EXHIBIT 3

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 1	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
A computing device for processing a transaction between a first client device, and a second client device via a transfer mechanism, the transfer mechanism comprising a decentralized digital currency, the computing device comprising:	The **Computing Device | Facilitator** includes: • the Coinbase (Owned, managed) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed) Ethereum supporting Archive Nodes and Light Nodes; and **Client Device** The First Client is an active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. The Second Client is a prospective Validator Node, which will include any prospective Coinbase (Owned, managed) Ethereum Validator Nodes. The Coinbase (Owned, managed) Ethereum Validator Nodes facilitates value transfer to newly activated Validator Nodes (as Second Client) based on work performed in producing blocks securely. The Beacon Chain upgrade brings proof-of-stake consensus to Ethereum. For this, active participants - known as validators – are required to propose, verify, and vouch for the validity of blocks. The Patent allows for, and the Claims do not prevent, the Computing Device from being, or including, the First Client or Second Client. This is detailed in the Patent description [0055]. *FIG. 1 (see Figure 16) depicts a typical embodiment for practicing the invention—especially for use with a distributed transfer mechanism—where the clients, transfer mechanism, facilitator, and data source are distinct participants. However, the depicted arrangement is not the only one contemplated by the invention. In an alternate embodiment, the facilitator provides some or all aspects of the transfer mechanism. In another embodiment, the facilitator comprises some or all aspects of a client. For example, part or all of a client's data store, the ability to initiate or accept offers, etc., could be "embedded" in the facilitator, thereby enabling the facilitator to operate as a client itself (e.g., one controlled by the owners of the facilitator, or on behalf of a third party who has entrusted control to the facilitator). In yet another embodiment, the facilitator comprises the data source. Many configurations are contemplated by the invention are possible, and will become apparent to one skilled in the art.*

EXHIBIT 3

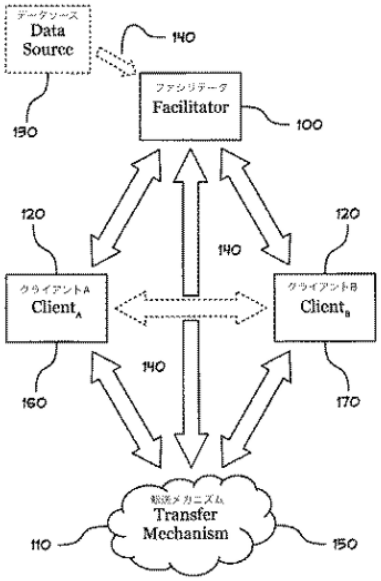
	
• *a memory for storing a first asymmetric key pair, the first asymmetric key pair comprising a first private key and a first public key;*	The First Client is an <u>active</u> Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. These consist of a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer, server or server cloud instance). • Transaction record sector (stores transactions and data that haven't been submitted to the blockchain yet) • a first key pair sector which is generated and stored on the device (typically) • The asymmetric key pair generated and/or stored consists of a first private key and a first public key and is stored on the device (typically) Recommended hardware requirements for running a node. https://launchpad.ethereum.org/en/checklist A description of the Validator Node's keys is described in the following link. https://kb.beaconcha.in/ethereum-2-keys *The validator signing key consists of two elements:* 1. *Validator **private** key* 2. *Validator **public** key* *The purpose of the **validator private key** is to **actively** sign on-chain (ETH2) operations such as block proposals and attestations. Therefore, these keys have to be held in a hot wallet.*

EXHIBIT 3

	Example Validator Client (Prysmatic) installation guide shows key management. https://docs.prylabs.network/docs/install/install-with-script/
• *a network interface for receiving terms, the terms comprising:*	The First Client is an active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. The Ethereum Network requires network connectivity in order to achieve PoS consensus. https://launchpad.ethereum.org/en/checklist https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#attesting https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#broadcast-attestation https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#broadcast-aggregate
○ *at least one of a first principal data or a second principal data;*	**First principle data** The initial deposit staking amount to participate as an Eth2 Validator in the Ethereum Network. *The Beacon Chain upgrade brings proof-of-stake consensus to Ethereum. For this, we need active participants - known as validators - to propose, verify, and vouch for the validity of blocks. In exchange, honest validators receive financial rewards. Importantly, as a validator you'll need to post ETH as collateral - in other words, have some funds at stake. The only way to become a validator is to make a one-way ETH transaction to the deposit contract on the current Ethereum chain.* https://launchpad.ethereum.org/en/overview https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#deposits
○ *a reference to at least one of a first data source or a second data source; and*	**First data source** The Beacon Chain reward and penalty algorithm. https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://kb.beaconcha.in/rewards-and-penalties#block-reward https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#attestations
○ *an expiration timestamp;*	At the beginning of each epoch (every 32 slots, except GENESIS), several things happen, including

EXHIBIT 3

	• Justification and finalization of the chain • Assignment of rewards and penalties to attesters • Update of the validator registry • The special slashing penalty (see above), and • Some final updates (computing effective balances, resets, etc) https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#epoch-processing
• *a computer processor coupled to the memory and the network interface, the computer processor configured to:*	The First Client is an active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. These consist of a computer hardware/software combination to run.
• *read the first private key from the memory;*	The active Coinbase (Owned, managed) Ethereum Validator Full Nodes is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1. The validator creates, signs, and broadcasts a block that is a child of parent that satisfies a valid beacon chain state transition. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal ***Preparing for a BeaconBlock*** *To construct a BeaconBlockBody, a block (BeaconBlock) is defined with the necessary context for a block proposal:* ***Slot*** *Set block.slot = slot where slot is the current slot at which the validator has been selected to propose. The parent selected must satisfy that parent.slot < block.slot.* *Note: There might be "skipped" slots between the parent and block. These skipped slots are processed in the state transition function without per-block processing.* ***Proposer index*** *Set block.proposer_index = validator_index where validator_index is the validator chosen to propose at this slot. The private key mapping to state.validators[validator_index].pubkey is used to sign the block.* **BLS public key** Validator public keys are G1 points on the BLS12-381 curve. A private key, privkey, must be securely generated along with the resultant pubkey. This privkey must be "hot", that is, constantly available to sign data throughout the lifetime of the validator.

EXHIBIT 3

	https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#bls-public-key
• *compute a first cryptographic signature from the first private key;*	The active Coinbase (Owned, managed) Ethereum Validator Full Nodes is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1. The validator creates, signs, and broadcasts a block that is a child of parent that satisfies a valid beacon chain state transition. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal ***Preparing for a BeaconBlock*** *To construct a BeaconBlockBody, a block (BeaconBlock) is defined with the necessary context for a block proposal:* ***Slot*** *Set block.slot = slot where slot is the current slot at which the validator has been selected to propose. The parent selected must satisfy that parent.slot < block.slot.* *Note: There might be "skipped" slots between the parent and block. These skipped slots are processed in the state transition function without per-block processing.* ***Proposer index*** *Set block.proposer_index = validator_index where validator_index is the validator chosen to propose at this slot. The private key mapping to state.validators[validator_index].pubkey is used to sign the block.* Signature `signed_block = SignedBeaconBlock(message=block, signature=block_signature)`, where `block_signature` is obtai `def get_block_signature(state: BeaconState, block: BeaconBlock, privkey: int) -> BLSSignature:` `  domain = get_domain(state, DOMAIN_BEACON_PROPOSER, compute_epoch_at_slot(block.slot))` `  signing_root = compute_signing_root(block, domain)` `  return bls.Sign(privkey, signing_root)`
• *create an inchoate data record comprising:*	The SignedBeaconBlock, as produced by the Validator (as the block producer), containing the deposit of a prospective Validator that is yet to be added to the list of registered Validators.
• *a commit input for receiving a commit data from a commit transaction;*	The 32ETH deposit stake to instantiate an Eth2 Validator.
• *one or more output data obtained from at least one of the first principal data or the second principal data, and a value data from at least one of the first data*	The prospective Validator's reward or penalty. This is implied as these are rules of the network. The Beacon Chain reward and penalty algorithm uses predefined values and formulas, along with the effective staking balance of the participating nodes to calculate the reward or penalty for the Validators.

EXHIBIT 3

source or the second data source; and	https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://kb.beaconcha.in/rewards-and-penalties#block-reward https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#attestations
• *the first cryptographic signature; and*	The Validator (as the block producer) is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1. The validator creates, signs, and broadcasts a block that is a child of parent that satisfies a valid beacon chain state transition. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal Signature `signed_block = SignedBeaconBlock(message=block, signature=block_signature)`, where `block_signature` is obtai `def get_block_signature(state: BeaconState, block: BeaconBlock, privkey: int) -> BLSSignature:` `  domain = get_domain(state, DOMAIN_BEACON_PROPOSER, compute_epoch_at_slot(block.slot))` `  signing_root = compute_signing_root(block, domain)` `  return bls.Sign(privkey, signing_root)` Note that the signed block includes the Eth1 data containing the 32ETH deposit stake to instantiate a Validator. **Eth1 Data** The block.body.eth1_data field is for block proposers to vote on recent Eth1 data. This recent data contains an Eth1 block hash as well as the associated deposit root (as calculated by the get_deposit_root() method of the deposit contract) and deposit count after execution of the corresponding Eth1 block. If over half of the block proposers in the current Eth1 voting period vote for the same eth1_data then state.eth1_data updates immediately allowing new deposits to be processed. Each deposit in block.body.deposits must verify against state.eth1_data.eth1_deposit_root. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#eth1-data
• *publish the inchoate data record to at least one of the first client device or the second client device,*	As the Computing Device and the First Client are the same device (Coinbase (Owned, managed) Ethereum Validator Node), the inchoate data record is already known by the First Client. As such the inchoate data record is considered to be published to the First Client on creation of the inchoate data record by the Validator Node. *A validator is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain*

EXHIBIT 3

	during slot - 1. ***The validator creates, signs, and broadcasts a block*** *that is a child of parent that satisfies a valid beacon chain state transition.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal
wherein the decentralized digital currency comprises a distributed ledger that enables processing the transaction between the first client device and the second client device without the need for a trusted central authority,	Ether is used as the decentralised currency. The Ethereum network maintains a distributed ledger without the need for a trusted central authority. From the Ethereum yellow paper. *2.1. Value. In order to incentivise computation within the network, there needs to be an agreed method for transmitting value. To address this issue, Ethereum has an intrinsic currency, Ether, known also as ETH and sometimes referred to by the Old English ¯D.* https://ethereum.github.io/yellowpaper/paper.pdf
wherein at least one of the first client device or the second client device signs the inchoate data record and saves a copy of the inchoate data record on at least one of the first client device or the second client device; and	As the Computing Device and the First Client are the same device (Coinbase (Owned, managed) Ethereum Validator Node), the inchoate data record is already signed by the First Client. The signed inchoate data record (as a complete data record) is broadcast and recorded on the Ethereum Network. *A validator is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1.* ***The validator creates, signs, and broadcasts a block*** *that is a child of parent that satisfies a valid beacon chain state transition.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal
wherein the inchoate data record is used by at least one of the first client device or the second client device to create a complete data record and to create the transaction by broadcasting the complete data record for transmitting and receiving among network participants in the computer network for recording in the distributed ledger,	As the Computing Device and the First Client are the same device (Coinbase (Owned, managed) Ethereum Validator Node), the inchoate data record is already signed by the First Client. The signed inchoate data record (as a complete data record) is broadcast and recorded on the Ethereum Network. The SignedBeaconBlock, in order to achieve consensus, is required to be signed by the Ethereum Network as the First Client (the Eth2 Validators) as either the Block Producer, Attester or Aggregator's. **Proposer index**

EXHIBIT 3

	Set block.proposer_index = validator_index where validator_index is the validator chosen to propose at this slot. The private key mapping to state.validators[validator_index].pubkey is used to sign the block. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#proposer-index **Attesting** *A validator is expected to create, sign, and broadcast an attestation during each epoch. The committee, assigned index, and assigned slot for which the validator performs this role during an epoch are defined by get_committee_assignment(state, epoch, validator_index).* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#attesting **Construct aggregate** *If the validator is selected to aggregate (is_aggregator()), they construct an aggregate attestation via the following.* *Collect attestations seen via gossip during the slot that have an equivalent attestation_data to that constructed by the validator. If len(attestations) > 0, create an aggregate_attestation: Attestation with the following fields.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#construct-attestation def get_aggregate_signature(attestations: Sequence[Attestation]) -> BLSSignature: signatures = [attestation.signature for attestation in attestations] return bls.Aggregate(signatures) _____ The Coinbase (Owned, managed) Ethereum Validator Node uses the SignedBeaconBlock (as a complete data record) along with the broadcasted Aggregated Attestations to update its record of the Beacon Chain. Finality is achieved if two epochs in a row are justified. *ETH2 uses Casper Proof of Stake, specifically, something called a "finality gadget". The ETH2 finality process is defined as follows:* 1. *If > 2/3rds of validators vote correctly on the chain head during an epoch, we call the last epoch **justified*** 2. *If two epochs in a row are justified, the current_epoch - 2 is considered **finalized*** https://hackmd.io/@prysmaticlabs/finality#How-Finality-Works-in-ETH2 _____ The Coinbase (Owned, managed) Ethereum Validator Node as the First Client process the Beacon Block and records the new Validator in the

EXHIBIT 3

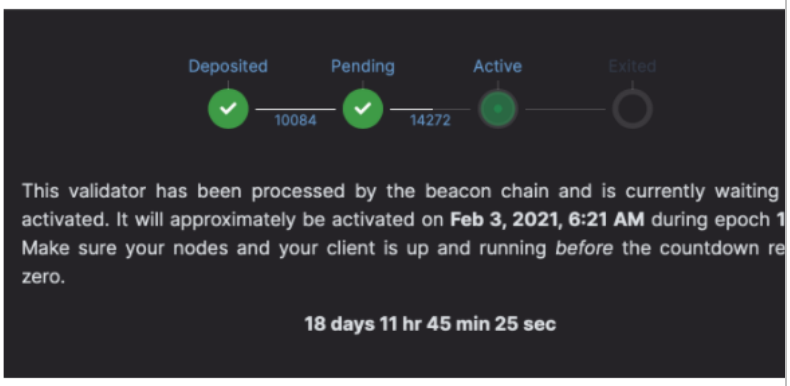
	Registry on identifying a new deposit in the Deposit Contract. The Beacon Block is saved by each Eth2 Client. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#deposits
wherein the at least one of the computing device, the first client device, or the second client device verifies the recording of the complete data record in the distributed ledger by observing an external state.	The new Validator Node, as the Second Client, can view the status of its activation from https://beaconcha.in/  *Once a deposit is made to the Ethereum Beacon Chain, it will join a waiting queue before joining the network.* *All Ethereum 2.0 deposits have two delays before going into the waiting queue:* 1. *Eth1 data inclusion delay: The Beacon Chain follows Eth1 with some delay to make sure the Eth1 data is finalized.* 2. *Eth1 data voting delay: The Beacon Chain validators vote on the deposits to process every 4 hours.* *You can check more details about your validator, including the estimated time to join the network, at https://beaconcha.in/.* https://intercom.help/stakefish/en/articles/4799068-when-will-my-validator-join-the-network-and-be-activated

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 2	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
The device of claim 1, where: *the computer processor is configured to obtain the one or more output data based on:*	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed) Ethereum supporting Archive Nodes and Light Nodes; and **Client Device** The First Client is the active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. The Second Client is the newly activated Validator Node, which will include any prospective Coinbase (Owned, managed or offered) Ethereum Validator Nodes. The Coinbase (Owned, managed) Ethereum Validator Nodes facilitates value transfer to newly activated Validator Nodes (as Second Client) based on work performed in producing blocks securely. The Beacon Chain upgrade brings proof-of-stake consensus to Ethereum. For this, active participants - known as validators – are required to propose, verify, and vouch for the validity of blocks.
the first principal data; and *the value data from the first data source.*	The Coinbase (Owned, managed) Ethereum Validator Full Nodes processes the validator rewards at the end of each epoch. This is calculated from the Beacon Chain reward and penalty algorithm (Data Source) and considers the staked principle. https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://kb.beaconcha.in/rewards-and-penalties#block-reward Each Validator account is updated with the reward/penalty. The Beacon Block is updated accordingly. def process_rewards_and_penalties(state: BeaconState) -> None: # No rewards are applied at the end of `GENESIS_EPOCH` because rewards are for work done in the previous epoch if get_current_epoch(state) == GENESIS_EPOCH: return rewards, penalties = get_attestation_deltas(state) for index in range(len(state.validators)): increase_balance(state, ValidatorIndex(index), rewards[index])

EXHIBIT 3

	decrease_balance(state, ValidatorIndex(index), penalties[index]) https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#process_rewards_and_penalties def process_slashings(state: BeaconState) -> None: epoch = get_current_epoch(state) total_balance = get_total_active_balance(state) adjusted_total_slashing_balance = min(sum(state.slashings) * PROPORTIONAL_SLASHING_MULTIPLIER, total_balance) for index, validator in enumerate(state.validators): if validator.slashed and epoch + EPOCHS_PER_SLASHINGS_VECTOR // 2 == validator.withdrawable_epoch: increment = EFFECTIVE_BALANCE_INCREMENT # Factored out from penalty numerator to avoid uint64 overflow penalty_numerator = validator.effective_balance // increment * adjusted_total_slashing_balance penalty = penalty_numerator // total_balance * increment decrease_balance(state, ValidatorIndex(index), penalty) https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#slashings

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 3	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
The device of claim 1, where the computer processor is further configured to:	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed) Ethereum supporting Archive Nodes and Light Nodes; and **Client Device** The First Client is the active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. The Second Client is the newly activated Validator Node, which will include any prospective Coinbase (Owned, managed) Ethereum Validator Nodes. The Coinbase (Owned, managed) Ethereum Validator Nodes facilitates value transfer to newly activated Validator Nodes (as Second Client) based on work performed in producing blocks securely. The Beacon Chain upgrade brings proof-of-stake consensus to Ethereum. For this, active participants - known as validators – are required to propose, verify, and vouch for the validity of blocks.
compute a second cryptographic signature from the first private key;	The Voluntary Exit is initiated and signed from the Validator Nodes Client CLI. Teku Client *teku voluntary-exit --validator-keys=<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>[,<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>...]...* https://docs.teku.consensys.net/en/latest/Reference/CLI/Subcommands/Voluntary-Exit/ Lighthouse Client *In order to initiate an exit, users can use the lighthouse account validator exit command.* *The --keystore flag is used to specify the path to the EIP-2335 voting keystore for the validator.* *The --beacon-node flag is used to specify a beacon chain HTTP endpoint that confirms to the Eth2.0 Standard Beacon Node API specifications. That beacon node will be used to validate and propagate the voluntary exit. The default value for this flag is http://localhost:5052.* *The --network flag is used to specify a particular Eth2 network (default is mainnet).* *The --password-file flag is used to specify the path to the file containing the password for the voting keystore. If this flag is not provided, the user will be prompted to enter the password.* *After validating the password, the user will be prompted to enter a special exit phrase as a final confirmation after which the voluntary exit will be published to the beacon chain.* https://lighthouse-book.sigmaprime.io/voluntary-exit.html
create an another inchoate data record comprising:	A Signed Voluntary Exit message is created.

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 3	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
a commit input for receiving the commit data from the commit transaction;	The Voluntary Exit is initiated and signed from the Validator Client. Teku Client *teku voluntary-exit --validator-keys=<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>[,<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>...]...* https://docs.teku.consensys.net/en/latest/Reference/CLI/Subcommands/Voluntary-Exit/ Lighthouse Client *In order to initiate an exit, users can use the lighthouse account validator exit command.* *The --keystore flag is used to specify the path to the EIP-2335 voting keystore for the validator.* *The --beacon-node flag is used to specify a beacon chain HTTP endpoint that confirms to the Eth2.0 Standard Beacon Node API specifications. That beacon node will be used to validate and propagate the voluntary exit. The default value for this flag is http://localhost:5052.* *The --network flag is used to specify a particular Eth2 network (default is mainnet).* *The --password-file flag is used to specify the path to the file containing the password for the voting keystore. If this flag is not provided, the user will be prompted to enter the password.* *After validating the password, the user will be prompted to enter a special exit phrase as a final confirmation after which the voluntary exit will be published to the beacon chain.* https://lighthouse-book.sigmaprime.io/voluntary-exit.html
a refund output comprising a refund data;	The Voluntary Exit message contains a reference to the ValidatorIndex which is a pointer to the account. The refund output is the account balance. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#voluntary-exits def process_voluntary_exit(state: BeaconState, signed_voluntary_exit: SignedVoluntaryExit) -> None: voluntary_exit = signed_voluntary_exit.message validator = state.validators[voluntary_exit.validator_index] # Verify the validator is active assert is_active_validator(validator, get_current_epoch(state)) # Verify exit has not been initiated assert validator.exit_epoch == FAR_FUTURE_EPOCH # Exits must specify an epoch when they become valid; they are not valid before then assert get_current_epoch(state) >= voluntary_exit.epoch # Verify the validator has been active long enough assert get_current_epoch(state) >= validator.activation_epoch + SHARD_COMMITTEE_PERIOD # Verify signature domain = get_domain(state, DOMAIN_VOLUNTARY_EXIT, voluntary_exit.epoch) signing_root = compute_signing_root(voluntary_exit, domain) assert bls.Verify(validator.pubkey, signing_root, signed_voluntary_exit.signature)

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 3	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
	# Initiate exit initiate_validator_exit(state, voluntary_exit.validator_index)
the second cryptographic signature; and	The Signed Voluntary Exit message contains the signature from the Validator. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#signedvoluntaryexit class SignedVoluntaryExit(Container): message: VoluntaryExit signature: BLSSignature
a lock time; and	One of the main design decisions of the Ethereum 2 project is performing the roll-out of the system over several phases. This decision has a serious impact on voluntary exits. Even though validators are able to perform an exit in Phase 0 and Phase 1, they will have to wait until Phase 2 to be able to withdraw. This means their staked funds will be frozen until withdrawals are available, which should be around 2 years after Mainnet launch. https://docs.prylabs.network/docs/wallet/exiting-a-validator/
publish the another inchoate data record to at least one of the first client device or the second client device.	As the Computing Device and the First Client are the same device (Coinbase Ethereum Validator Node), the inchoate data record is already signed by the First Client. As such the inchoate data record is considered to be published to the First Client on creation of the inchoate data record by the Validator Node. The signed inchoate data record (as a complete data record) is broadcast and recorded on the Ethereum Network. The following is an example output message from the Lighthouse Client. Successfully published voluntary exit for validator 0xabcd Voluntary exit has been accepted into the beacon chain, but not yet finalized. Finalization may take several minutes or longer. Before finalization there is a low probability that the exit may be reverted. Current epoch: 29946, Exit epoch: 29951, Withdrawable epoch: 30207 Please keep your validator running till exit epoch Exit epoch in approximately 1920 secs. https://lighthouse-book.sigmaprime.io/voluntary-exit.html

