaps not *commonly* approved).

Theorem 2. `Phragmms` *satisfies the PJR.*

Proof. See Lemma 17 in [9]. □

5 Scalability

Golden Gate uses a Substrate[36] based *relay chain* architecture to future-proof its scalability. Blockchains frequently suffer scaling issues as traffic increases. There are several approaches to tackling this problem. A relay chain is a blockchain that secures other associated blockchains, referred to as *parachains*. The parachains operate independently. Compared to the main chain, parachains may run at high speed. When traffic becomes excessive, validators can launch a new parachain to reduce congestion.

Golden Gate aims to bridge to as many other relay chains as possible. This is a rejection of the "principle of the minimum number of relay chains"[27],

which threatens to create walled gardens of blockchain that refuse to peer[5]. Relay chains have first-class support for bridging to other relay chains. Bridging in this fashion allows for efficient parachain-to-parachain intercommunication. Communication may occur even across separate relay-chain networks. Golden Gate's open policy maximizes its reach as a blockchain interoperability platform.

6 Virtual Machine

Golden Gate differs from other Layer 0 protocols[6] by having an environment for running programs.

Golden Gate supports the *Ethereum Virtual Machine* (EVM), *WebAssembly* (WASM), and native zero-knowledge verification. EVM support means that developers can write in either Solidity or Vyper. On the other hand, WASM support allows authors to write in Rust or any language that compiles to WebAssembly. Golden Gate accomplishes this by using AStar's XVM pallet[2].

Golden Gate supports multiple VM environments for better-decentralized finance software. For example, suppose a Decentralized Autonomous Organization (DAO) wanted to air-drop a token with an exponential vesting schedule. The DAO would have 50% of the token vested in 2 years, 75% in 4 years, etc. Because blockchain VMs do not support floating-point, these calculations require fixed-point arithmetic. While a fixed-point library exists in Solidity for use with the EVM[38], Rust has higher-quality libraries for fixed-point arithmetic[33]. In other circumstances, EVM support is preferable.

Native zero-knowledge verification means authors may write zero-knowledge programs with efficient on-chain support. Expensive gas costs for zero-knowledge verification present a barrier to widespread zero-knowledge adoption. For instance, verifying a zkSNARK for a batch of Ed25519 signatures on Ethereum costs 300K gas[15]. Golden Gate's solution is to provide zero-knowledge verification precompiles. It provides EVM-based precompiles for Groth16[18] and Plonk[11] proof systems. Both precompiles take the following inputs as `bytes`:

1. `verification_key`: the key used for verifying the correctness of the zkSNARK proof
2. `proof`: the zero-knowledge proof, which encodes any private inputs
3. `public_inputs`: an ELN-encoded list of public inputs used by the proof

Aside from basic capabilities, the Golden Gate VM provides block headers of other blockchains for proving data availability. Golden Gate accomplishes this by running light clients to other blockchains. This functionality is part of Golden Gate's support for the LayerZero protocol, as discussed in §2.3.

[5]Some readers may be surprised to learn that ISPs refusing to connect is a problem for the traditional internet, even at the time of this writing. Cogent (AS174) and Hurricane Electric (AS6939) do not peer with one another over IPv6. An interested reader can visit the ISPs' *looking glass* sites and attempt to ping (using IPv6) the other to confirm. See `https://www.cogentco.com/en/looking-glass` and `http://lg.he.net/` respectively.

[6]For instance, Axelar[4, 3] lacks a programmable virtual machine.

7 Summary

Golden Gate is a novel Layer 0 providing several value propositions.

Along with its token GGX, the utility of Golden Gate may be summarized as follows:

1. GGX is the currency of the courier network for running cross-chain transactions.
2. Golden Gate is a smart contract platform.
3. Users are rewarded with GGX for staking and securing the network.

Golden Gate solves the challenges of interchain infrastructure by bringing new innovations to the table. These include scalability, orchestration and secure zero-knowledge verification. Golden Gate sets the bar for infrastructure solutions to follow.