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 7 A system for processing a transaction between a first client device and a second client device via a transfer mechanism the system comprising a computing device, the first client device, the second client device, and the transfer mechanism.	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
7. a. *the computing device comprising:* i. *a first memory comprising for storing a first asymmetric key pair, the first asymmetric key pair comprising a first private key and a first public key;*	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed) Ethereum supporting Archive Nodes and Light Nodes; and **Client Device** The First Client is an active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. These consist of a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer, server or server cloud instance). • Transaction record sector (stores transactions and data that haven't been submitted to the blockchain yet) • a first key pair sector which is generated and stored on the device (typically) • The asymmetric key pair generated and/or stored consists of a first private key and a first public key and is stored on the Eth Node Recommended hardware requirements for running a node. https://launchpad.ethereum.org/en/checklist A description of the Validator's Node keys is described in the following link. https://kb.beaconcha.in/ethereum-2-keys *The validator signing key consists of two elements:* 1. *Validator **private** key* 2. *Validator **public** key* *The purpose of the **validator private key** is to **actively** sign on-chain (ETH2) operations such as block proposals and attestations. Therefore, these keys have to be held in a hot wallet.*

EXHIBIT 3

	Example Validator Client (Prysmatic) installation guide shows key management. https://docs.prylabs.network/docs/install/install-with-script/
ii. *a first network interface for receiving terms, the terms comprising.*	The First Client is an active Coinbase (Owned, Managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. The Ethereum Network requires network connectivity in order to achieve PoS consensus. https://launchpad.ethereum.org/en/checklist https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#attesting https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#broadcast-attestation https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#broadcast-aggregate
A. *At least one of a first principal data or second principle data;*	**First principle data** The initial deposit staking amount to participate as a Eth2 Validator in the Ethereum Network. *The Beacon Chain upgrade brings proof-of-stake consensus to Ethereum. For this, we need active participants - known as validators - to propose, verify, and vouch for the validity of blocks. In exchange, honest validators receive financial rewards. Importantly, as a validator you'll need to post ETH as collateral - in other words, have some funds at stake. The only way to become a validator is to make a one-way ETH transaction to the deposit contract on the current Ethereum chain.* https://launchpad.ethereum.org/en/overview https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#deposits
B. a reference to at least one of a first data source or a second data source; and	**First data source** The Beacon Chain reward and penalty algorithm. https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://kb.beaconcha.in/rewards-and-penalties#block-reward https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#attestations

EXHIBIT 3

C. *an expiration timestamp,*	At the beginning of each epoch (every 32 slots, except GENESIS), several things happen, including: • Justification and finalization of the chain • Assignment of rewards and penalties to attesters • Update of the validator registry • The special slashing penalty (see above), and • Some final updates (computing effective balances, resets, etc) https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#epoch-processing
iii. *a first computer processor coupled to the first memory and the first network interface, the first computer processor configured to:*	The First Client is an active Coinbase (Owned, Managed) Ethereum Validator Full Nodes. The Computing Device and the First Client are the same device. These consist of a computer hardware/software combination to run.
A. *read the first private key from the first memory;*	The Ethereum Validator Full Node is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1. The validator creates, signs, and broadcasts a block that is a child of parent that satisfies a valid beacon chain state transition. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal ***Preparing for a BeaconBlock*** *To construct a BeaconBlockBody, a block (BeaconBlock) is defined with the necessary context for a block proposal:* ***Slot*** *Set block.slot = slot where slot is the current slot at which the validator has been selected to propose. The parent selected must satisfy that parent.slot < block.slot.* *Note: There might be "skipped" slots between the parent and block. These skipped slots are processed in the state transition function without per-block processing.* ***Proposer index*** *Set block.proposer_index = validator_index where validator_index is the validator chosen to propose at this slot. The private key mapping to state.validators[validator_index].pubkey is used to sign the block.* **BLS public key** Validator public keys are G1 points on the BLS12-381 curve. A private key, privkey, must be securely generated along with the

EXHIBIT 3

	resultant pubkey. This privkey must be "hot", that is, constantly available to sign data throughout the lifetime of the validator. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#bls-public-key
B. *compute a first cryptographic signature from the first private key;*	The Ethereum Validator Full Node is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1. The validator creates, signs, and broadcasts a block that is a child of parent that satisfies a valid beacon chain state transition. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal ***Preparing for a BeaconBlock*** *To construct a BeaconBlockBody, a block (BeaconBlock) is defined with the necessary context for a block proposal:* ***Slot*** *Set block.slot = slot where slot is the current slot at which the validator has been selected to propose. The parent selected must satisfy that parent.slot < block.slot.* *Note: There might be "skipped" slots between the parent and block. These skipped slots are processed in the state transition function without per-block processing.* ***Proposer index*** *Set block.proposer_index = validator_index where validator_index is the validator chosen to propose at this slot. The private key mapping to state.validators[validator_index].pubkey is used to sign the block.* Signature `signed_block = SignedBeaconBlock(message=block, signature=block_signature)`, where `block_signature` is obtai `def get_block_signature(state: BeaconState, block: BeaconBlock, privkey: int) -> BLSSignature:` `  domain = get_domain(state, DOMAIN_BEACON_PROPOSER, compute_epoch_at_slot(block.slot))` `  signing_root = compute_signing_root(block, domain)` `  return bls.Sign(privkey, signing_root)`
C. *create an inchoate data record comprising:*	The SignedBeaconBlock, as produced by the Ethereum Network's Validator, containing the deposit of a prospective Validator that is yet to be added to the list of registered Validators.
a. *a commit input for receiving a commit data from a commit transaction;*	The 32ETH deposit stake required to instantiate a prospective Eth2 Validator.
b. one or more outputs obtained from at least one of the first principal data or the second principal data, and a value data from at least one	The prospective Validator's reward or penalty. This is implied as these are rules of the network. The Beacon Chain reward and penalty algorithm uses predefined values and formulas, along with the effective staking balance of the participating nodes to calculate the reward or penalty for the Validators.

EXHIBIT 3

of the first data source or the second data source; and	https://consensys.net/blog/codefi/rewards-and-penalties-on-ethereum-20-phase-0/ https://kb.beaconcha.in/rewards-and-penalties#block-reward https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#attestations
c. *the first cryptographic signature; and*	The Ethereum Network Validator is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1. The validator creates, signs, and broadcasts a block that is a child of parent that satisfies a valid beacon chain state transition. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal Signature `signed_block = SignedBeaconBlock(message=block, signature=block_signature)`, where `block_signature` is obtai `def get_block_signature(state: BeaconState, block: BeaconBlock, privkey: int) -> BLSSignature:` `  domain = get_domain(state, DOMAIN_BEACON_PROPOSER, compute_epoch_at_slot(block.slot))` `  signing_root = compute_signing_root(block, domain)` `  return bls.Sign(privkey, signing_root)` Note that the signed block includes the Eth1 data containing the 32ETH deposit stake to instantiate a Validator. **Eth1 Data** The block.body.eth1_data field is for block proposers to vote on recent Eth1 data. This recent data contains an Eth1 block hash as well as the associated deposit root (as calculated by the get_deposit_root() method of the deposit contract) and deposit count after execution of the corresponding Eth1 block. If over half of the block proposers in the current Eth1 voting period vote for the same eth1_data then state.eth1_data updates immediately allowing new deposits to be processed. Each deposit in block.body.deposits must verify against state.eth1_data.eth1_deposit_root. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#eth1-data
D. *publish the inchoate data record to at least one of the first client device or the second client device;*	As the Computing Device and the First Client are the same device (Coinbase (Owned, managed) Ethereum Validator Node), the inchoate data record is already known by the First Client. As such the inchoate data record is considered to be published to the First Client on creation of the inchoate data record by the Validator Node. *A validator is expected to propose a SignedBeaconBlock at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock,*

EXHIBIT 3

	parent, that in their view of the fork choice is the head of the chain during slot - 1. ***The validator creates, signs, and broadcasts a block*** *that is a child of parent that satisfies a valid beacon chain state transition.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal
7. b. the first client comprises: i. *a second memory for storing a second asymmetric key pair, the second asymmetric key pair comprising a second private key and a second public key.*	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed) Ethereum supporting Archive Nodes and Light Nodes; and **Client Device** The First Client is an active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.i.
ii. *a second network interface; and*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.ii.
iii. *a second computer processor coupled to the second memory and the second network interface, the second computer processor configured to:*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.A-E.
A. *read the second private key from the second key pair sector;*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.A.
B. *read the inchoate disbursement transaction record,*	The Facilitator and the First Client are both considered to be the same device. Thus, the clause is not assessable.
C. *compute a second cryptographic signature from the second private key*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.B.
D. *create a complete data record comprising:* I. *the commit input,*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
II. *the output data,*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
III. *the first cryptographic signature, and*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.

EXHIBIT 3

IV. *the second cryptographic signature,*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
E. *create a transaction by submitting the complete data record to the transfer mechanism.*	The Facilitator and the First Client are the same device. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.D.
7. c. the second client comprises: i. *a third memory for storing a third asymmetric key pair, the third asymmetric key pair comprising a third private key and a third public key.*	The Second Client is a prospective Validator Node, which will include any prospective Coinbase (Owned, managed, offered) Ethereum Validator Nodes. These consist of a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer, server or server cloud instance). • Transaction record sector (stores transactions and data that haven't been submitted to the blockchain yet) • a first key pair sector which is generated and stored on the device (typically) • The asymmetric key pair generated and/or stored consists of a first private key and a first public key and is stored on the device Recommended hardware requirements for running a node. https://launchpad.ethereum.org/en/checklist A description of the Validator Node's keys is described in the following link. https://kb.beaconcha.in/ethereum-2-keys *The validator signing key consists of two elements:* 1. *Validator **private** key* 2. *Validator **public** key* *The purpose of the **validator private key** is to **actively** sign on-chain (ETH2) operations such as block proposals and attestations. Therefore, these keys have to be held in a hot wallet.* Example Validator Client (Prysmatic) installation guide shows key management. https://docs.prylabs.network/docs/install/install-with-script/
ii. *a third network interface; and*	The newly activated Validator requires network connectivity in order to contribute to PoS consensus. https://launchpad.ethereum.org/en/checklist https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal

EXHIBIT 3

	https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#attesting https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#broadcast-attestation https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#broadcast-aggregate
iii. a third computer processor coupled to the third memory and the third network interface, the third computer processor configured to read the third private key from memory; and	The newly activated Validator is required to sign (by reading the locally stored private key) produced blocks, attestations and aggregated attestations. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#signature https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#aggregate-signature https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#aggregate-signature-1
wherein the at least one of the first client device or the second client device signs the inchoate data record and saves a copy of the inchoate data record on at least one of the first client device or the second client device,	As the Computing Device and the First Client are the same device (Coinbase (Owned, managed) Ethereum Validator Node), the inchoate data record is already signed by the First Client. The signed inchoate data record (as a complete data record) is broadcast and recorded on the Ethereum Network. *A validator is expected to propose a* SignedBeaconBlock *at the beginning of any slot during which is_proposer(state, validator_index) returns True. To propose, the validator selects the BeaconBlock, parent, that in their view of the fork choice is the head of the chain during slot - 1.* ***The validator creates, signs, and broadcasts a block*** *that is a child of parent that satisfies a valid* beacon chain state transition*.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#block-proposal
wherein the transfer mechanism comprising a decentralized digital currency that comprises a distributed ledger that enables processing the transaction between the first client device and the second client device without the need of a trusted central authority,	Ether is used as the decentralised currency. The Ethereum network maintains a distributed ledger without the need for a trusted central authority. From the Ethereum yellow paper. *2.1. Value. In order to incentivise computation within the network, there needs to be an agreed method for transmitting value. To address this issue, Ethereum has an intrinsic currency, Ether, known also as ETH and sometimes referred to by the Old English ¯D.* https://ethereum.github.io/yellowpaper/paper.pdf
wherein the transaction is created by broadcasting the complete data record	As the Computing Device and the First Client are the same device (Coinbase (Owned, managed) Ethereum Validator Node), the inchoate

EXHIBIT 3

for transmitting and receiving among network participants in the computer network for recording in the distributed ledger, and	data record is already signed by the First Client. The signed inchoate data record (as a complete data record) is broadcast and recorded on the Ethereum Network. The SignedBeaconBlock, in order to achieve consensus, is required to be signed by the Ethereum Network as the First Client (the Eth2 Validators) as either the Block Producer, Attester or Aggregator's. **Proposer index** *Set block.proposer_index = validator_index where validator_index is the validator chosen to propose at this slot. The private key mapping to state.validators[validator_index].pubkey is used to sign the block.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#proposer-index **Attesting** *A validator is expected to create, sign, and broadcast an attestation during each epoch. The committee, assigned index, and assigned slot for which the validator performs this role during an epoch are defined by get_committee_assignment(state, epoch, validator_index).* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#attesting **Construct aggregate** *If the validator is selected to aggregate (is_aggregator()), they construct an aggregate attestation via the following.* *Collect attestations seen via gossip during the slot that have an equivalent attestation_data to that constructed by the validator. If len(attestations) > 0, create an aggregate_attestation: Attestation with the following fields.* https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md#construct-attestation def get_aggregate_signature(attestations: Sequence[Attestation]) -> BLSSignature: signatures = [attestation.signature for attestation in attestations] return bls.Aggregate(signatures) _____ The Coinbase (Owned, managed) Ethereum Validator Node uses the SignedBeaconBlock (as a complete data record) along with the broadcasted Aggregated Attestations to update its record of the Beacon Chain. Finality is achieved if two epochs in a row are justified. *ETH2 uses Casper Proof of Stake, specifically, something called a "finality gadget". The ETH2 finality process is defined as follows:* 3. *If > 2/3rds of validators vote correctly on the chain head during an epoch, we call the last epoch* ***justified*** 4. *If two epochs in a row are justified, the current_epoch - 2 is considered* ***finalized***

EXHIBIT 3

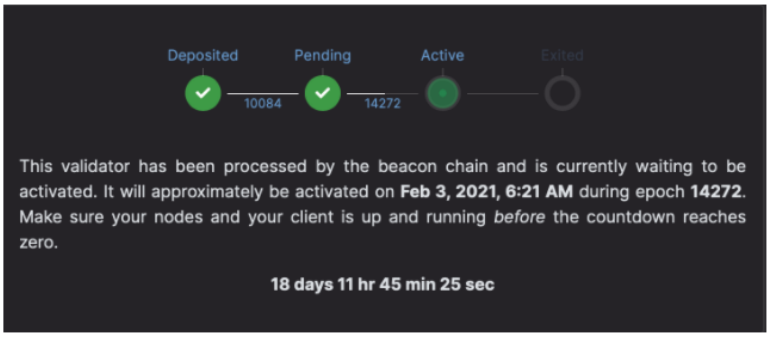
	https://hackmd.io/@prysmaticlabs/finality#How-Finality-Works-in-ETH2 _____ The Coinbase (Owned, managed) Ethereum Validator Node as the First Client process the Beacon Block and records the new Validator in the Registry on identifying a new deposit in the Deposit Contract. The Beacon Block is saved by each Eth2 Client. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#deposits
wherein at least one of the computer device, the first client device, or the second client device verifies the recording of the complete data record in the distributed ledger by observing an external state	The new Validator Node, as the Second Client, can view the status of its activation from https://beaconcha.in/  *Once a deposit is made to the Ethereum Beacon Chain, it will join a waiting queue before joining the network.* *All Ethereum 2.0 deposits have two delays before going into the waiting queue:* 3. *Eth1 data inclusion delay: The Beacon Chain follows Eth1 with some delay to make sure the Eth1 data is finalized.* 4. *Eth1 data voting delay: The Beacon Chain validators vote on the deposits to process every 4 hours.* *You can check more details about your validator, including the estimated time to join the network, at https://beaconcha.in/.* https://intercom.help/stakefish/en/articles/4799068-when-will-my-validator-join-the-network-and-be-activated

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 8	**Coinbase Products & Services** **Payment of block rewards to new Validator Nodes under Proof of Stake**
The system of claim 7, where the first computer processor is further configured to:	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed) Ethereum supporting Archive Nodes and Light Nodes; and **Client Device** The First Client is the active Coinbase (Owned, managed) Ethereum Validator Full Nodes. The Facilitator and the First Client are the same device. The Second Client is the newly activated Validator Node, which will include any prospective Coinbase (Owned, managed) Ethereum Validator Nodes.
compute a third cryptographic signature from the first private key;	The Voluntary Exit is initiated and signed from the Validator Nodes Client CLI. Teku Client *teku voluntary-exit --validator-keys=<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>[,<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>...]...* https://docs.teku.consensys.net/en/latest/Reference/CLI/Subcommands/Voluntary-Exit/ Lighthouse Client *In order to initiate an exit, users can use the lighthouse account validator exit command.* *The --keystore flag is used to specify the path to the EIP-2335 voting keystore for the validator.* *The --beacon-node flag is used to specify a beacon chain HTTP endpoint that confirms to the Eth2.0 Standard Beacon Node API specifications. That beacon node will be used to validate and propagate the voluntary exit. The default value for this flag is http://localhost:5052.* *The --network flag is used to specify a particular Eth2 network (default is mainnet).* *The --password-file flag is used to specify the path to the file containing the password for the voting keystore. If this flag is not provided, the user will be prompted to enter the password.* *After validating the password, the user will be prompted to enter a special exit phrase as a final confirmation after which the voluntary exit will be published to the beacon chain.* https://lighthouse-book.sigmaprime.io/voluntary-exit.html
create another inchoate data record comprising:	A Signed Voluntary Exit message is created.
a commit input for receiving the commit data from the commit transaction;	The Voluntary Exit is initiated and signed from the Validator Client. Teku Client *teku voluntary-exit --validator-keys=<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>[,<KEY_DIR>:<PASS_DIR> | <KEY_FILE>:<PASS_FILE>...]...* https://docs.teku.consensys.net/en/latest/Reference/CLI/Subcommands/Voluntary-Exit/

EXHIBIT 3

	Lighthouse Client *In order to initiate an exit, users can use the lighthouse account validator exit command.* *The --keystore flag is used to specify the path to the EIP-2335 voting keystore for the validator.* *The --beacon-node flag is used to specify a beacon chain HTTP endpoint that confirms to the Eth2.0 Standard Beacon Node API specifications. That beacon node will be used to validate and propagate the voluntary exit. The default value for this flag is http://localhost:5052.* *The --network flag is used to specify a particular Eth2 network (default is mainnet).* *The --password-file flag is used to specify the path to the file containing the password for the voting keystore. If this flag is not provided, the user will be prompted to enter the password.* *After validating the password, the user will be prompted to enter a special exit phrase as a final confirmation after which the voluntary exit will be published to the beacon chain.* https://lighthouse-book.sigmaprime.io/voluntary-exit.html
a refund output comprising a refund data; and	The Voluntary Exit message contains a reference to the ValidatorIndex which is a pointer to the account. The refund output is the account balance. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#voluntary-exits def process_voluntary_exit(state: BeaconState, signed_voluntary_exit: SignedVoluntaryExit) -> None: voluntary_exit = signed_voluntary_exit.message validator = state.validators[voluntary_exit.validator_index] # Verify the validator is active assert is_active_validator(validator, get_current_epoch(state)) # Verify exit has not been initiated assert validator.exit_epoch == FAR_FUTURE_EPOCH # Exits must specify an epoch when they become valid; they are not valid before then assert get_current_epoch(state) >= voluntary_exit.epoch # Verify the validator has been active long enough assert get_current_epoch(state) >= validator.activation_epoch + SHARD_COMMITTEE_PERIOD # Verify signature domain = get_domain(state, DOMAIN_VOLUNTARY_EXIT, voluntary_exit.epoch) signing_root = compute_signing_root(voluntary_exit, domain) assert bls.Verify(validator.pubkey, signing_root, signed_voluntary_exit.signature) # Initiate exit initiate_validator_exit(state, voluntary_exit.validator_index)
the third cryptographic signature; and	The Signed Voluntary Exit message contains the signature from the Validator. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/beacon-chain.md#signedvoluntaryexit class SignedVoluntaryExit(Container): message: VoluntaryExit signature: BLSSignature

EXHIBIT 3

publish the another inchoate data record to at least one of the first client and the second client.	As the Computing Device and the First Client are the same device (Coinbase Ethereum Validator Node), the inchoate data record is already signed by the First Client. As such the inchoate data record is considered to be published to the First Client on creation of the inchoate data record by the Validator Node. The signed inchoate data record (as a complete data record) is broadcast and recorded on the Ethereum Network. The following is an example output message from the Lighthouse Client. Successfully published voluntary exit for validator 0xabcd Voluntary exit has been accepted into the beacon chain, but not yet finalized. Finalization may take several minutes or longer. Before finalization there is a low probability that the exit may be reverted. Current epoch: 29946, Exit epoch: 29951, Withdrawable epoch: 30207 Please keep your validator running till exit epoch Exit epoch in approximately 1920 secs. https://lighthouse-book.sigmaprime.io/voluntary-exit.html

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 1 *A computing device for processing a transaction between a first client device, and a second client device via a transfer mechanism, the transfer mechanism comprising a decentralized digital currency*	**Coinbase Products & Services** **Payment to Validator Node from a transaction (that incurs a transaction fee) on the Solana Network.**
The computing device comprising:	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed or offered) Solana Validator Full Nodes; and • the Coinbase (Owned, managed or offered) Solana supporting Archive Nodes and Light Nodes; and • the Coinbase (Solana compatible) wallets; Where both the Coinbase Nodes and the Coinbase Solana compatible end user wallet device are networked to have direct or indirect communication with each other. The Computing Device and the First Client are considered to be the same device. Four (4) instances have been identified that represent various implementations that exist. • The Computing Device and the First Client are the same device where the First Client runs Coinbase Wallet software to create, sign and submit transactions to the Solana Node for processing. • The Computing Device and the First Client are the same device where the First Client is a Client Browser or Command Line Interface on a Coinbase Solana Node used to create, sign and submit transactions to the Coinbase Solana Node for processing. • The Computing Device and the First Client are the same device where the First Client utilises Coinbase Private Key management mechanism in order to sign and submit transactions to the Coinbase Solana Node for processing. • The Computing Device and the First Client are the same device where the First Client is a Coinbase Validator Node with a vote account key pair to sign voting transactions. The Patent allows for, and the Claims do not prevent, the Computing System from being, or including, the First Client. This is detailed in the Patent description [0055]. *FIG. 1 (see Figure 16) depicts a typical embodiment for practicing the invention—especially for use with a distributed transfer mechanism—where the clients, transfer mechanism, facilitator, and data source are distinct participants. However, the depicted arrangement is not the only one contemplated by the invention. In an alternate embodiment,*

EXHIBIT 3

	the facilitator provides some or all aspects of the transfer mechanism. In another embodiment, the facilitator comprises some or all aspects of a client. For example, part or all of a client's data store, the ability to initiate or accept offers, etc., could be "embedded" in the facilitator, thereby enabling the facilitator to operate as a client itself (e.g., one controlled by the owners of the facilitator, or on behalf of a third party who has entrusted control to the facilitator). In yet another embodiment, the facilitator comprises the data source. Many configurations are contemplated by the invention are possible, and will become apparent to one skilled in the art.
• *a memory for storing a first asymmetric key pair, the first asymmetric key pair comprising a first private key and a first public key;*	The First Client, as part of the Computing Device, need a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer or mobile phone). • Transaction record sector (stores transactions that haven't been submitted to the blockchain yet) kept via the crypto software wallets Where the First Client runs Coinbase Wallet software to sign transactions, contains: • a first key pair sector which is generated and stored in the wallet software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the wallet software. • The wallet software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the First Client is a Coinbase Solana Node Client Browser or Command Line Interface to transaction creation, contains: • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key. Where the First Client utilises Coinbase Private key management in order to sign transactions, contains: • a first key pair sector which is stored in the key management vault/software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the key management vault/software. • The key management vault/software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the First Client is a Coinbase Validator Node with a vote account key pair to sign voting transactions contains • a first key pair sector which is generated and stored on the device

EXHIBIT 3

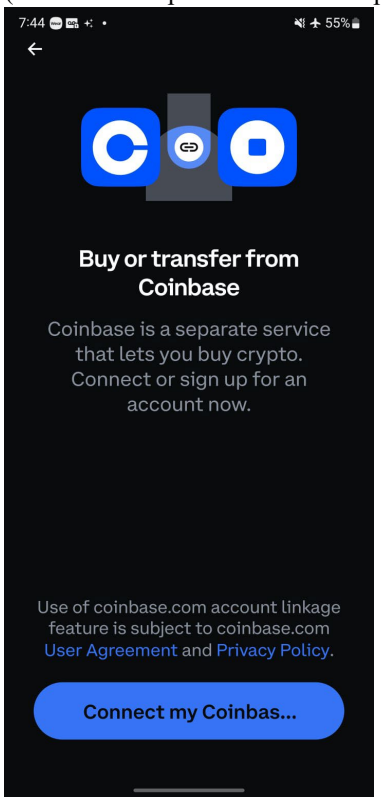
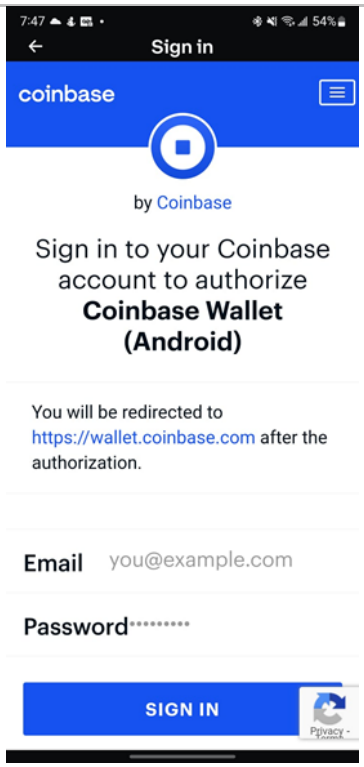
	• The asymmetric key pair stored consists of a first private key and a first public key. Source Code https://github.com/solana-labs/solana/blob/master/validator/src/main.rs#L2999
• *a network interface for receiving terms, the terms comprising:*	The Coinbase Wallet product serves as the network interface usage implementation, we see the following. The app prompts you to connect to "Connect my Coinbase Wallet" Note that the mobile data interface is set to airplane mode, which disables all network data i/o coming into and out of the mobile device (reference the 'plane' icon in the upper RHS corner).  Upon attempting to "Connect my Coinbase…" with the data interface disabled, a blank screen results, as follows:

EXHIBIT 3

	 The Coinbase Wallet app interfaces with nodes and servers in the Coinbase infrastructure to query information or connect to a client. To fetch data like user identification verification credentials, token swap rates, token balances, and other information from the Coinbase infrastructure servers, nodes and smart contracts, such infrastructure responds to queries through the Coinbase Wallet app, the network interface, Once the network interface is enabled, as shown by the wi-fi and cellular service icons in the upper right-hand corner that have replaced the airplane mode symbol, the interface for entering and submitting user credentials is available after querying Coinbase's servers. The interface actually states that "you will be redirected to https://wallet.coinbase.com (their website) after the authorization"

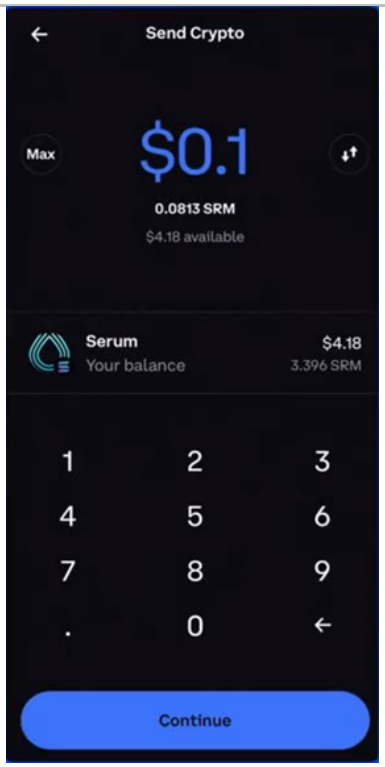
EXHIBIT 3



Once the Coinbase Wallet app is connected, it is able to query Coinbase infrastructure and populate the app with a plethora of information.

Choosing the "Send" option as an illustration, we can opt to send SOL to any address.

EXHIBIT 3



The Coinbase Wallet app populates the app with prices, quantities and transaction fees.

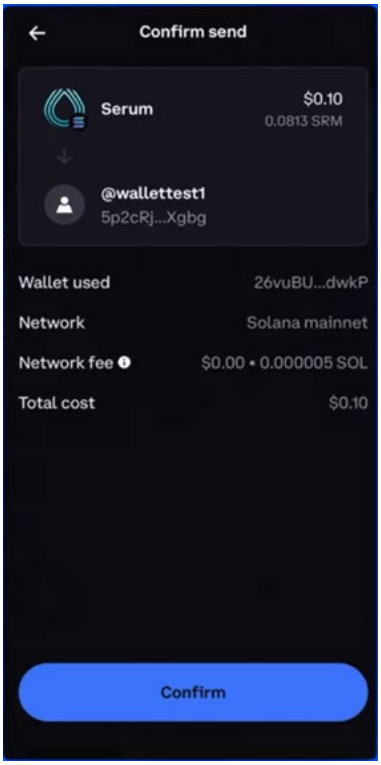


EXHIBIT 3

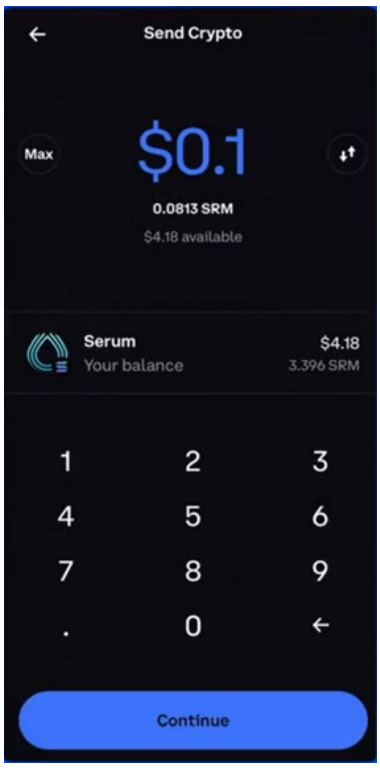
	Note below, with the airplane mode turned on to deprive the Coinbase Wallet app access to the Coinbase infrastructure results in a very different outcome when one attempts to perform the send transaction  An error dialog is given after the “continue” button is depressed.

EXHIBIT 3

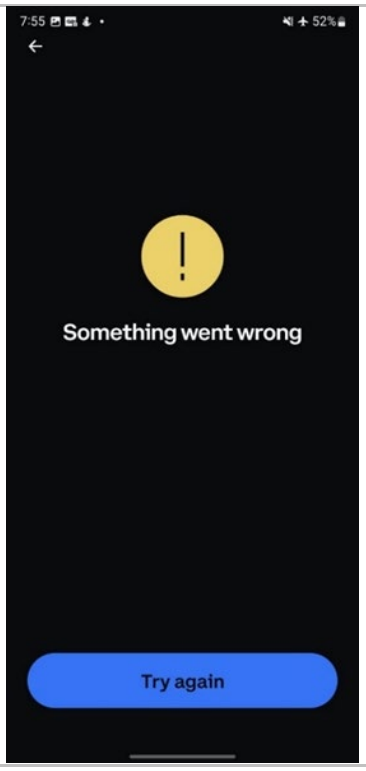
	
o *at least one of a first principal data or a second principal data;*	**First principle data** A transaction fee provided by the Client to pay for sending a transaction. Note: Before any transaction instructions are processed, the fee payer account balance will be deducted to pay for transaction fees. If the fee payer balance is not sufficient to cover transaction fees, the transaction will be dropped by the cluster. If the balance was sufficient, the fees will be deducted whether the transaction is processed successfully or not. In fact, if any of the transaction instructions return an error or violate runtime restrictions, all account changes *except* the transaction fee deduction will be rolled back. https://solanacookbook.com/core-concepts/transactions.html#fees https://jstarry.notion.site/Transaction-Fees-f09387e6a8d84287aa16a34ecb58e239
o *a reference to at least one of a first data source or a second data source; and*	**First data source** The current lampoons per signature Each validator uses signatures per slot (SPS) to estimate network congestion and SPS target to estimate the desired processing capacity of the cluster. The validator learns the SPS target from the genesis config, whereas it calculates SPS from recently processed transactions. The genesis config also defines a target lamports_per_signature, which is the fee to charge per signature when the cluster is operating at SPS target. https://docs.solana.com/implemented-proposals/transaction-fees#congestion-driven-fees

EXHIBIT 3

	Note that the Client uses the JSON RPC API to query the cluster for the current fee parameters. `getFeeForMessage` https://docs.solana.com/developing/clients/jsonrpc-api#getfeeformessage **Second data source** The portion of the transaction fee payable to the Leader. 50% of each transaction fee is burned, with the remaining fee retained by the validator that processes the transaction. https://docs.solana.com/inflation/terminology#effective-inflation-rate-
o *an expiration timestamp;*	**1. The transaction needs to be performed within a timeframe. Thus the expiration time is implied from the use of the network.** *A transaction includes a recent blockhash to prevent duplication and to give transactions lifetimes. Any transaction that is completely identical to a previous one is rejected, so adding a newer blockhash allows multiple transactions to repeat the exact same action. Transactions also have lifetimes that are defined by the blockhash, as any transaction whose blockhash is too old will be rejected.* https://docs.solana.com/developing/programming-model/transactions#recent-blockhash *A blockhash contains a 32-byte SHA-256 hash. It is used to indicate when a client last observed the ledger. Validators will reject transactions when the blockhash is too old.* https://docs.solana.com/developing/programming-model/transactions#blockhash-format Source Code Calculation of blockhash as part of transaction creation. https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L656 **2. The Leader Validator processes the disbursement amount on receipt of the voting transaction.** *Before any transaction instructions are processed, the fee payer account balance will be deducted to pay for transaction fees. If the fee payer balance is not sufficient to cover transaction fees, the transaction will be dropped by the cluster. If the balance was sufficient, the fees will be deducted whether the transaction is processed successfully or not. In fact, if any of the transaction instructions return an error or violate runtime restrictions, all account changes *except* the transaction fee deduction will be rolled back.* https://solanacookbook.com/core-concepts/transactions.html#fees https://jstarry.notion.site/Transaction-Fees-f09387e6a8d84287aa16a34ecb58e239

EXHIBIT 3

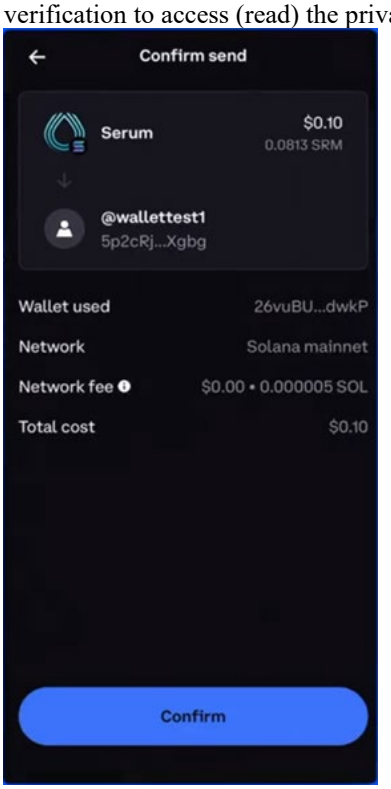
	Note the following. The description of the patent allows for the terms to define a point in time 'on or after the expiration timestamp or at a time or upon an event as defined by the terms…'. In this instance the event is the reception of the transaction for processing. Patent references: *[0123] 22. On or after the expiration timestamp or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …* *[0186] 21. On or after the expiration timestamp, or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …*
• *a computer processor coupled to the memory and the network interface, the computer processor configured to:*	The hardware device running the App (First Client as part of the Computing Device) such as a computer or mobile phone.
i. read the first private key from the memory;	The hardware device running the Coinbase Wallet software on a computer or mobile phone. Coinbase Wallet app requesting verification to access (read) the private key to authorize transaction.  RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L654

EXHIBIT 3

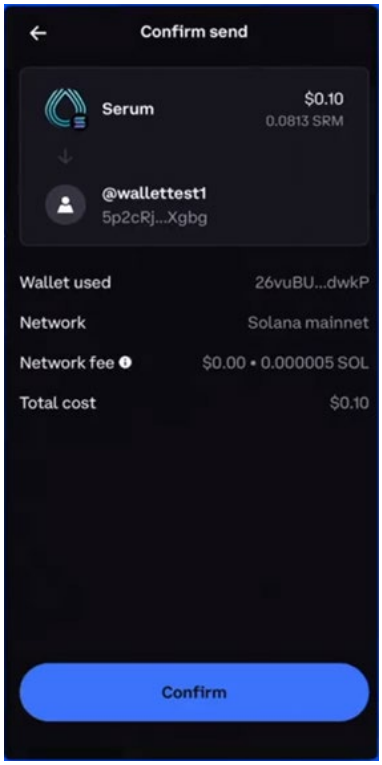
	The validator identity is a system account that is used to pay for all the vote transaction fees submitted to the vote account. Because the validator is expected to vote on most valid blocks it receives, the validator identity account is frequently (potentially multiple times per second) signing transactions and paying fees. For this reason the validator identity keypair must be stored as a "hot wallet" in a keypair file on the same system the validator process is running. Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L17
ii. compute a first cryptographic signature from the first private key;	The hardware device running the Coinbase Wallet software on a computer or mobile phone. Coinbase Wallet app requesting verification to access (read) the private key calculate a signature to authorize transaction.  RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L129 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L813 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L654

EXHIBIT 3

	The validator identity is a system account that is used to pay for all the vote transaction fees submitted to the vote account. Because the validator is expected to vote on most valid blocks it receives, the validator identity account is frequently (potentially multiple times per second) signing transactions and paying fees. For this reason the validator identity keypair must be stored as a "hot wallet" in a keypair file on the same system the validator process is running. Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39
iii. create an inchoate data record comprising:	The First Client as part of the Computing Device creates a transaction message to be sent to the Solana Network. RPC JSON API calls are defined to allow interaction with the Solana Network. The App constructs the following RPC message. **sendTransaction** Object - The transaction object 1. header - The message header contains three unsigned 8-bit values. The first value is the number of required signatures in the containing transaction. The second value is the number of those corresponding account addresses that are read-only. The third value in the message header is the number of read-only account addresses not requiring signatures. 2. Account addresses - The addresses that require signatures appear at the beginning of the account address array, with addresses requesting write access first and read-only accounts following. The addresses that do not require signatures follow the addresses that do, again with read-write accounts first and read-only accounts following. 3. Blockhash - A blockhash contains a 32-byte SHA-256 hash. It is used to indicate when a client last observed the ledger. Validators will reject transactions when the blockhash is too old 4. Instructions - An instruction contains a program id index, followed by a compact-array of account address indexes, followed by a compact-array of opaque 8-bit data. The program id index is used to identify an on-chain program that can interpret the opaque data. The program id index is an unsigned 8-bit index to an account address in the message's array of account addresses. The account address indexes are each an unsigned 8-bit index into that same array. Official Documentation Reference https://docs.solana.com/developing/programming-model/transactions https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Code Reference https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L128

EXHIBIT 3

	https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L640 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L648 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L11
• *a commit input for receiving a commit data from a commit transaction;*	The commit input is the primary account address which will pay the transaction fee. Transactions are required to have at least one account which has signed the transaction and is writable. Writable signer accounts are serialized first in the list of transaction accounts and the first of these accounts is always used as the "fee payer". Note that the Client can use the JSON RPC API to query the cluster for the current fee parameters. (`getFeeForMessage`) to determine the total fee payable to ensure that the paying account has the required amount to pay. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#getfeeformessage RPC Code Reference The transaction sent to the RPC Client contains the Transaction generated by the App. https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L128 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L640 _____ The commit input is the primary account address which will pay the transaction fee. The validator identity is a system account that is used to pay for all the vote transaction fees submitted to the vote account. Because the validator is expected to vote on most valid blocks it receives, the validator identity account is frequently (potentially multiple times per second) signing transactions and paying fees. For this reason the validator identity keypair must be stored as a "hot wallet" in a keypair file on the same system the validator process is running. https://docs.solana.com/running-validator/vote-accounts#validator-identity Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L24 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L31

EXHIBIT 3

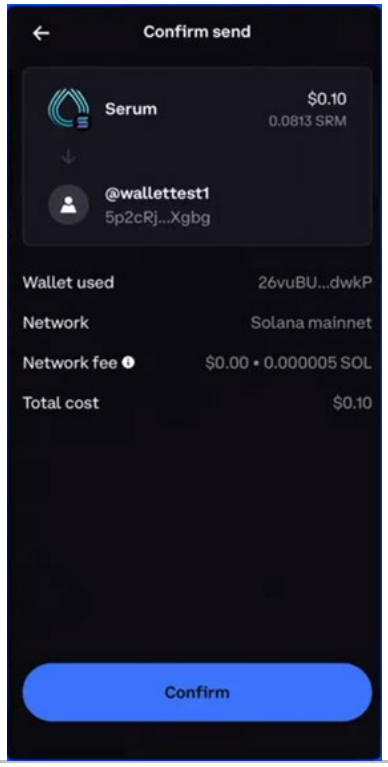
• *one or more output data obtained from at least one of the first principal data or the second principal data, and a value data from at least one of the first data source or the second data source; and*	The output data included in the transaction message is as follows. • The list of signatures in the message. The Validator leader will use this, along with the clusters current lampoons per signature, to calculate the total transaction fee. Official Documentation Reference https://docs.solana.com/implemented-proposals/transaction-fees#congestion-driven-fees RPC Code Reference The transaction sent to the RPC Client contains the Transaction generated by the App. https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L128 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L640 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L36 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L38 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39
• *the first cryptographic signature; and*	The hardware device running the Coinbase Wallet software on a computer or mobile phone. Coinbase Wallet app requesting verification to access (read) the private key to calculate a signature to authorize transaction. 

EXHIBIT 3

	The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. The Coinbase Wallet constructs the following RPC message. **sendTransaction** Submits a signed transaction to the cluster for processing. Before submitting, the following preflight checks are performed: The transaction signatures are verified The transaction is simulated against the bank slot specified by the preflight commitment. On failure an error will be returned. Preflight checks may be disabled if desired. It is recommended to specify the same commitment and preflight commitment to avoid confusing behavior. The returned signature is the first signature in the transaction, which is used to identify the transaction (transaction id). This identifier can be easily extracted from the transaction data before submission. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction https://docs.solana.com/developing/programming-model/transactions#signatures https://solanacookbook.com/core-concepts/transactions.html#fees RPC Code Reference https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L129 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L813 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L654 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L38 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39
iv. *publish the inchoate data record to at least one of the first client device or the second client device,*	Given the Computing Device and the First Client are the same device, this clause is not applicable. The First Client as part of the Computing Device does create a transaction message (where the signed inchoate transaction record = complete transaction record). Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code

EXHIBIT 3

	https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935 Validator Code Reference https://github.com/solana-labs/solana/blob/master/gossip/src/cluster_info.rs#L1087
wherein the decentralized digital currency comprises a distributed ledger that enables processing the transaction between the first client device and the second client device without the need for a trusted central authority,	A SOL is the name of Solana's native token, which can be passed to nodes in a Solana cluster in exchange for running an on-chain program or validating its output. The system may perform micropayments of fractional SOLs, which are called lamports. https://docs.solana.com/introduction#what-are-sols
wherein the inchoate data record is used by at least one of the first client device or the second client device to create a complete data record and to create the transaction by broadcasting the complete data record for transmitting and receiving among network participants in the computer network for recording in the distributed ledger,	The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. Only the First Client is required to sign the transaction and so once signed the RPC JSON API call '**sendTransaction'** is considered to be a complete data record and is broadcast to the Leader Validator via the Coinbase Solana Node. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935 Validator Code Reference https://github.com/solana-labs/solana/blob/master/gossip/src/cluster_info.rs#L1087
wherein at least one of the first client device or the second client device signs the inchoate data record and saves a copy of the inchoate data record on at least one of the first client device or the second client device; and	The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. Only the First Client is required to sign the transaction and so once signed the RPC JSON API call '**sendTransaction'** is considered to be a complete data record and is broadcast to the Leader Validator via the Coinbase Solana Node. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L38

EXHIBIT 3

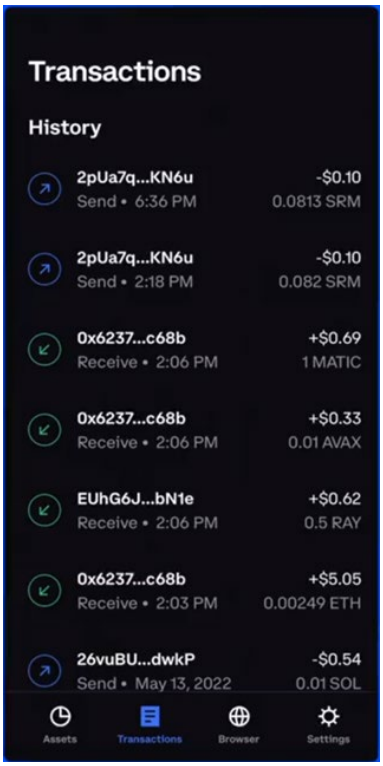
	https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39 _____ A copy of the inchoate data record is (optionally, as described in pgs 11 and 15 of the general description text of the patent) saved on the Leader Validator node.
wherein the at least one of the computing device, the first client device, or the second client device verifies the recording of the complete data record in the distributed ledger by observing an external state.	The Coinbase Wallet shows the transaction history.  The Coinbase Wallet shows the transaction details along with a link to the Solana explorer.