8 Bibliography

[1] Handan Kilinc Alper. *BABE*. Research at W3F. Oct. 8, 2021. URL: `https://research.web3.foundation/en/latest/polkadot/block-production/Babe.html` (visited on 11/28/2022).

[2] *Astar-Frame*. Astar Network, Feb. 6, 2023. URL: `https://github.com/AstarNetwork/astar-frame` (visited on 02/21/2023).

[3] "Axelar Network: Connecting Applications with Blockchain Ecosystems". Jan. 2021. URL: `https://axelar.network/axelar_whitepaper.pdf` (visited on 12/20/2022).

[4] *Axelar: A Decentralized Blockchain Interoperability Network*. Version 0.29.0. Axelar Network, Dec. 1, 2022. URL: `https://github.com/axelarnetwork/axelar-core` (visited on 12/14/2022).

[5] Paulo S. L. M. Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*. 2005. URL: `https://eprint.iacr.org/2005/133` (visited on 12/01/2022).

[6] Daniel J. Bernstein et al. *Twisted Edwards Curves*. 2008. URL: `https://eprint.iacr.org/2008/013` (visited on 12/27/2022).

[7] Guido Bertoni et al. "The Keccak SHA-3 Submission". Jan. 14, 2011. URL: `https://keccak.team/files/Keccak-submission-3.pdf` (visited on 11/28/2022).

[8] Daniel R L Brown. "SEC 2: Recommended Elliptic Curve Domain Parameters". Jan. 27, 2010. URL: `https://www.secg.org/sec2-v2.pdf` (visited on 12/27/2022).

[9] Alfonso Cevallos and Alistair Stewart. "A Verifiably Secure and Proportional Committee Election Rule". In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. Sept. 26, 2021, pp. 29–42. DOI: 10.1145/3479722.3480988. arXiv: 2004.12990 [cs]. URL: http://arxiv.org/abs/2004.12990 (visited on 12/08/2022).

[10] Peter Czaban. *XCMP Overview*. Research at W3F. Nov. 22, 2022. URL: https://research.web3.foundation/en/latest/polkadot/XCMP/index.html (visited on 11/22/2022).

[11] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive Arguments of Knowledge*. 2019. URL: https://eprint.iacr.org/2019/953 (visited on 02/21/2023).

[12] Rosario Gennaro and Steven Goldfeder. *Fast Multiparty Threshold ECDSA with Fast Trustless Setup*. 2019. URL: https://eprint.iacr.org/2019/114 (visited on 03/13/2023).

[13] Rosario Gennaro, Daniele Micciancio, and Tal Rabin. "An Efficient Non-Interactive Statistical Zero-Knowledge Proof System for Quasi-Safe Prime Products". In: *5th ACM Conference on Computer and Communication Security (CCS'98)*. ACM. San Francisco, California: ACM Press, Nov. 1998, pp. 67–72.

[14] Rosario Gennaro et al. "Robust Threshold DSS Signatures". In: *Advances in Cryptology — EUROCRYPT '96*. Ed. by Ueli Maurer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1996, pp. 354–371. ISBN: 978-3-540-68339-1. DOI: 10.1007/3-540-68339-9_31.

[15] Garvit Goel, Rahul Ghangas, and Jinank Jain. *Verify Ed25519 Signatures Cheaply on Eth Using ZK-Snarks*. Ethereum Research. July 25, 2022. URL: https://ethresear.ch/t/verify-ed25519-signatures-cheaply-on-eth-using-zk-snarks/13139 (visited on 02/21/2023).

[16] Christopher Goes. *The Interblockchain Communication Protocol: An Overview*. June 29, 2020. DOI: 10.48550/arXiv.2006.15918. arXiv: 2006.15918 [cs]. URL: http://arxiv.org/abs/2006.15918 (visited on 12/30/2022).

[17] Lorenzo Grassi et al. *Poseidon: A New Hash Function for Zero-Knowledge Proof Systems*. 2019. URL: https://eprint.iacr.org/2019/458 (visited on 11/29/2022).