EXHIBIT 3

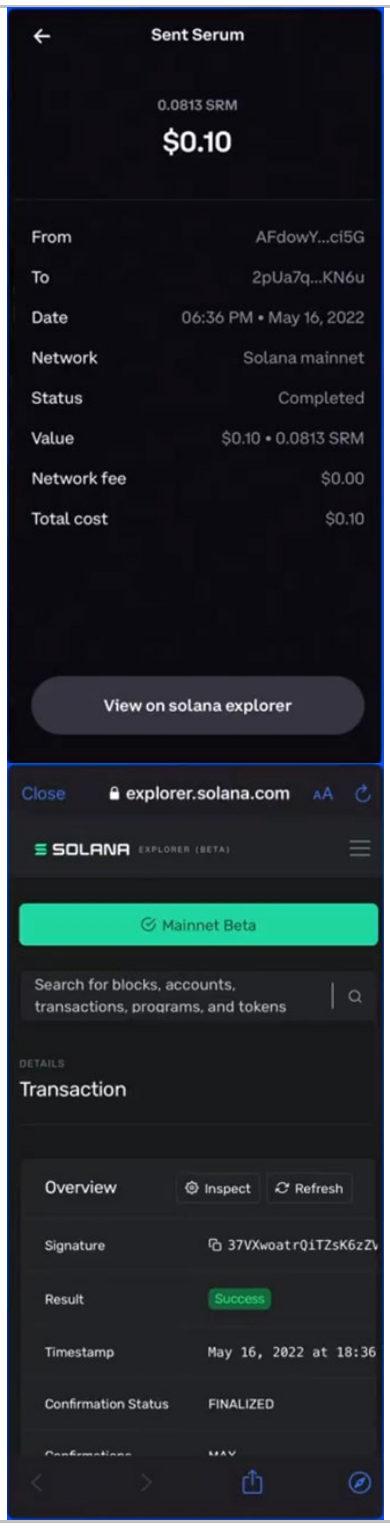

Sent Serum
0.0813 SRM
$0.10
From
AFdowY...ci5G
To
2pUa7q...KN6u
Date
06:36 PM • May 16, 2022
Network
Solana mainnet
Status
Completed
Value
$0.10 • 0.0813 SRM
Network fee
$0.00
Total cost
$0.10
View on solana explorer
Close
explorer.solana.com
SOLANA
Mainnet Beta
Search for blocks, accounts, transactions, programs, and tokens
Transaction
Overview
Inspect
Refresh
Signature
Result
Success
Timestamp
May 16, 2022 at 18:36
Confirmation Status
FINALIZED

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 2	**Coinbase Products & Services** **Payment to Validator Node from a transaction (that incurs a transaction fee) on the Solana Network.**
The device of claim 1, where: *the computer processor is configured to obtain the one or more output data based on:*	The following disbursements are made. 1. The transaction fee is withdrawn from the senders account by the Lead Validator as part of processing the transaction and (likely) inclusion into the Block. 2. The transaction fees collected from all processed transactions are banked for disbursement at the end of the epoch. 3. The transaction fees collected are disbursed to the Lead Validators vote account based on the accounts commission rate with the remainder of the rewards being distributed to all of the stake accounts delegated to that vote account, proportional to the active stake weight of each stake account.
the first principal data; and *the value data from the first data source.*	Before any transaction instructions are processed, the fee payer account balance will be deducted to pay for transaction fees. If the fee payer balance is not sufficient to cover transaction fees, the transaction will be dropped by the cluster. If the balance was sufficient, the fees will be deducted whether the transaction is processed successfully or not. In fact, if any of the transaction instructions return an error or violate runtime restrictions, all account changes *except* the transaction fee deduction will be rolled back. https://solanacookbook.com/core-concepts/transactions.html#fees https://jstarry.notion.site/Transaction-Fees-f09387e6a8d84287aa16a34ecb58e239 The collected transaction fees (of those transactions included in the Block) by the Lead Validator are: 1. burnt based on the genesis burnt rate (second data source) 2. written to the Validators vote account for disbursement. Source Code – Calculation of fee for the individual transaction using the first Data Source https://github.com/solana-labs/solana/blob/master/runtime/src/bank.rs#L4805 https://github.com/solana-labs/solana/blob/master/runtime/src/bank.rs#L4718 Source Code – Withdraw transaction fee from senders account https://github.com/solana-labs/solana/blob/master/runtime/src/bank.rs#L4822 https://github.com/solana-labs/solana/blob/master/runtime/src/bank.rs#L6256

EXHIBIT 3

	Source Code – Burning of a percentage of all fees collected using the second Data Source https://github.com/solana-labs/solana/blob/master/runtime/src/bank.rs#L3280 https://github.com/solana-labs/solana/blob/master/runtime/src/bank.rs#L3284

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 7 A system for processing a transaction between a first client device and a second client device via a transfer mechanism the system comprising a computing device, the first client device, the second client device, and the transfer mechanism.	**Coinbase Products & Services** **Payment to Validator Node from a transaction (that incurs a transaction fee) on the Solana Network.**
7. a. *the computing device comprising:* *i. a first memory comprising for storing a first asymmetric key pair, the first asymmetric key pair comprising a first private key and a first public key;*	The First Client, as part of the Facilitator, need a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer or mobile phone). • Transaction record sector (stores transactions that haven't been submitted to the blockchain yet) kept via the crypto software wallets Where the First Client runs Coinbase Wallet software to sign transactions, contains: • a first key pair sector which is generated and stored in the wallet software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the wallet software. • The wallet software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the First Client is a Coinbase Solana Node Client Browser or Command Line Interface to transaction creation, contains: • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key. Where the First Client utilises Coinbase Private key management in order to sign transactions, contains: • a first key pair sector which is stored in the key management vault/software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the key management vault/software. • The key management vault/software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the First Client is a Coinbase Validator Node with a vote account key pair to sign voting transactions contains • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key Source Code https://github.com/solana-labs/solana/blob/master/validator/src/main.rs#L2999

EXHIBIT 3

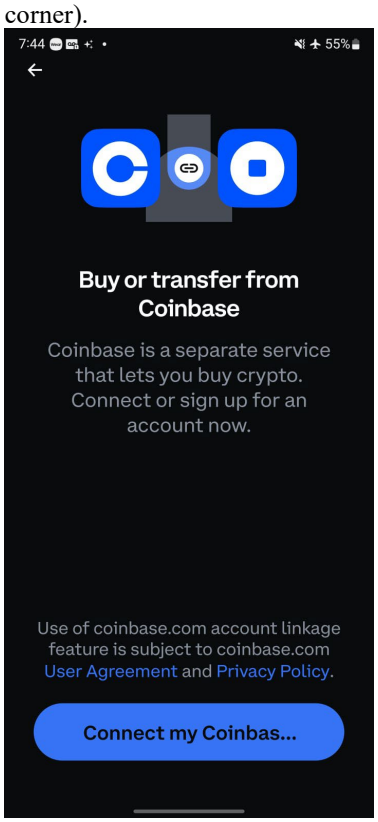
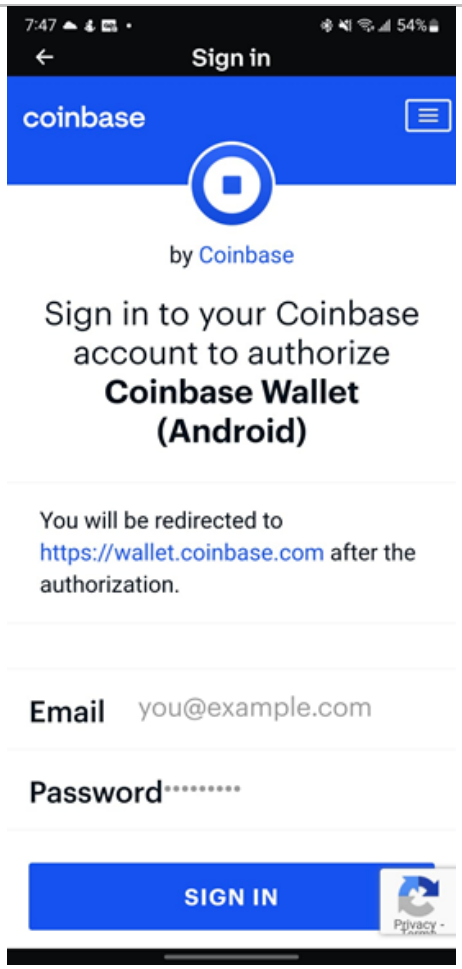
ii.	*a first network interface for receiving terms, the terms comprising:*	The Coinbase Wallet product serves as the network interface usage implementation, we see the following. The app prompts you to connect to "Connect my Coinbase Wallet" Note that the mobile data interface is set to airplane mode, which disables all network data i/o coming into and out of the mobile device (reference the 'plane' icon in the upper RHS corner).  Upon attempting to "Connect my Coinbase…" with the data interface disabled, a blank screen results, as follows:

EXHIBIT 3

	 The Coinbase Wallet app interfaces with nodes and servers in the Coinbase infrastructure to query information or connect to a client. To fetch data like user identification verification credentials, token swap rates, token balances, and other information from the Coinbase infrastructure servers, nodes and smart contracts, such infrastructure responds to queries through the Coinbase Wallet app, the network interface, Once the network interface is enabled, as shown by the wi-fi and cellular service icons in the upper right-hand corner that have replaced the airplane mode symbol, the interface for entering and submitting user credentials is available after querying Coinbase's servers. The interface actually states that "you will be redirected to https://wallet.coinbase.com (their website) after the authorization"

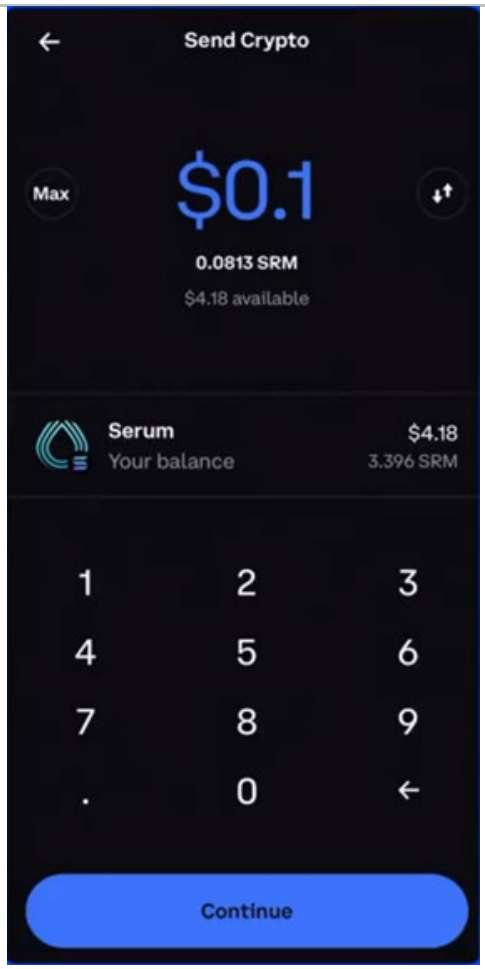
EXHIBIT 3



Once the Coinbase Wallet app is connected, it is able to query Coinbase infrastructure and populate the app with a plethora of information.

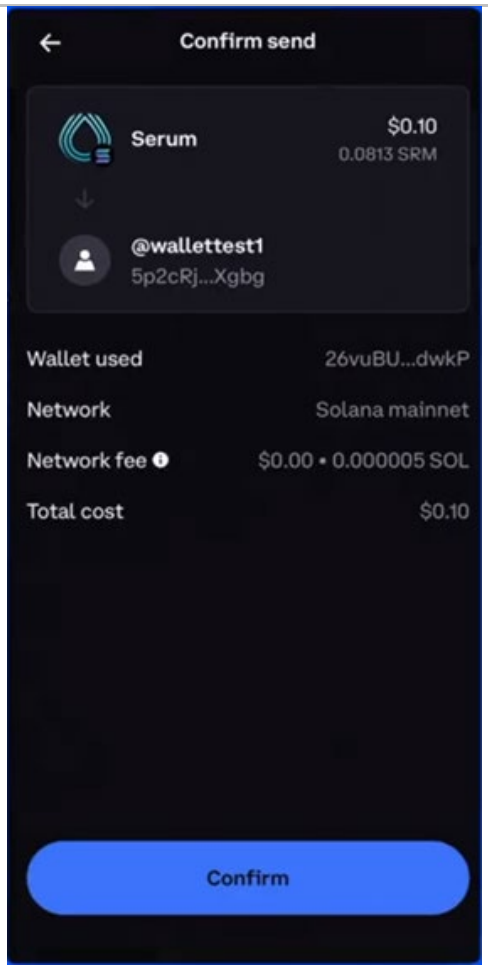
Choosing the “Send” option as an illustration, we can opt to send SOL to any address.

EXHIBIT 3



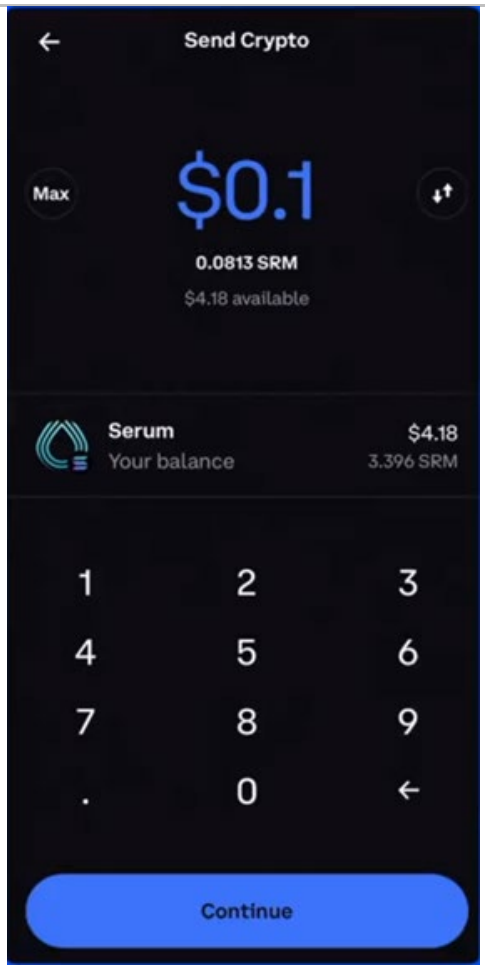
The Coinbase Wallet app populates the app with prices, quantities and transaction fees.

EXHIBIT 3



Note below, with the airplane mode turned on to deprive the Coinbase Wallet app access to the Coinbase infrastructure results in a very different outcome when one attempts to perform the send transaction

EXHIBIT 3



An error dialog is given after the “continue” button is depressed.

EXHIBIT 3

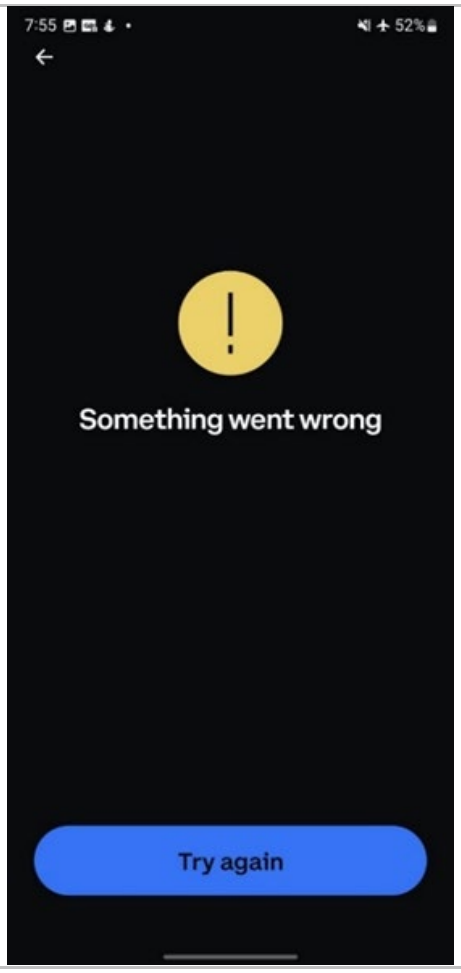
	
A. at least one of a first principal data or a second principal data;	**First principle data** A transaction fee provided by the Client to pay for sending a transaction. Note: Before any transaction instructions are processed, the fee payer account balance will be deducted to pay for transaction fees. If the fee payer balance is not sufficient to cover transaction fees, the transaction will be dropped by the cluster. If the balance was sufficient, the fees will be deducted whether the transaction is processed successfully or not. In fact, if any of the transaction instructions return an error or violate runtime restrictions, all account changes *except* the transaction fee deduction will be rolled back. https://solanacookbook.com/core-concepts/transactions.html#fees https://jstarry.notion.site/Transaction-Fees-f09387e6a8d84287aa16a34ecb58e239
B. a reference to at least one of a first data source or a second data source; and	**First data source** The current lampoons per signature Each validator uses signatures per slot (SPS) to estimate network congestion and SPS target to estimate the desired processing capacity of the cluster. The validator learns the SPS target from the genesis config, whereas it calculates SPS from recently processed transactions. The genesis config also defines a target lamports_per_signature, which is the fee to charge per signature when the cluster is operating at SPS target. https://docs.solana.com/implemented-proposals/transaction-fees#congestion-driven-fees

EXHIBIT 3

	Note that the Client uses the JSON RPC API to query the cluster for the current fee parameters. `getFeeForMessage` https://docs.solana.com/developing/clients/jsonrpc-api#getfeeformessage **Second data source** The portion of the transaction fee payable to the Leader. 50% of each transaction fee is burned, with the remaining fee retained by the validator that processes the transaction. https://docs.solana.com/inflation/terminology#effective-inflation-rate-
C. *an expiration timestamp,*	**1. The transaction needs to be performed within a timeframe.** *A transaction includes a recent blockhash to prevent duplication and to give transactions lifetimes. Any transaction that is completely identical to a previous one is rejected, so adding a newer blockhash allows multiple transactions to repeat the exact same action. Transactions also have lifetimes that are defined by the blockhash, as any transaction whose blockhash is too old will be rejected.* https://docs.solana.com/developing/programming-model/transactions#recent-blockhash *A blockhash contains a 32-byte SHA-256 hash. It is used to indicate when a client last observed the ledger. Validators will reject transactions when the blockhash is too old.* https://docs.solana.com/developing/programming-model/transactions#blockhash-format Source Code Calculation of blockhash as part of transaction creation. https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L656 **2. The Leader Validator processes the disbursement amount on receipt of the voting transaction.** *Before any transaction instructions are processed, the fee payer account balance will be deducted to pay for transaction fees. If the fee payer balance is not sufficient to cover transaction fees, the transaction will be dropped by the cluster. If the balance was sufficient, the fees will be deducted whether the transaction is processed successfully or not. In fact, if any of the transaction instructions return an error or violate runtime restrictions, all account changes *except* the transaction fee deduction will be rolled back.* https://solanacookbook.com/core-concepts/transactions.html#fees https://jstarry.notion.site/Transaction-Fees-f09387e6a8d84287aa16a34ecb58e239 Note the following. The description of the patent allows for the terms to define a point in time ‘on or after the expiration timestamp or at a time or upon an event as defined by the terms…’. In this instance the event is the reception of the transaction for processing. Patent references:

EXHIBIT 3

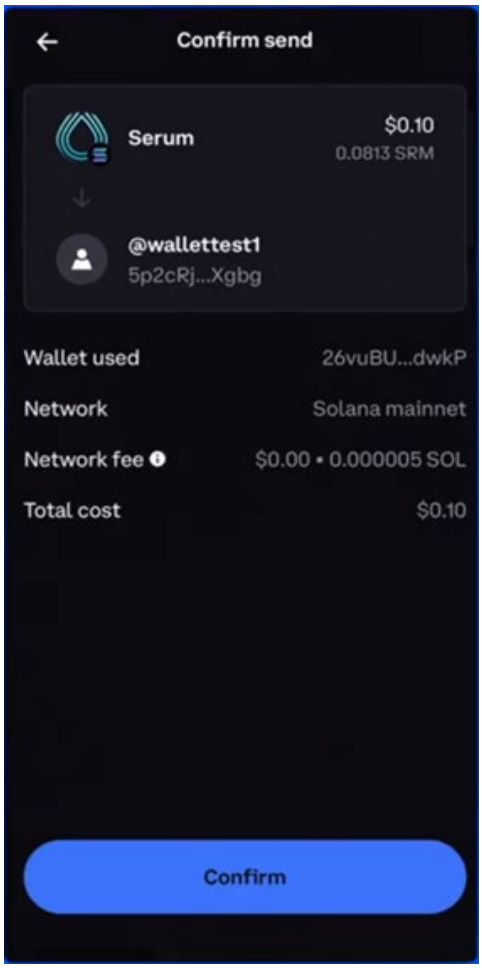
		[0123] 22. On or after the expiration timestamp or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, … *[0186] 21. On or after the expiration timestamp, or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …*
iii.	*a first computer processor coupled to the first memory and the first network interface, the first computer processor configured to:*	The hardware device running the App (First Client as part of the Facilitator) such as a computer or mobile phone.
A.	*read the first private key from the first memory*	The hardware device running the Coinbase Wallet software on a computer or mobile phone. Coinbase Wallet app requesting verification to access (read) the private key to authorize transaction.  RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L654 Validator Code Reference

EXHIBIT 3

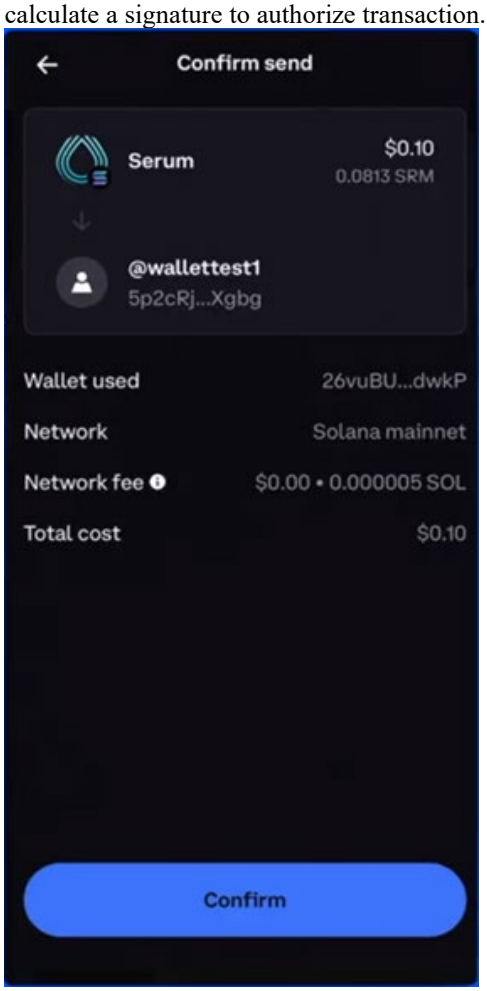
	https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L17
B. *compute a first cryptographic signature from the first private key;*	The hardware device running the Coinbase Wallet software on a computer or mobile phone. Coinbase Wallet app requesting verification to access (read) the private key to calculate a signature to authorize transaction.  RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L129 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L813 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L654 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39
C. *create an inchoate data record comprising:*	The App (First Client as part of the Facilitator) creates a transaction message to be sent to the Solana Network. RPC JSON API calls are defined to allow interaction with the Solana Network. The App constructs the following RPC message. **sendTransaction** Object - The transaction object

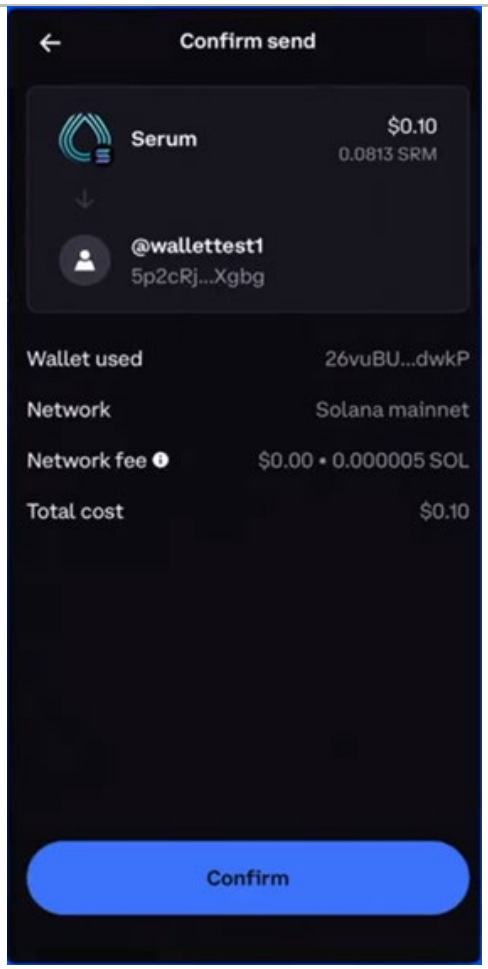
EXHIBIT 3

	1. header - The message header contains three unsigned 8-bit values. The first value is the number of required signatures in the containing transaction. The second value is the number of those corresponding account addresses that are read-only. The third value in the message header is the number of read-only account addresses not requiring signatures. 2. Account addresses - The addresses that require signatures appear at the beginning of the account address array, with addresses requesting write access first and read-only accounts following. The addresses that do not require signatures follow the addresses that do, again with read-write accounts first and read-only accounts following. 3. Blockhash - A blockhash contains a 32-byte SHA-256 hash. It is used to indicate when a client last observed the ledger. Validators will reject transactions when the blockhash is too old 4. Instructions - An instruction contains a program id index, followed by a compact-array of account address indexes, followed by a compact-array of opaque 8-bit data. The program id index is used to identify an on-chain program that can interpret the opaque data. The program id index is an unsigned 8-bit index to an account address in the message's array of account addresses. The account address indexes are each an unsigned 8-bit index into that same array. Official Documentation Reference https://docs.solana.com/developing/programming-model/transactions https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Code Reference https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L128 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L640 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L648 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L11
I. a commit input for receiving a commit data from a commit transaction;	The commit input is the primary account address which will pay the transaction fee. *Transactions are required to have at least one account which has signed the transaction and is writable. Writable signer accounts are serialized first in the list of transaction accounts and the first of these accounts is always used as the "fee payer".* *Note that the Client can use the JSON RPC API to query the cluster for the current fee parameters. (`getFeeForMessage`) to determine the total fee payable to ensure that the paying account has the required amount to pay.* https://docs.solana.com/developing/clients/jsonrpc-api#getfeeformessage RPC Code Reference The transaction sent to the RPC Client contains the Transaction generated by the App. https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L128 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L640

EXHIBIT 3

		The commit input is the primary account address which will pay the transaction fee. The validator identity is a system account that is used to pay for all the vote transaction fees submitted to the vote account. Because the validator is expected to vote on most valid blocks it receives, the validator identity account is frequently (potentially multiple times per second) signing transactions and paying fees. For this reason the validator identity keypair must be stored as a "hot wallet" in a keypair file on the same system the validator process is running. https://docs.solana.com/running-validator/vote-accounts#validator-identity Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L24 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L31
II.	*one or more outputs obtained from at least one of the first principal data or the second principal data, and a value data from at least one of the first data source or the second data source; and;*	The output data included in the transaction message is as follows. • The list of signatures in the message. The Validator leader will use this, along with the clusters current lampoons per signature, to calculate the total transaction fee. Official Documentation Reference https://docs.solana.com/implemented-proposals/transaction-fees#congestion-driven-fees RPC Code Reference The transaction sent to the RPC Client contains the Transaction generated by the App. https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L128 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L640 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L36 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L38 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39
III.	*The first cryptographic signature; and*	The hardware device running the Coinbase Wallet software on a computer or mobile phone. Coinbase Wallet app requesting verification to access (read) the private key to calculate a signature to authorize transaction.

EXHIBIT 3



The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. The Coinbase Wallet constructs the following RPC message.

sendTransaction

Submits a signed transaction to the cluster for processing.

Before submitting, the following preflight checks are performed:

The transaction signatures are verified
The transaction is simulated against the bank slot specified by the preflight commitment. On failure an error will be returned. Preflight checks may be disabled if desired. It is recommended to specify the same commitment and preflight commitment to avoid confusing behavior.
The returned signature is the first signature in the transaction, which is used to identify the transaction (transaction id). This identifier can be easily extracted from the transaction data before submission.

Official Documentation Reference
https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction
https://docs.solana.com/developing/programming-model/transactions#signatures
https://solanacookbook.com/core-concepts/transactions.html#fees

EXHIBIT 3

	RPC Code Reference https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L129 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L813 https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L654 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L38 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39
D. publish the inchoate data record to at least one of the first client device or the second client device;;	Given the Facilitator and the First Client are the same device, this clause is not necessarily applicable. The App (First Client as part of the Facilitator) does create a transaction message (where the signed inchoate transaction record = complete transaction record) to the Solana Network and the Second Client. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935
7. b. the first client comprises: *i. a second memory for storing a second asymmetric key pair, the second asymmetric key pair comprising a second private key and a second public key.*	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.i.
ii. a second network interface; and	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.ii.
iii. a second computer processor coupled to the second memory and the second network interface, the second computer processor configured to:	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.A-E.

EXHIBIT 3

A. read the second private key from the second memory;	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.A.
B. read the inchoate data record	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable.
C. compute a second cryptographic signature from the second private key;	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.B.
D. create a complete data record comprising: *I. the commit input;*	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
II. the output data;	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
III. the first cryptographic signature, and	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
IV. the second cryptographic signature,	The Facilitator and the First Client are both considered to be the same device; namely the App or mechanism that is using the Solana Network by sending a transaction. Thus, the clause is not assessable as it is covered by the above clauses 7a.iii.C.
E. create a transaction by submitting the complete data record to the transfer mechanism;	The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. Only the First Client is required to sign the transaction and so once signed the RPC JSON API call **'sendTransaction'** is considered to be a complete data record and is broadcast to the Leader Validator via the Coinbase Solana Node. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935 _____ The Leader Validator, as the second client, uses the transaction (completed disbursement transaction record) and other transactions in the TX Pool to create a Block Transaction. The Block is broadcast to all network participants for consensus and recording to the distributed ledger.

EXHIBIT 3

	Clients send transactions to any validator's Transaction Processing Unit (TPU) port. If the node is in the validator role, it forwards the transaction to the designated leader. If in the leader role, the node bundles incoming transactions, timestamps them creating an entry, and pushes them onto the cluster's data plane. Once on the data plane, the transactions are validated by validator nodes, effectively appending them to the ledger. https://docs.solana.com/cluster/overview#sending-transactions-to-a-cluster
7. c. the second client comprises: *i. a third memory for storing a third asymmetric key pair, the third asymmetric key pair comprising a third private key and a third public key;*	The Validator Note (including the Leader Validator) requires an account (with key pair) in order to sign the produced blocks. *7 System Architecture* *7.1 Components* *7.1.1 Leader, Proof of History generator* *The Leader is an elected Proof of History generator. It consumes arbitrary user transactions and outputs a Proof of History sequence of all the transactions that guarantee a unique global order in the system. After each batch of transactions the Leader outputs a signature of the state that is the result of running the transactions in that order. This signature is signed with the identity of the Leader.* https://cryptorating.eu/whitepapers/Solana/solana-whitepaper-en.pdf *Vote Authority#* *The vote authority keypair is used to sign each vote transaction the validator node wants to submit to the cluster. This doesn't necessarily have to be unique from the validator identity, as you will see later in this document. Because the vote authority, like the validator identity, is signing transactions frequently, this also must be a hot keypair on the same file system as the validator process.* https://docs.solana.com/running-validator/vote-accounts#vote-authority Source Code https://github.com/solana-labs/solana/blob/master/validator/src/main.rs#L2999
ii. a third network interface; and	The Validator Nodes require network interface in order for them to participate in Block Production and network consensus. *Networking#* *Internet service should be at least 300Mbit/s symmetric, commercial. 1GBit/s preferred* *Port Forwarding#* *The following ports need to be open to the internet for both inbound and outbound* *It is not recommended to run a validator behind a NAT. Operators who choose to do so should be comfortable configuring their networking equipment and debugging any traversal issues on their own.* *Required#* *8000-10000 TCP/UDP - P2P protocols (gossip, turbine, repair, etc). This can be limited to any free 12 port range with --dynamic-port-range* https://docs.solana.com/running-validator/validator-reqs#networking

EXHIBIT 3

iii. a third computer processor coupled to the third memory and the third network interface, the third computer processor configured to read the third private key from the third memory; and	The Solana Network Node running the Validator software. https://docs.solana.com/running-validator/validator-reqs
wherein the at least one of the first client device or the second client device signs the inchoate data record and saves a copy of the inchoate data record on at least one of the first client device or the second client device.	The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. Only the First Client is required to sign the transaction and so once signed the RPC JSON API call '**sendTransaction'** is considered to be a complete data record and is broadcast to the Leader Validator via the Coinbase Solana Node. Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935 Validator Code Reference https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L38 https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_transaction.rs#L39 _____ A copy of the inchoate data record is (optionally, as described in pgs 11 and 15 of the general description text of the patent) saved on the Leader Validator node.
wherein the transfer mechanism comprising a decentralized digital currency that comprises a distributed ledger that enables processing the transaction between the first client device and the second client device without the need of a trusted central authority,	A SOL is the name of Solana's native token, which can be passed to nodes in a Solana cluster in exchange for running an on-chain program or validating its output. The system may perform micropayments of fractional SOLs, which are called lamports. https://docs.solana.com/introduction#what-are-sols
wherein the transaction is created by broadcasting the complete data record for transmitting and receiving among network participants in the computer network for recording in the distributed ledger, and	The Coinbase Wallet creates a transaction message to be sent to the Coinbase Solana Node. RPC JSON API calls are defined to allow interaction with the Solana Nodes. Only the First Client is required to sign the transaction and so once signed the RPC JSON API call '**sendTransaction'** is considered to be a complete data record and is broadcast to the Leader Validator via the Coinbase Solana Node.

EXHIBIT 3

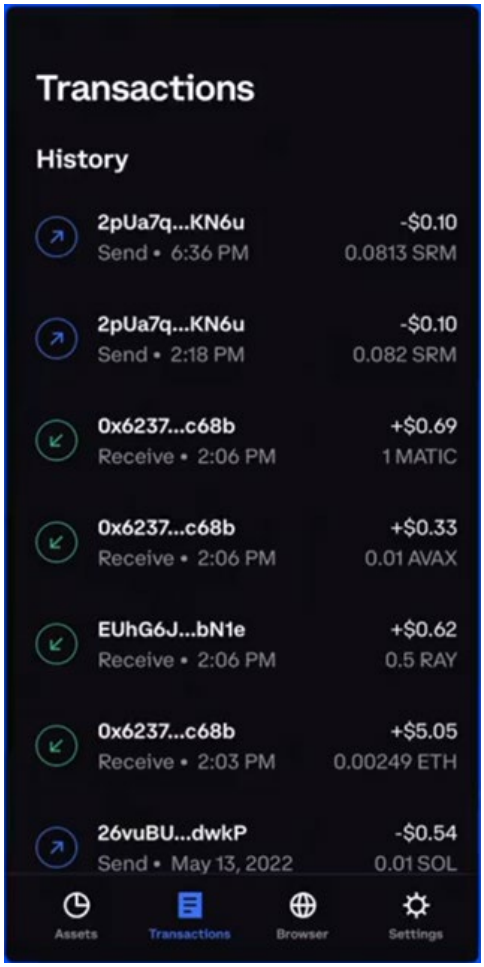
	Official Documentation Reference https://docs.solana.com/developing/clients/jsonrpc-api#sendtransaction RPC Source Code https://github.com/solana-labs/solana/blob/master/client/src/nonblocking/rpc_client.rs#L935 Validator Code Reference https://github.com/solana-labs/solana/blob/master/gossip/src/cluster_info.rs#L1087
wherein at least one of the computer device, the first client device, or the second client device verifies the recording of the complete data record in the distributed ledger by observing an external state	The Coinbase Wallet shows the transaction history.  The Coinbase Wallet shows the transaction details along with a link to the Solana explorer.

EXHIBIT 3

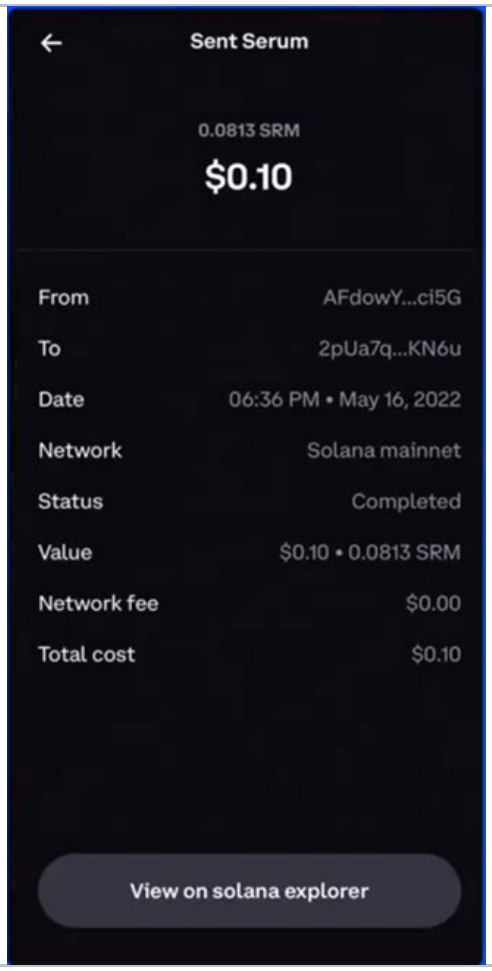

Sent Serum
0.0813 SRM
$0.10
From
AFdowY...ci5G
To
2pUa7q...KN6u
Date
06:36 PM • May 16, 2022
Network
Solana mainnet
Status
Completed
Value
$0.10 • 0.0813 SRM
Network fee
$0.00
Total cost
$0.10
View on solana explorer

EXHIBIT 3

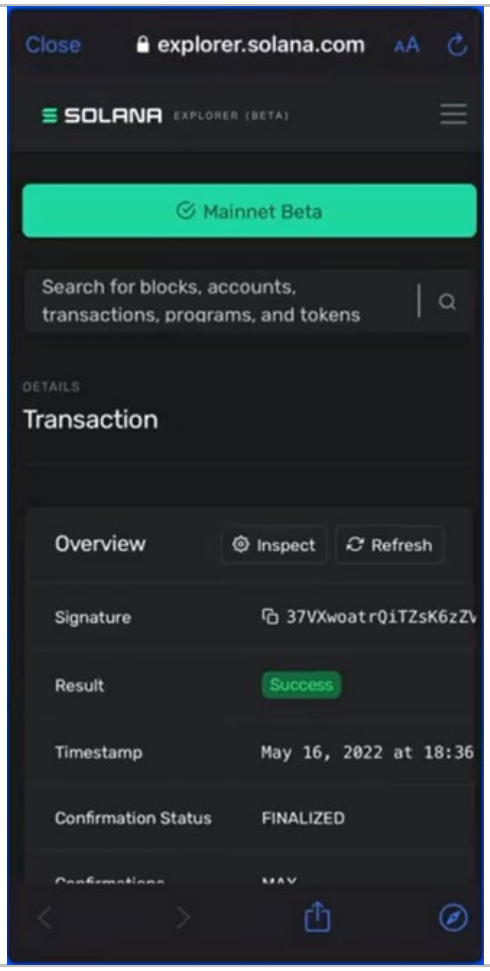

Close
explorer.solana.com
AA
SOLANA EXPLORER (BETA)
Mainnet Beta
Search for blocks, accounts, transactions, programs, and tokens
DETAILS
Transaction
Overview
Inspect
Refresh
Signature
37VXwoatrQiTZsK6zZ
Result
Success
Timestamp
May 16, 2022 at 18:36
Confirmation Status
FINALIZED

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 1 *A computing device for processing a transaction between a first client device, and a second client device via a transfer mechanism, the transfer mechanism comprising a decentralized digital currency*	**Coinbase Products & Services** **The transfer of a NFT (on the Ethereum network) from one party to another.**
The computing device comprising:	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed or offered) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed or offered) Ethereum supporting Archive Nodes and Light Nodes; and • the Coinbase (Ethereum compatible) wallets; Where both the Coinbase Nodes and the Coinbase Ethereum compatible end user wallet device are networked to have direct or indirect communication with each other. **Client Device** The Second Client is the end user device that accepts an offer for an NFT. The First Client is the end user device that authorises an NFT to be offered for sale. Three (3) Client instances have been identified that represent various implementations that exist. 1. Client is a device running an application that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js. This device will also use a key storage wallet with EIP-191 or EIP-712 signing support (such as Metamask/Coinbase Wallet) to send an NFT transactions. The use of an EIP-1193 based Ethereum Provider API or EIP-191/EIP-712 signing support means the Client is considered to be part of the Ethereum Network. http://eips/ethereum.org/EIPS/eip-191 http://eips/ethereum.org/EIPS/eip-712 http://eips/ethereum.org/EIPS/eip-1193 2. Client representative of Eth Client Browser or Command Line Interface to create or send an NFT. The Client is considered to be part of the Ethereum Network. 3. Client is a device running an application that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js. This device will also use a key storage wallet. This device uses private key management in order to sign transactions The Patent allows for, and the Claims do not prevent, the Computing System from being, or including, the Client. This is detailed in the Patent description [0055]. *FIG. 1 (see Figure 16) depicts a typical embodiment for practicing the invention—especially for use with a distributed transfer mechanism—where the clients, transfer mechanism, facilitator, and data source are distinct participants. However, the depicted arrangement is not the only one*

EXHIBIT 3

	contemplated by the invention. In an alternate embodiment, the facilitator provides some or all aspects of the transfer mechanism. In another embodiment, the facilitator comprises some or all aspects of a client. For example, part or all of a client's data store, the ability to initiate or accept offers, etc., could be "embedded" in the facilitator, thereby enabling the facilitator to operate as a client itself (e.g., one controlled by the owners of the facilitator, or on behalf of a third party who has entrusted control to the facilitator). In yet another embodiment, the facilitator comprises the data source. Many configurations are contemplated by the invention are possible, and will become apparent to one skilled in the art. In addition, paragraph 0055 reads as follows: *It will become apparent to one skilled in the art that aspects of each of embodiments above may be commingled. For example, the first client could transmit the offer to the facilitator, where the second client could find and retrieve it. As mentioned above, aspects of one or both of the first client and the second client could coincide with the facilitator allowing many of the above steps to be omitted as redundant where the facilitator is entrusted to act as a proxy for or on behalf of one of the first party and the second party. The facilitator could contain aspects of one of the clients, but not the other, in which case the extra-facilitator client would optionally independently validate transaction records it received from the facilitator before signing them, etc. In such embodiments, the facilitator typically comprises a means to control aspects of a client it comprises via an interface such as a web-based user interface (UI), an application programmer's interface (API), etc.*
• *a memory for storing a first asymmetric key pair, the first asymmetric key pair comprising a first private key and a first public key;*	The Client device hosting the API software/libraries (used by an App), as part of the Computing Device, need a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer or mobile phone). • Transaction record sector (stores transactions that haven't been submitted to the blockchain yet) kept via the crypto software wallets Where the Client running the App and Wallet software, contains: • a first key pair sector which is generated and stored in the wallet software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the wallet software. • The wallet software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the Client interacts with the Eth Client Browser or Command Line Interface to transaction creation, contains: • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key. Where the Client interacts with private Apps with Private key management in order to sign transactions, contains: • a first key pair sector which is stored in the key management vault/software