[18] Jens Groth. *On the Size of Pairing-based Non-interactive Arguments*. 2016. URL: https://eprint.iacr.org/2016/260 (visited on 11/21/2022).

[19] Mike Hamburg. *The STROBE Protocol Framework*. 2017. URL: https://eprint.iacr.org/2017/003 (visited on 11/28/2022).

[20] Kewei Hou and Tobias J. Moskowitz. "Market Frictions, Price Delay, and the Cross-Section of Expected Returns". In: *The Review of Financial Studies* 18.3 (2005), pp. 981–1020. ISSN: 0893-9454. JSTOR: 3598084.

[21] Jun Kimura. *IBC-Solidity*. Version v0.2.4. Hyperledger Labs, Dec. 27, 2022. URL: https://github.com/hyperledger-labs/yui-ibc-solidity (visited on 12/29/2022).

[22] Gerrit Köchling, Janis Müller, and Peter N. Posch. "Price Delay and Market Frictions in Cryptocurrency Markets". In: *Economics Letters* 174 (Jan. 1, 2019), pp. 39–41. ISSN: 0165-1765. DOI: 10.1016/j.econlet.2018.10.025. URL: https://www.sciencedirect.com/science/article/pii/S0165176518304361 (visited on 01/04/2023).

[23] *LayerZero*. Version 1.0.53. LayerZero-Labs, Sept. 7, 2022. URL: https://github.com/LayerZero-Labs/LayerZero (visited on 12/14/2022).

[24] Glenn M. Lilly. "Device for and Method of One-Way Cryptographic Hashing". U.S. pat. 6829355B2. Us National Security Agency. Dec. 7, 2004. URL: https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US6829355 (visited on 11/28/2022).

[25] Patrick McCorry et al. *SoK: Validating Bridges as a Scaling Solution for Blockchains*. 2021. URL: https://eprint.iacr.org/2021/1589 (visited on 12/30/2022).

[26] Tobias Oberstein. *EIP-665: Add Precompiled Contract for Ed25519 Signature Verification*. Ethereum Improvement Proposals. Mar. 25, 2018. URL: https://eips.ethereum.org/EIPS/eip-665 (visited on 12/29/2022).

[27] Polkadot Alliance. "Charter for a Polkadot Alliance". Nov. 28, 2022. URL: https://ipfs.io/ipfs/QmdBBY5JTzazy4zK3ZP4dMXVnM5XXo3QfhEkMrZEzxBFn6?filename=Polkadot_Alliance_Charter_v1_28Nov2022.pdf (visited on 12/16/2022).

[28] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/359340.359342. URL: https://dl.acm.org/doi/10.1145/359340.359342 (visited on 03/15/2023).

[29] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. Request for Comments RFC 7693. Internet Engineering Task Force, Nov. 2015. 30 pp. DOI: 10.17487/RFC7693. URL: https://datatracker.ietf.org/doc/rfc7693 (visited on 11/28/2022).

[30] Luis Sánchez-Fernández et al. "Proportional Justified Representation". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1 (Feb. 10, 2017). ISSN: 2374-3468. DOI: 10.1609/aaai.v31i1.10611. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10611 (visited on 01/05/2023).

[31] C. P. Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (Jan. 1, 1991), pp. 161–174. ISSN: 1432-1378. DOI: 10.1007/BF00196725. URL: https://doi.org/10.1007/BF00196725 (visited on 03/14/2023).

[32] Leo Schwartz. *The 5 Biggest Crypto Hacks of 2022*. Fortune. Dec. 30, 2022. URL: https://fortune.com/crypto/2022/12/30/5-biggest-crypto-hacks-2022/ (visited on 01/04/2023).

[33] Trevor Spiteri. *Fixed*. Version 1.22.0. Feb. 19, 2023. URL: https://gitlab.com/tspiteri/fixed (visited on 02/21/2023).

[34] Balaji S. Srinivasan. *Quantifying Decentralization*. Medium. Oct. 31, 2017. URL: https://news.earn.com/quantifying-decentralization-e39db233c28e (visited on 12/09/2022).