EXHIBIT 3

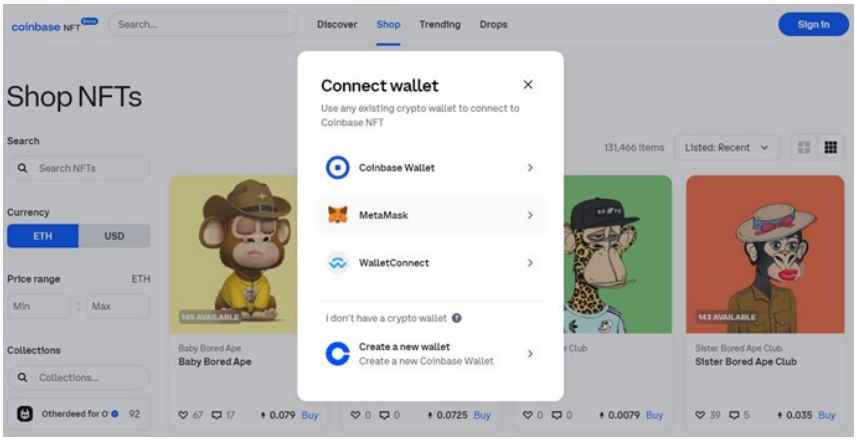
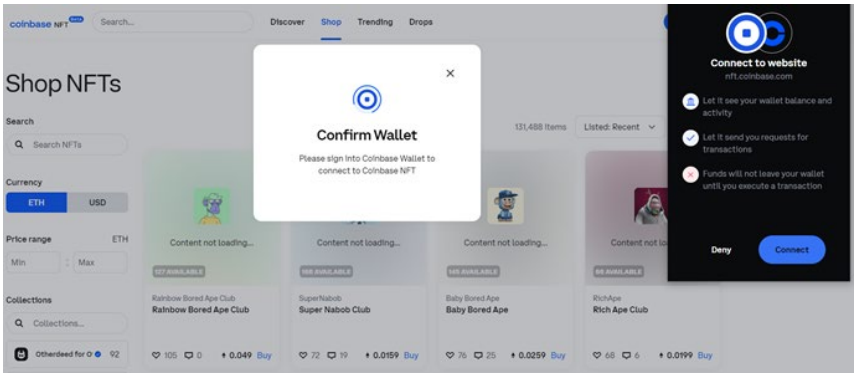
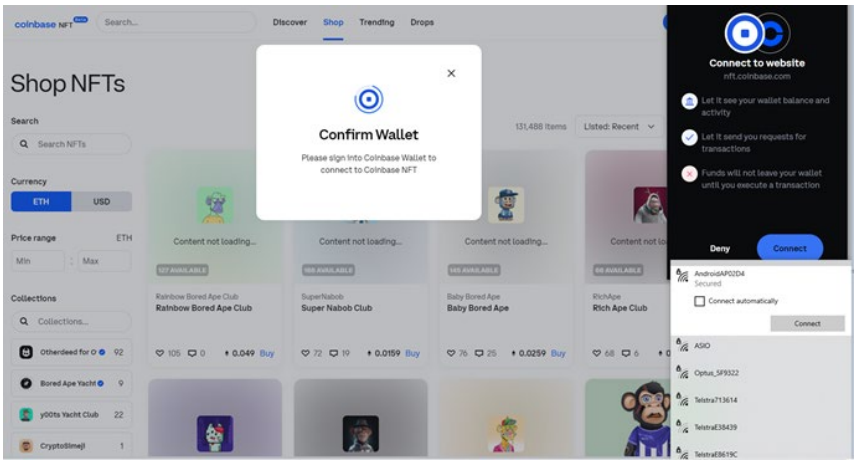
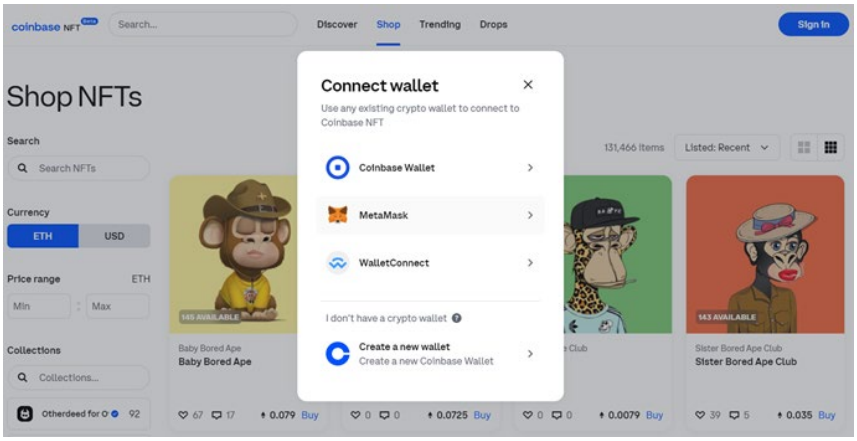
	• The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the key management vault/software. • The key management vault/software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair.
• *a network interface for receiving terms, the terms comprising:*	The Client device requires a network interface in order to connect a wallet to the marketplace to offer the NFT for sale. 1. Click on the 'Sign In' button top RHS.  2. Selecting 'Coinbase Wallet' prompts the user to connect to the website.  3. Attempting to connect without a network interface (as shown with no Wifi connected).

EXHIBIT 3



4. The user is not connected to the NFT marketplace and is instead requested to connect wallet.



5. When connected to a network the user creates a username and password that is associated from the wallet signature.

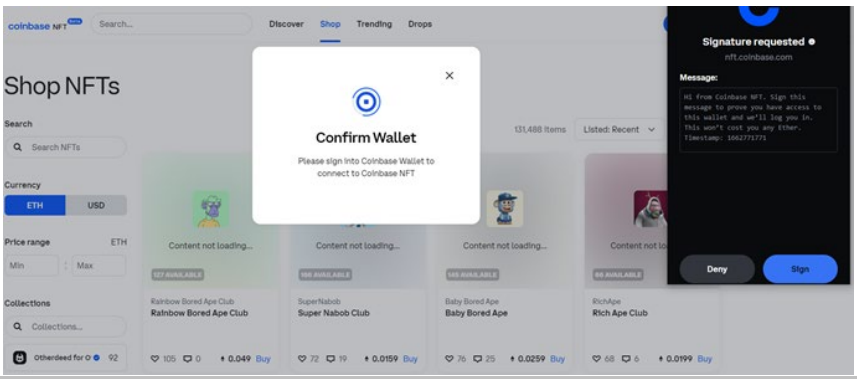


EXHIBIT 3

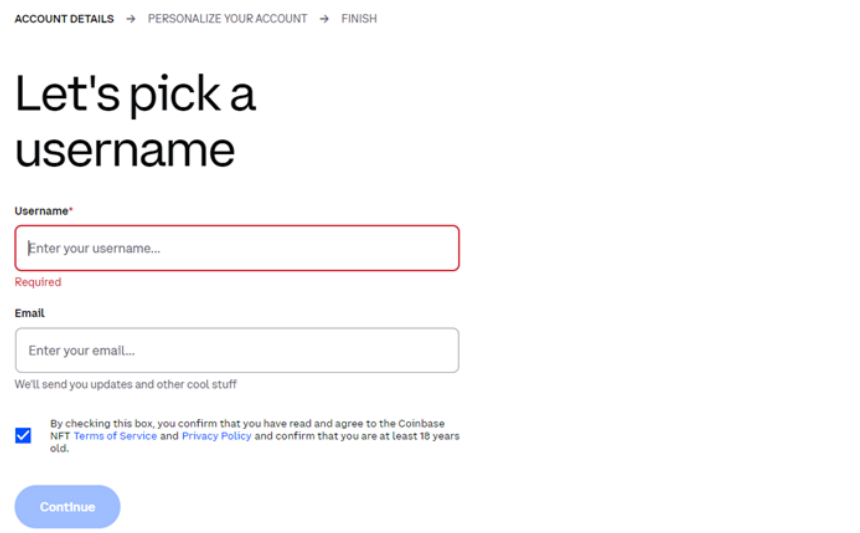
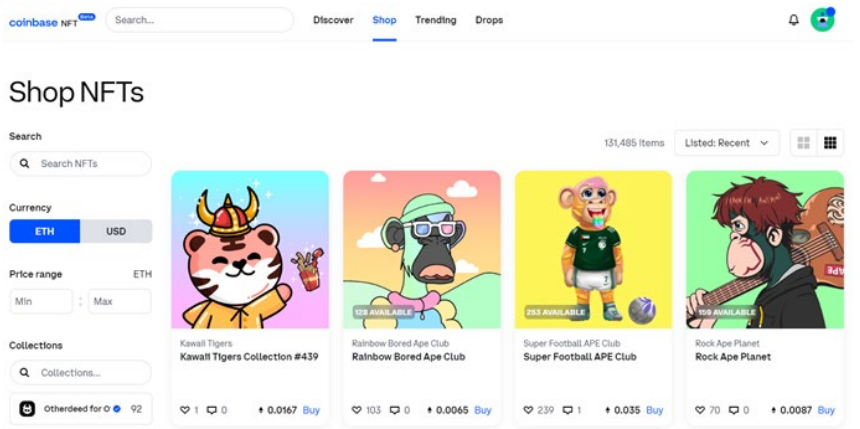
	 6. The user is logged in as indicated at the top RHS icon. 
o *at least one of a first principal data or a second principal data;*	**First principle data** The first principle data is the value represented/encapsulated by the NFT. Examples include but are not limited to art, gifs, videos, files, documents, deeds representing a real world asset, agreements.

EXHIBIT 3

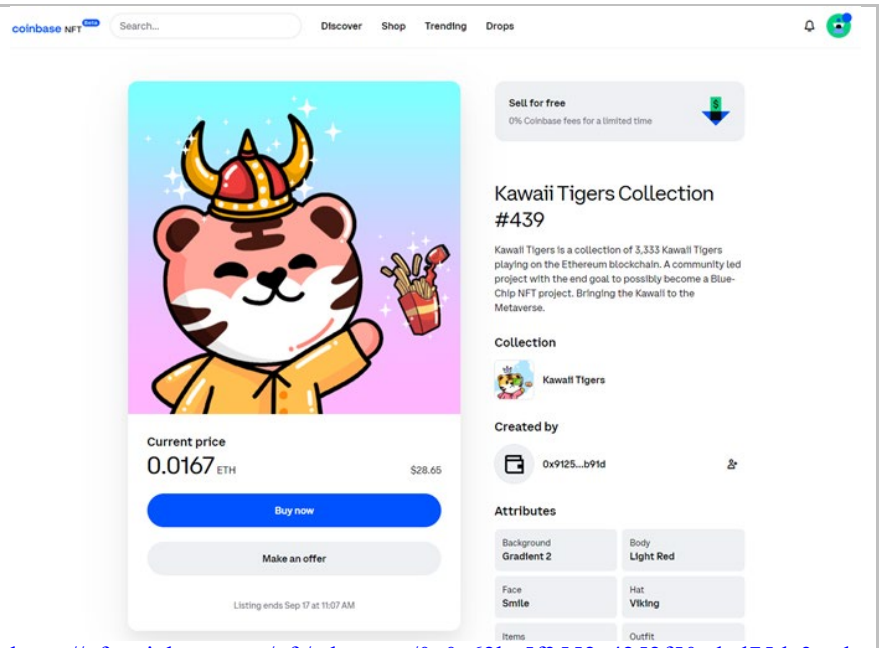
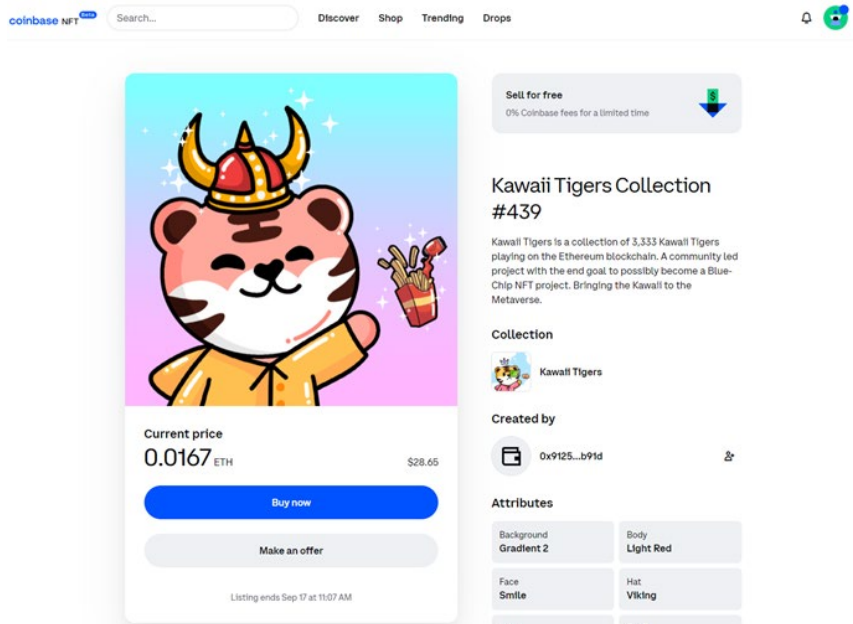
	 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439 **Second principle data** The seller will define a minimum or market price that they are willing to sell the NFT for.  https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439
o *a reference to at least one of a first data source or a second*	**First data source** The NFT smart contract which contains a metadata URI link describes the value.

EXHIBIT 3

data source; and	Metadata URI Link `/// @notice A distinct Uniform Resource Identifier (URI) for a given asset.` `/// @dev Throws if \`_tokenId\` is not a valid NFT. URIs are defined in RFC` `///  3986. The URI may point to a JSON file that conforms to the "ERC721` `///  Metadata JSON Schema".` `function tokenURI(uint256 _tokenId) external view returns (string);` https://eips.ethereum.org/EIPS/eip-721 **Second data source** The NFT smart contract which contains a royalty payment fee as per ERC2981 for consideration of the disbursement function. Royalty Info `/// @notice Called with the sale price to determine how much royalty is owed and to /// whom.` `/// @param _tokenId - the NFT asset queried for royalty information` `/// @param _salePrice - the sale price of the NFT asset specified by _tokenId` `/// @return receiver - address of who should be sent the royalty payment` `/// @return royaltyAmount - the royalty payment amount for _salePrice` `function royaltyInfo(` `  uint256 _tokenId,` `  uint256 _salePrice` `) external view returns (` `  address receiver,` `  uint256 royaltyAmount` `);` https://eips.ethereum.org/EIPS/eip-2981

EXHIBIT 3

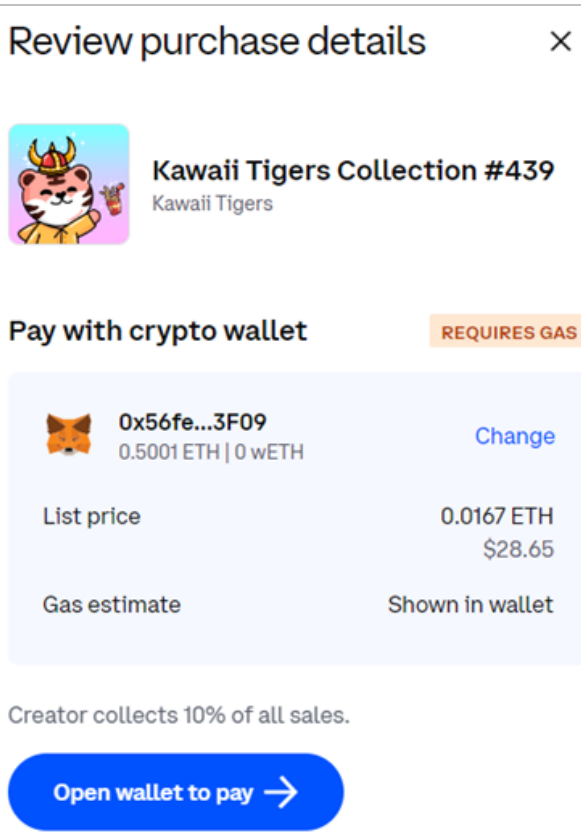
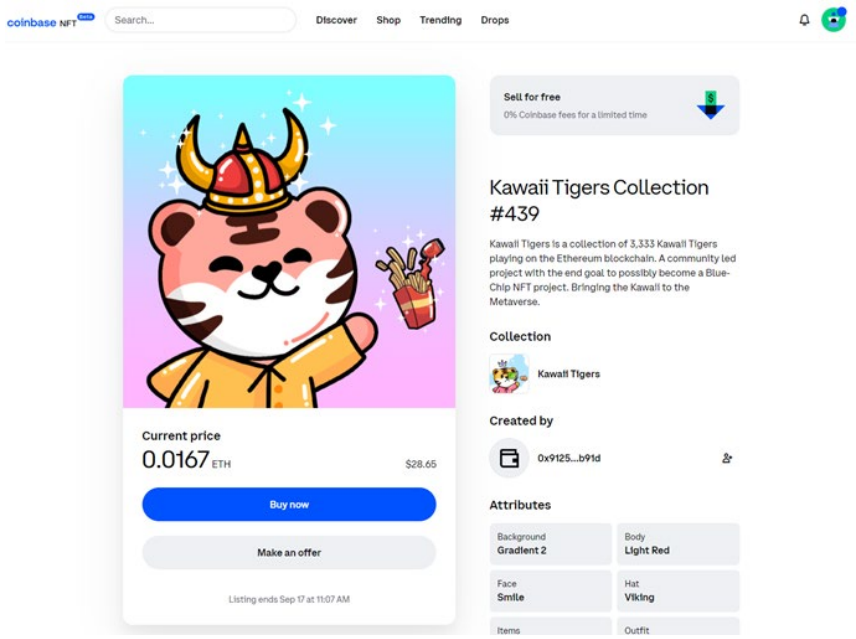
	 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439 Indicates that 10% of sale goes to the Creator.
o *an expiration timestamp;*	The approval by the NFT owner to sell or allow a transfer of ownership of the NFT is implied to be infinite or until accepted by a purchaser as no explicit deadline is mandated. The approve function as defined by the EIP-721 is as follows. `/// @notice Change or reaffirm the approved address for an NFT` `/// @dev The zero address indicates there is no approved address. Throws unless msg.sender is the current NFT owner, or an authorized operator of the current owner.` `/// @param _approved The new approved NFT controller` `/// @param _tokenId The NFT to approve` `function approve(address _approved, uint256 _tokenId) external payable;`

EXHIBIT 3



https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439

Shows an expiration with the Listing ending Sep 17 at 11:07am.

Note that the expiration period can also be represented through an event which in this case can be a subsequent all to the approve function to change the approved address back to the NFT owner. The description of the patent allows for the terms to define a point in time 'on or after the expiration timestamp or at a time or upon an event as defined by the terms…'. In this instance the event is the reception of the transaction for processing.

Patent references:
[0123] 22. On or after the expiration timestamp or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …

[0186] 21. On or after the expiration timestamp, or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …

Also note that there is a draft EIP-4494 which seeks to change this infinite time to a specified deadline expiration. This demonstrates that the 'approve' function does have a time component albeit undefined and implied.

https://eips.ethereum.org/EIPS/eip-4494

```
/// @notice Function to approve by way of owner signature
/// @param spender the address to approve
/// @param tokenId the index of the NFT to approve the
spender on
/// @param deadline a timestamp expiry for the permit
/// @param sig a traditional or EIP-2098 signature
```

EXHIBIT 3

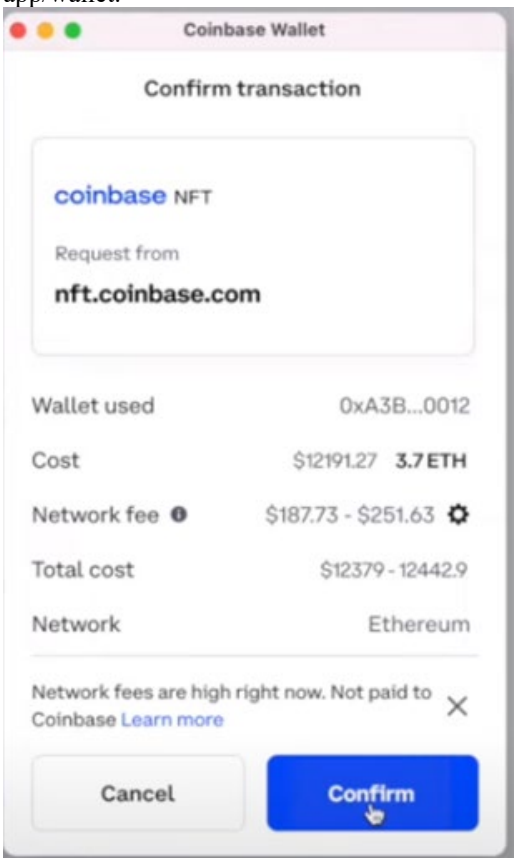
	`function permit(address spender, uint256 tokenId, uint256 deadline, bytes memory sig) external;`
• *a computer processor coupled to the memory and the network interface, the computer processor configured to:*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone.
i. *read the first private key from the memory;*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via the Wallet private key either before using the App or for sign individual transactions. The private key is encrypted on the device and accessed for reading (and subsequently used to calculate a transaction signature) by the app/wallet from the passcode/protection mechanism of the app/wallet.  Coinbase Wallet requesting confirmation to access the private key to authorise and sign the transaction.
ii. *compute a first cryptographic signature from the first private key;*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via a Wallet before using the App.

EXHIBIT 3

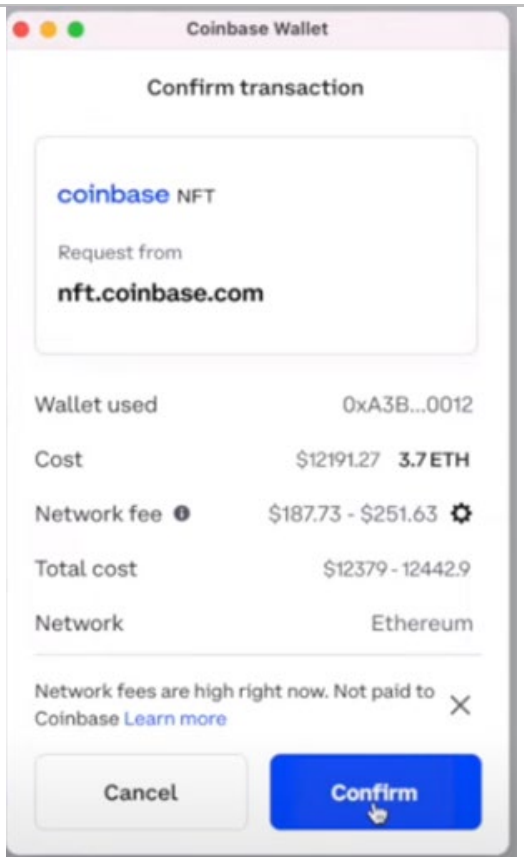
		 Coinbase Wallet requesting confirmation to access the private key to authorise, calculate and sign the transaction. **eth_SignTransaction** web3.eth.signTransaction(transactionObject, address [, callback]) Signs a transaction. Parameters Object - The transaction data to sign. See web3.eth.sendTransaction() for more. String - Address to sign transaction with. Function - (optional) Optional callback, returns an error object as first parameter and the result as second. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#signtransaction
iii.	*create an inchoate data record comprising:*	While the Computing Device and the First Client device are the same device, the Inchoate Transaction is created for the NFT seller to approve an account (typically a smart contract account) to transfer ownership of the NFT when the terms are met. The App (First Client as part of the Computing Device) creates a transaction message to be sent to the Ethereum Network. RPC JSON API calls are defined to allow interaction with the Ethereum network. The App constructs the following message which is aligned with this standard. **eth_sendTransaction** (EIP1559) Creates new message call transaction or a contract creation, if the data field contains code.

EXHIBIT 3

Object - The transaction object

1. from - String|Number: The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified. Or an address or index of a local wallet in web3.eth.accounts.wallet.
2. to - String: (optional) The destination address of the message, left undefined for a contract-creation transaction.
3. value - Number|String|BN|BigNumber: (optional) The value transferred for the transaction in wei, also the endowment if it's a contract-creation transaction.
4. gas - Number: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded).
5. gasPrice - Number|String|BN|BigNumber: (optional) The price of gas for this transaction in wei, defaults to web3.eth.gasPrice.
6. type - Number|String|BN|BigNumber: (optional) A positive unsigned 8-bit number between 0 and 0x7f that represents the type of the transaction.
7. maxFeePerGas - Number|String|BN: (optional, defaulted to (2 * block.baseFeePerGas) + maxPriorityFeePerGas) The maximum fee per gas that the transaction is willing to pay in total
8. maxPriorityFeePerGas - Number|String|BN (optional, defaulted to 1 Gwei) The maximum fee per gas to give miners to incentivize them to include the transaction (Priority fee)
9. accessList - List of hexstrings (optional) a list of addresses and storage keys that the transaction plans to access
10. data - String: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code.
11. nonce - Number: (optional) Integer of the nonce. This allows to overwrite your own pending transactions that use the same nonce.
12. chain - String: (optional) Defaults to mainnet.
13. hardfork - String: (optional) Defaults to london.

Returns

1. DATA, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available.

https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#sendtransaction

The App (First Client as part of the Computing Device) initiates a smart contract call populating the data field of the above message.

From: https://eips.ethereum.org/EIPS/eip-721

```
/// @notice Change or reaffirm the approved address for an
NFT
/// @dev The zero address indicates there is no approved
address. Throws unless `msg.sender` is the current NFT
owner, or an authorized operator of the current owner.
/// @param _approved The new approved NFT controller
/// @param _tokenId The NFT to approve
function approve(address _approved, uint256 _tokenId)
external payable;
```

EXHIBIT 3

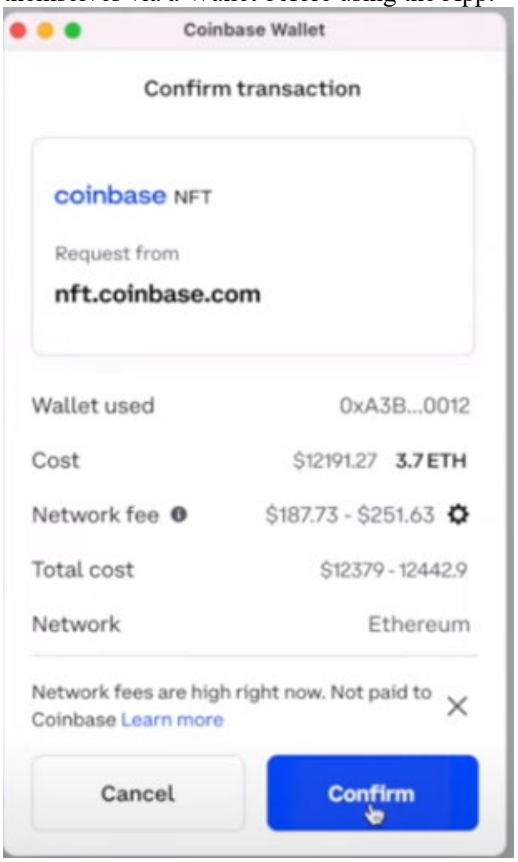
• *a commit input for receiving a commit data from a commit transaction;*	The commit input is the offer to sell the NFT indicated by an 'approve' function call. From: https://eips.ethereum.org/EIPS/eip-721 `/// @notice Change or reaffirm the approved address for an NFT` `/// @dev The zero address indicates there is no approved address. Throws unless msg.sender is the current NFT owner, or an authorized operator of the current owner.` `/// @param _approved The new approved NFT controller` `/// @param _tokenId The NFT to approve` `function approve(address _approved, uint256 _tokenId) external payable;`
• *one or more output data obtained from at least one of the first principal data or the second principal data, and a value data from at least one of the first data source or the second data source; and*	The output data includes: • The signed acceptance of the offered NFT based on the value description of the Metadata URI link. • The disbursement amount to the seller and the originator based on the royalty fee definition of the NFT.
• *the first cryptographic signature; and*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via a Wallet before using the App.  Coinbase Wallet requesting confirmation to access the private key to authorise, calculate and sign the transaction.

EXHIBIT 3

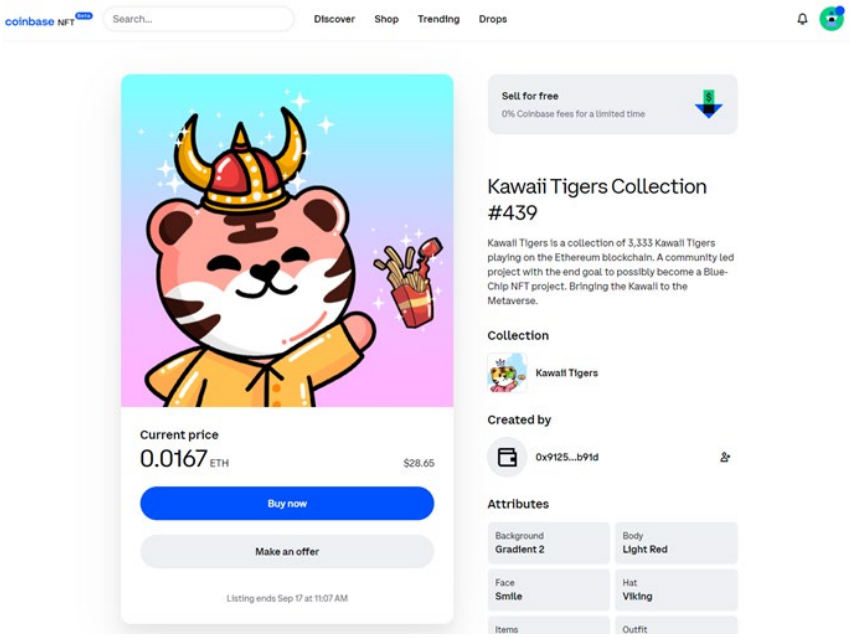
	eth_SignTransaction web3.eth.signTransaction(transactionObject, address [, callback]) Signs a transaction. Parameters Object - The transaction data to sign. See web3.eth.sendTransaction() for more. String - Address to sign transaction with. Function - (optional) Optional callback, returns an error object as first parameter and the result as second. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#signtransaction
iv. publish the inchoate data record to at least one of the first client device or the second client device,	The published Inchoate Data Record is the offered NFT for sale on a marketplace. The following is an example from the Coinbase NFT Marketplace.  https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439
wherein the decentralized digital currency comprises a distributed ledger that enables processing the transaction between the first client device and the second client device without the need for a trusted central authority,	Ether is used as the decentralised currency. The Ethereum network maintains a distributed ledger without the need for a trusted central authority. From the Ethereum yellow paper. *2.1. Value. In order to incentivise computation within the network, there needs to be an agreed method for transmitting value. To address this issue, Ethereum has an intrinsic currency, Ether, known also as ETH and sometimes referred to by the Old English ¯D.* https://ethereum.github.io/yellowpaper/paper.pdf
wherein the inchoate data record is used by at least one of the first client device or the second client device to create a complete data	The NFT approved for sale is accepted by the buyer with the safeTransfer function call to transfer ownership to the buyer.

EXHIBIT 3

record and to create the transaction by broadcasting the complete data record for transmitting and receiving among network participants in the computer network for recording in the distributed ledger,	The App (Second Client as part of the Facilitator) creates a transaction message to be sent to the Ethereum Network. RPC JSON API calls are defined to allow interaction with the Ethereum network. The App constructs the following message which is aligned with this standard. **eth_sendTransaction** (EIP1559) Creates new message call transaction or a contract creation, if the data field contains code. Object - The transaction object 1. from - String|Number: The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified. Or an address or index of a local wallet in web3.eth.accounts.wallet. 2. to - String: (optional) The destination address of the message, left undefined for a contract-creation transaction. 3. value - Number|String|BN|BigNumber: (optional) The value transferred for the transaction in wei, also the endowment if it's a contract-creation transaction. 4. gas - Number: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded). 5. gasPrice - Number|String|BN|BigNumber: (optional) The price of gas for this transaction in wei, defaults to web3.eth.gasPrice. 6. type - Number|String|BN|BigNumber: (optional) A positive unsigned 8-bit number between 0 and 0x7f that represents the type of the transaction. 7. maxFeePerGas - Number|String|BN: (optional, defaulted to (2 * block.baseFeePerGas) + maxPriorityFeePerGas) The maximum fee per gas that the transaction is willing to pay in total 8. maxPriorityFeePerGas - Number|String|BN (optional, defaulted to 1 Gwei) The maximum fee per gas to give miners to incentivize them to include the transaction (Priority fee) 9. accessList - List of hexstrings (optional) a list of addresses and storage keys that the transaction plans to access 10. data - String: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code. 11. nonce - Number: (optional) Integer of the nonce. This allows to overwrite your own pending transactions that use the same nonce. 12. chain - String: (optional) Defaults to mainnet. 13. hardfork - String: (optional) Defaults to london. Returns 2. DATA, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#sendtransaction The App (Second Client as part of the Facilitator) initiates a smart contract call populating the data field of the above message. From: https://eips.ethereum.org/EIPS/eip-721

EXHIBIT 3

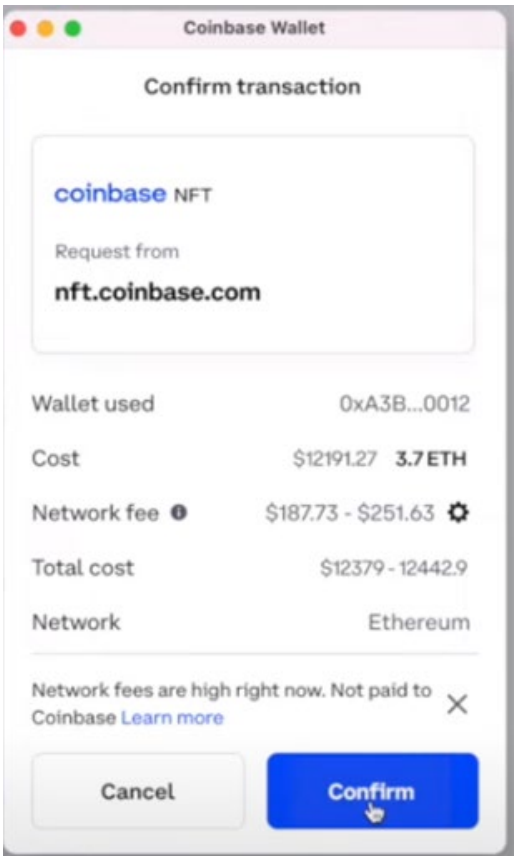
	ERC-721 standardizes a safe transfer function `safeTransferFrom` *(overloaded with and without a* `bytes` *parameter) and an unsafe function* `transferFrom`*. Transfers may be initiated by:* • *The owner of an NFT* • *The approved address of an NFT* • *An authorized operator of the current owner of an NFT*
wherein at least one of the first client device or the second client device signs the inchoate data record and saves a copy of the inchoate data record on at least one of the first client device or the second client device; and	The Second Client requires the buyer to authenticate themselves via a Wallet before using the App. The wallet private key is used to sign the transaction for the EVM smart contract call.  Coinbase Wallet requesting confirmation to access the private key to authorise and sign the transaction. Note that it is not mandatory for the inchoate data record to be saved by the device as it is optional as described in the Patent general description. **[0095]** *11. The second client creates the complete commit transaction record by signing the inchoate commit transaction record and optionally saves a copy in non-transitory memory, the complete commit transaction record comprising:* **[0111]** *15. Optionally, the first client saves a copy of the complete commit transaction record in non-transitory memory.*

EXHIBIT 3

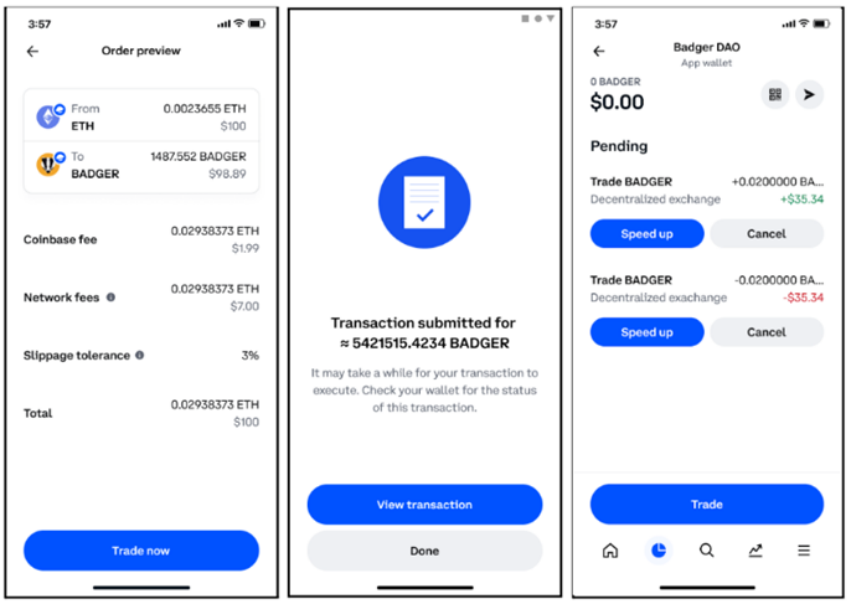
wherein the at least one of the computing device, the first client device, or the second client device verifies the recording of the complete data record in the distributed ledger by observing an external state.	The Second Client verifies that the Complete Data Record (Transaction) has been recorded by the network as a complete transaction record. In the below image this is shown in the 'History' tab with the transaction shown as confirmed.  https://help.coinbase.com/en/coinbase/trading-and-funding/trade-on-dex/check-transaction-status When a transaction is sent, a transaction hash is received. This can be used to retrieve transaction details. Use the JSON RPC API command getTransactionByHash({transaction hash}) to retrieve the transaction details. The returned blockNumber should be non-null if the transaction has been mined and included into a block. Then call JSON RPC API eth_blockNumber to get the current block height. The number of confirmations is the eth_blockNumber result minus the eth_getTransaction blockNumber result. https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_blocknumber https://github.com/ethereum/wiki/wiki/JSON-RPC#getTransactionByHash

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 2	**Coinbase Products & Services** **The transfer of a NFT (on the Ethereum network) from one party to another.**
The device of claim 1, where: the computer processor is configured to obtain the one or more output data based on:	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed or offered) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed or offered) Ethereum supporting Archive Nodes and Light Nodes; and • the Coinbase (Ethereum compatible) wallets; Where both the Coinbase Nodes and the Coinbase Ethereum compatible end user wallet device are networked to have direct or indirect communication with each other.
the first principal data; and the value data from the first data source.	The Validator node, as part of processing the received complete transaction, disburses the value by sending the NFT (with the metadata link) to the recipient. The Validator node, as part of processing the received complete transaction, disburses the royalty fee to the NFT originator address. Metadata URI Link */// @notice A distinct Uniform Resource Identifier (URI) for a given asset.* */// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC* */// 3986. The URI may point to a JSON file that conforms to the "ERC721* */// Metadata JSON Schema".* **function** tokenURI(**uint256** _tokenId) **external view returns** (**string**); https://eips.ethereum.org/EIPS/eip-721 Royalty Info */// @notice Called with the sale price to determine how much royalty is owed and to /// whom.* */// @param _tokenId - the NFT asset queried for royalty information* */// @param _salePrice - the sale price of the NFT asset specified by _tokenId* */// @return receiver - address of who should be sent the royalty payment* */// @return royaltyAmount - the royalty payment amount for _salePrice* **function** royaltyInfo(**uint256** _tokenId, **uint256** _salePrice) **external view returns** (**address** receiver, **uint256** royaltyAmount); https://eips.ethereum.org/EIPS/eip-2981

EXHIBIT 3

Claim Chart U.S. Patent No. 11,196,566 (the "'566 Patent") Coinbase	
Claim 7 A system for processing a transaction between a first client device and a second client device via a transfer mechanism the system comprising a computing device, the first client device, the second client device, and the transfer mechanism.	**Coinbase Products & Services** **The transfer of a NFT (on the Ethereum network) from one party to another.**
7. a. *the computing device comprising:* *i. a first memory comprising for storing a first asymmetric key pair, the first asymmetric key pair comprising a first private key and a first public key;*	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed or offered) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed or offered) Ethereum supporting Archive Nodes and Light Nodes; and • the Coinbase (Ethereum compatible) wallets; Where both the Coinbase Nodes and the Coinbase Ethereum compatible end user wallet device are networked to have direct or indirect communication with each other. **Client Device** The First Client is the end user device that accepts an offer for an NFT. The Second Client is the end user device that authorises an NFT to be offered for sale. Three (3) Client instances have been identified that represent various implementations that exist. 1. Client is a device running an App (Coinbase Wallet) that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js. This device will also use a key storage wallet with EIP-191 or EIP-712 signing support (such as Metamask/Coinbase Wallet) to send an NFT transactions. The use of an EIP-1193 based Ethereum Provider API or EIP-191/EIP-712 signing support means the Client is considered to be part of the Ethereum Network. http://eips/ethereum.org/EIPS/eip-191 http://eips/ethereum.org/EIPS/eip-712 http://eips/ethereum.org/EIPS/eip-1193 2. The Computing Device and the Client are the same device where the Client is a Client Browser or Command Line Interface on a Coinbase Ethereum Node used to create, sign and submit NFT transactions to the Coinbase Ethereum Node for processing. 3. Client is a device running an App (Coinbase Wallet) that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js.. This device will also use a key storage wallet. This device uses private key management in order to sign transactions

EXHIBIT 3

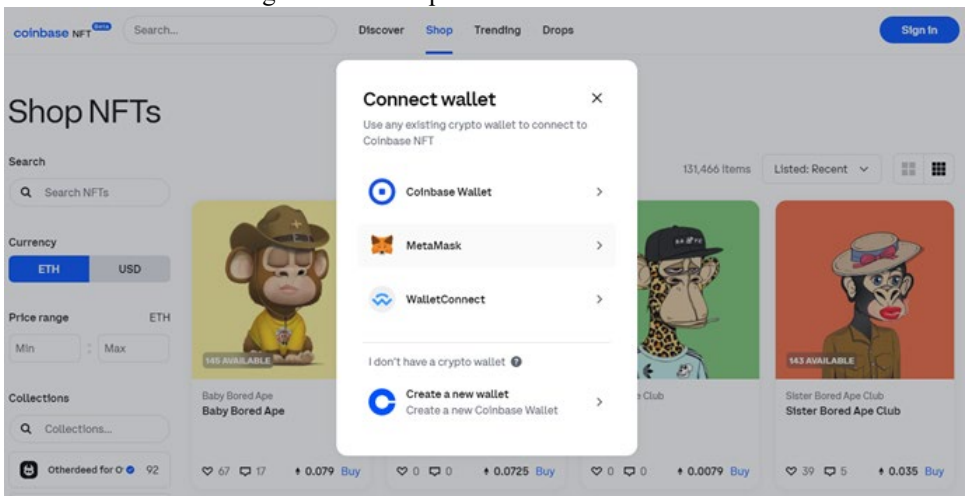
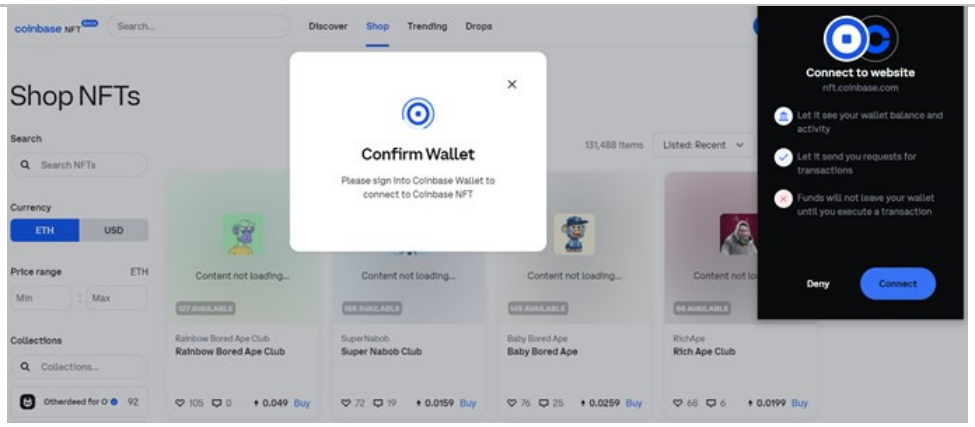
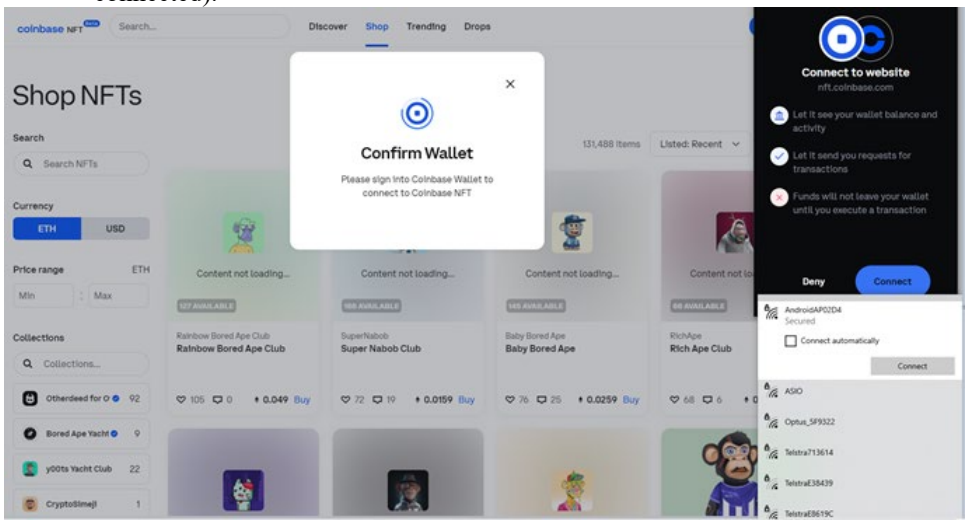
		Where the Client running the App (Coinbase Wallet) software, as part of the Facilitator, need a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer or mobile phone). • Transaction record sector (stores transactions that haven't been submitted to the blockchain yet) kept via the crypto software wallets Where the Client running the App (Coinbase Wallet) software, contains: • a first key pair sector which is generated and stored in the wallet software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the wallet software. • The wallet software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the Client interacts with the Eth Client Browser or Command Line Interface to transaction creation, contains: • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key. Where the Client interacts with App (Coinbase Wallet) with Private key management in order to sign transactions, contains: • a first key pair sector which is stored in the key management vault/software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the key management vault/software. • The key management vault/software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair.
ii.	*a first network interface for receiving terms, the terms comprising:*	The Client device requires a network interface in order to connect a wallet to the marketplace to offer the NFT for sale. 1. Click on the 'Sign In' button top RHS.  2. Selecting 'Coinbase Wallet' prompts the user to connect to the website.