[35] Alistair Stewart and Eleftherios Kokoris-Kogia. *GRANDPA: A Byzantine Finality Gadget*. July 3, 2020. DOI: 10.48550/arXiv.2007.01560. arXiv: 2007.01560 [cs]. URL: http://arxiv.org/abs/2007.01560 (visited on 11/28/2022).

[36] *Substrate*. Parity Technologies, Dec. 5, 2022. URL: https://github.com/paritytech/substrate (visited on 12/05/2022).

[37] *Substrate IBC Pallet (Work in Progress)*. Octopus Network, Nov. 23, 2022. URL: https://github.com/octopus-network/substrate-ibc (visited on 11/25/2022).

[38] Mikhail Vladimirov. *ABDK Libraries for Solidity*. ABDK, Feb. 21, 2023. URL: https://github.com/abdk-consulting/abdk-libraries-solidity (visited on 02/21/2023).

[39] Gavin Wood. *Polkadot Cross-Consensus Message (XCM) Format*. Version 3. Parity Technologies, Dec. 5, 2022. URL: https://github.com/paritytech/xcm-format (visited on 12/29/2022).

[40] Ryan Zarick, Bryan Pellegrino, and Caleb Banister. *LayerZero: Trustless Omnichain Interoperability Protocol*. Oct. 26, 2021. DOI: 10.48550/arXiv.2110.13871. arXiv: 2110.13871 [cs]. URL: http://arxiv.org/abs/2110.13871 (visited on 12/30/2022).

A Datasheet

A.1 Cryptographic Primitives

In Golden gate, we use the following cryptographic primitives:

- Hash Functions – Table 1
- Elliptic Curves – Tables 2 and 3
- Cryptographic Signature Protocols – Table 4

Hash function	Description	Usage
Blake2b[29]	High-speed hash function optimized for 64-bit platforms	All messages, blocks, and Merkle tree states
Strobe128[19]	A hash function based on Keccak-f(1600) that provides a hashing interface well suited to signatures and non-interactive zero-knowledge proofs	Sr25519 key scheme
Poseidon[17]	A hash function that optimizes Rank 1 Constraint Systems (R1CS) representation	Zero-knowledge proof protocols
Keccak256[7]	Ethereum hash function	Ethereum interoperability
SHA256[24]	Bitcoin hash function	Bitcoin interoperability

Table 1: Hash Functions

Curve	Description	Usage
Curve25519[6]	Curve for Ed25519 and Sr25519 signature schemes	Transactions signature
Secp256k1[8]	Bitcoin native curve	Bitcoin and Ethereum interoperability
BN256[5]	Curve with native support in the EVM	Zero-knowledge proofs

Table 2: Elliptic Curves

Curve	Equation	Finite field
Curve25519	$y^2 = x^3 + 486662x^2 + x$	$p = (2^{255} - 19)$
Secp256k1	$y^2 = x^3 + 7$	$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
BN256	$y^2 = x^3 + 3$	$p = 21888242871839275222246405745257275088696311157297823662689037894645226208583$

Table 3: Elliptic Curve Parameters

Cryptographic key	Description	Usage
Ed25519	EdDSA scheme using Schnorr signatures over Curve25519	Block finalization
Sr25519	Alternate Schnorr signature scheme over Curve25519	*Hierarchical Deterministic Key Derivation* (HDKD) implementation – also used in block production
Secp256k1	ECDSA signature scheme over Secp256k1	Transactions and transfers signature

Table 4: Cryptographic Signature Protocols

A.2 Account Model

A private key represents an account. The account model supports the following private keys: Ed25519, Sr25519, and Secp256k1. In addition, Golden Gate offers multi-signature and proxy accounts that we will discuss in depth alongside a classic account.

The Golden Gates uses a Substrate-based SS58 format. The address consists of two parts:

- Prefix - determines how to validate the address and shows the address network.
- Address - address data

A.3 Account Balances

Each account consists of several balances that give the user relevant info about his activities and token usage.

We require the user to have at least one token to prove that the account is active. We suspend and remove accounts with a balance of less than one token.

Each account contains different types of balance depending on account activity:

Balance type	Description
Total	Tokens in the account. The balance does not represent available funds
Transferable	Tokens available for use
Vested	Tokens sent to the account. GoldenGate will release tokens after some verification time controlled in blocks
Bonded	Tokens locked for staking
Democracy	Tokens locked for governance activity
Redeemable	Tokens available for unlocking. These tokens have passed the lock period and are available for usage
Locked	Tokens frozen for on-chain activities. The locks do not stack, so Golden Gate suspends the biggest lock. The difference between the current lock and the next biggest became redeemable when Golden Gate unlocks funds
Reserved	Tokens locked and not relevant to governance, staking, or vesting

A.4 Proxy Accounts

Any account can create a proxy account with limited or full access to the main one. The proxy account can do transactions on behalf of the creator with a limitation or not.

This approach helps to limit transactions made by the primary account and keeps it more secure. The primary account can remove or change the proxy if the user cannot access it anymore. This helps to come up with more granular security practices.

The bare lock deposit for identity consists of two constants:

- The `ProxyDepositBase` is the default deposit for setting up proxy accounts.
- The `ProxyDepositFactor` is the deposit for each used proxy.

The Golden Gate will unlock funds once the user removes proxies.

Golden Gate supports the following proxy types:

Proxy type	Transaction types
Any	All. Gives all rights to the proxy account. It does not give any security benefits, so the user should avoid it
Non-transfer	All, except balance transfers and vested transfers
Governance	Governance-related
Staking	Staking-related
Identity judgement	Transactions for registrars to judge an account's identity
Auction	Transactions for participation in para-chain auctions and crowd loans
Time-delayed	Time-delayed transactions. The proxy will announce its intended action and wait for the number of blocks defined in the delay before executing it. Also, it will include the hash of the intended function call in the announcement. The user can cancel the intended action by primary account or cancel-proxy within this time window
Cancel	Transactions for accounts to reject and remove any time-delay proxy announcements

A.5 Multi-Signature Accounts

Golden Gate supports multi-signature accounts. The multi-signature account consists of one or more addresses and the threshold. The threshold defines how many approvals Golden Gate requires to sign a transaction.

The account should have a locked deposit to participate in the network. The deposit consists of two constants:

- The deposit base is the deposit for the usage of a multi-sig account.
- The threshold deposit is for each approval required to sign a transaction.

A.6 Transaction Model

In Golden Gate, we call Transactions/state changes that are included into the block extrinsic.

There are three different extrinsic transaction types:

- Signed transaction - must include the signature of an account sending it; signed user should pay an execution fee.
- Unsigned transaction
- Inherent transaction - a particular case of unsigned transaction. The creator node is the only one that can add information to a block. Most of the data inserted by this type of transaction are assumed to be valid without validation.

A.7 Block Structure

A Golden Gate block consists of a header and a body.

The block body contains a list of extrinsic transactions included in the block.

The block header contains the following data:

Field name	Description
Parent hash	32-byte Blake2b hash of the parent block
Block height	An integer representing a block. The genesis block is a 0
State root	Merkle tree root hash after applied transactions
Staking	Staking-related
Transaction root	Cryptographic digest of the transaction series
Digest	Any chain-specific auxiliary data. It contains consensus-related data, including the block signature

A.8 Consensus

Golden Gate is a substrate-based chain that leverages hybrid consensus. Block production is separated from finalization to achieve fast chain growth with secure finality. Fast block production is achieved using round robin, although Golden Gate may switch to probabilistic block production with `BABE`[1] in the future. For efficient and secure block finalization, we are using the `GRANDPA`[35] protocol. The key principle behind this design is to get security guarantees level similar to instant-finality consensus while having block production speed similar to probabilistic safety consensus.

Estimated consensus parameters for this approach:

Parameter	Value
Time to finalization	12-60s
Block production time	6s
Block size	< 5 MiB
Estimated TPS	1000