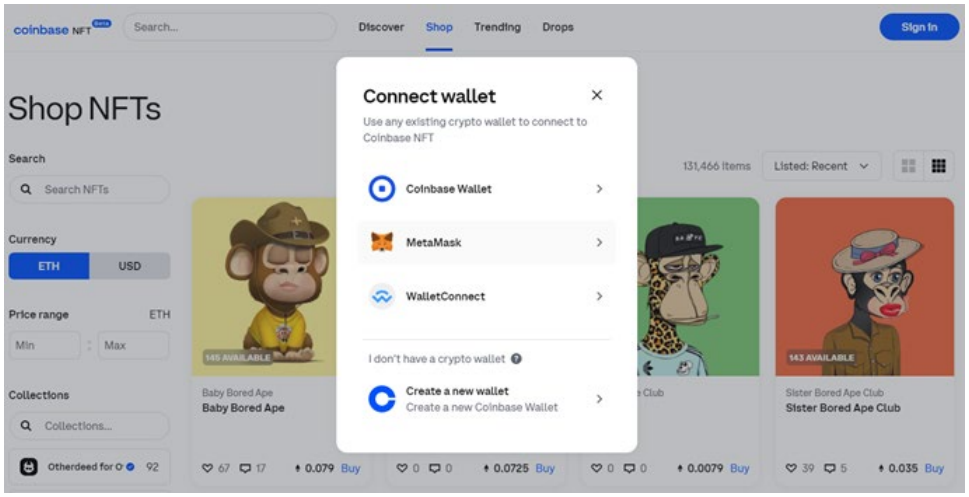
EXHIBIT 3



3. Attempting to connect without a network interface (as shown with no Wifi connected).



4. The user is not connected to the NFT marketplace and is instead requested to connect wallet.



5. When connected to a network the user creates a username and password that is associated from the wallet signature.

EXHIBIT 3

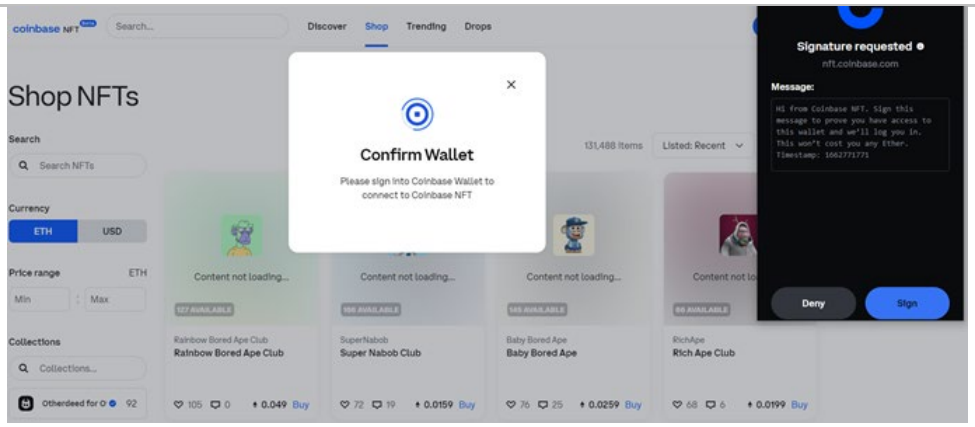
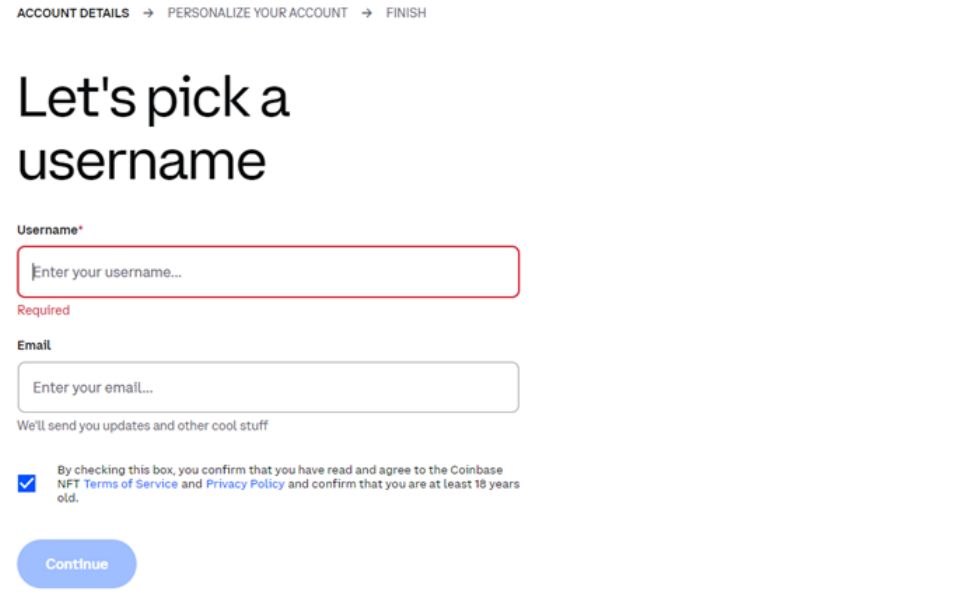
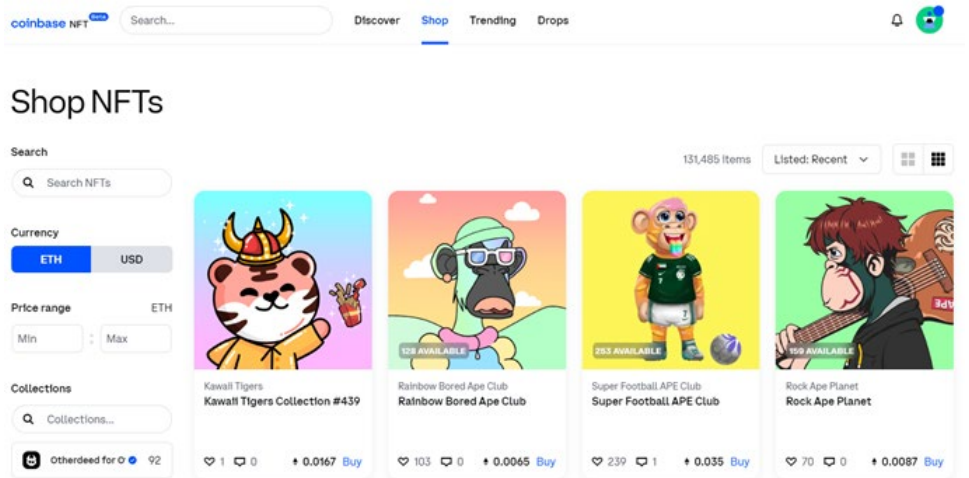
	  6. The user is logged in as indicated at the top RHS icon. 
A. at least one of a first principal data or a second principal data;	**First principle data** The first principle data is the value represented/encapsulated by the NFT. Examples include but are not limited to art, gifs, videos, files, documents, deeds representing a real world asset, agreements.

EXHIBIT 3

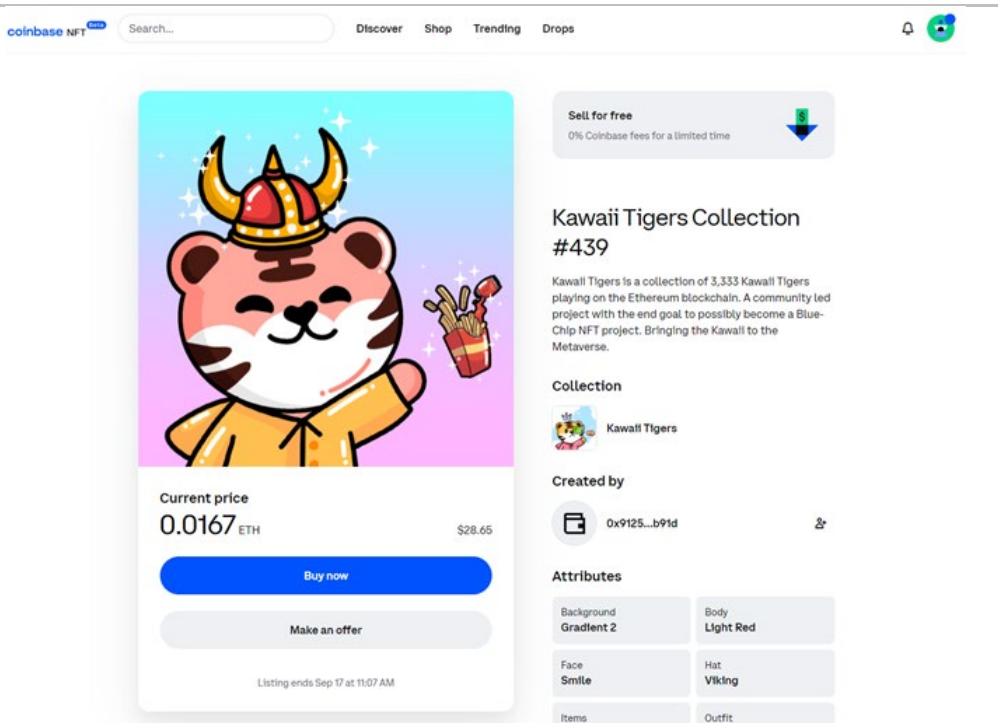
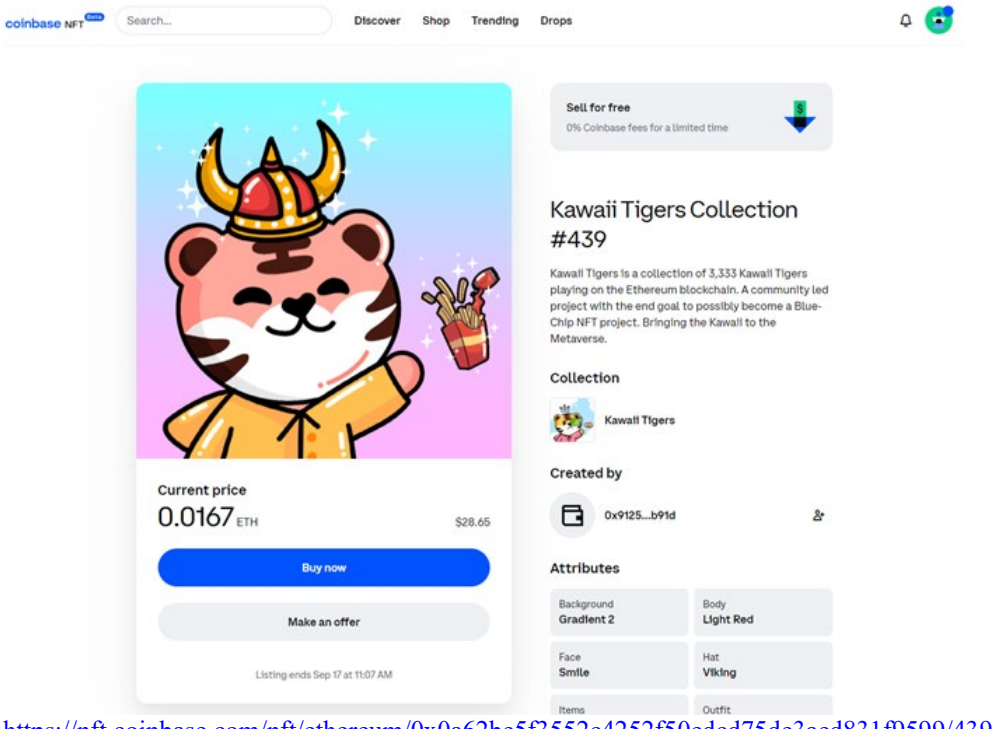
	 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439 **Second principle data** The seller will define a minimum or market price that they are willing to sell the NFT for.  https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439
B. a reference to at least one of a first data	**First data source** The NFT smart contract which contains a metadata URI link describes the value.

EXHIBIT 3

source or a second data source; and	Metadata URI Link `/// @notice A distinct Uniform Resource Identifier (URI) for a given asset.` `/// @dev Throws if \`_tokenId\` is not a valid NFT. URIs are defined in RFC` `///  3986. The URI may point to a JSON file that conforms to the "ERC721` `///  Metadata JSON Schema".` `function tokenURI(uint256 _tokenId) external view returns (string);` https://eips.ethereum.org/EIPS/eip-721 **Second data source** The NFT smart contract which contains a royalty payment fee as per ERC2981 for consideration of the disbursement function. Royalty Info `/// @notice Called with the sale price to determine how much royalty is owed and to /// whom.` `/// @param _tokenId - the NFT asset queried for royalty information` `/// @param _salePrice - the sale price of the NFT asset specified by _tokenId` `/// @return receiver - address of who should be sent the royalty payment` `/// @return royaltyAmount - the royalty payment amount for _salePrice` `function royaltyInfo(` `  uint256 _tokenId,` `  uint256 _salePrice` `) external view returns (` `  address receiver,` `  uint256 royaltyAmount` `);` https://eips.ethereum.org/EIPS/eip-2981

EXHIBIT 3

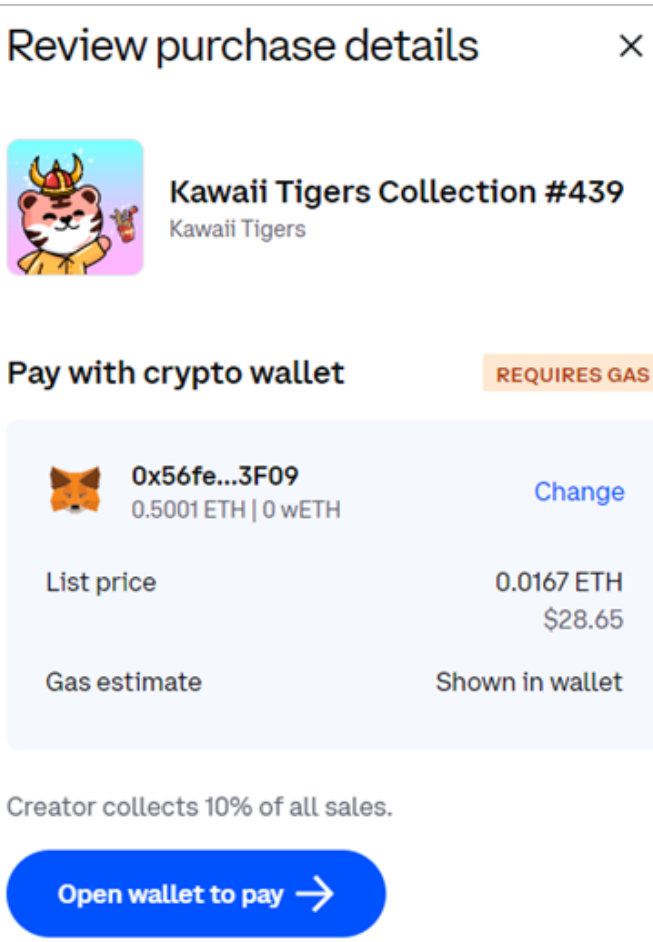
	 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439 Indicates that 10% of sale goes to the Creator.
C. *an expiration timestamp,*	The approval by the NFT owner to sell or allow a transfer of ownership of the NFT is implied to be infinite as no explicit deadline is configurable. The approve function as defined by the EIP-721 is as follows. `/// @notice Change or reaffirm the approved address for an NFT` `/// @dev The zero address indicates there is no approved address.` `Throws unless msg.sender is the current NFT owner, or an authorized operator of the current owner.` `/// @param _approved The new approved NFT controller` `/// @param _tokenId The NFT to approve` `function approve(address _approved, uint256 _tokenId) external payable;`

EXHIBIT 3

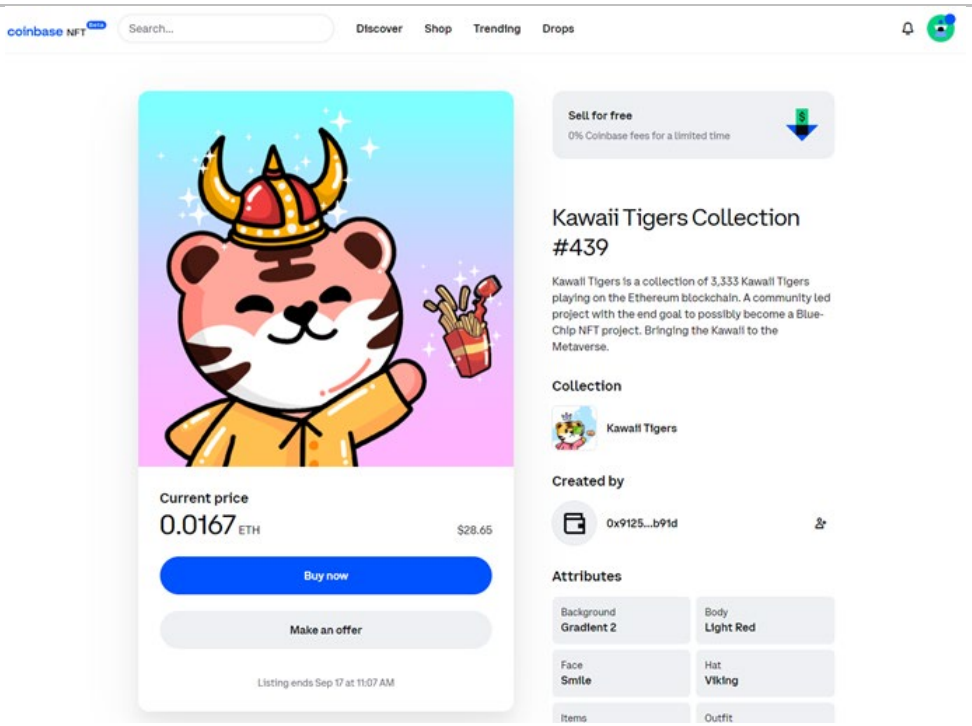
		 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439 Shows an expiration with the Listing ending Sep 17 at 11:07am. Note that the expiration period can be represented through an event which in this case can be a subsequent all to the approve function to change the approved address back to the NFT owner. The description of the patent allows for the terms to define a point in time 'on or after the expiration timestamp or at a time or upon an event as defined by the terms…'. In this instance the event is the reception of the transaction for processing. Patent references: *[0123] 22. On or after the expiration timestamp or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …* *[0186] 21. On or after the expiration timestamp, or at a time or upon an event as defined by the terms, and before the lock time of the complete refund transaction record, …* Also note that there is a draft EIP-4494 which seeks to change this infinite time to a specified deadline expiration. This demonstrates that the 'approve' function does have a time component albeit undefined and implied. https://eips.ethereum.org/EIPS/eip-4494 `/// @notice Function to approve by way of owner signature` `/// @param spender the address to approve` `/// @param tokenId the index of the NFT to approve the spender on` `/// @param deadline a timestamp expiry for the permit` `/// @param sig a traditional or EIP-2098 signature` `function permit(address spender, uint256 tokenId, uint256 deadline, bytes memory sig) external;`
iii.	*a first computer processor*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone.

EXHIBIT 3

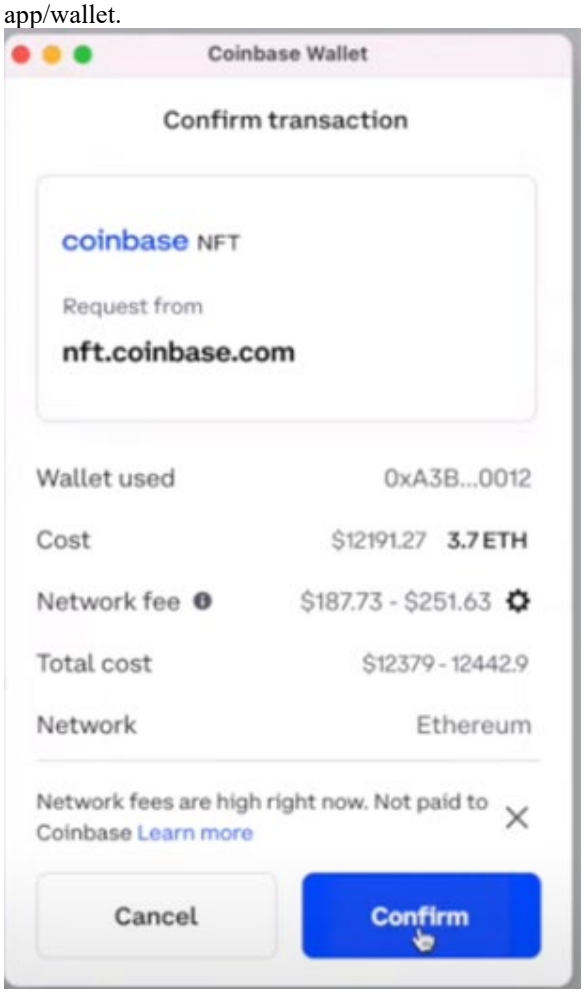
coupled to the first memory and the first network interface, the first computer processor configured to:	
A. read the first private key from the first memory	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via the Wallet private key either before using the App or for sign individual transactions. The private key is encrypted on the device and accessed for reading (and subsequently used to calculate a transaction signature) by the app/wallet from the passcode/protection mechanism of the app/wallet.  Coinbase Wallet requesting confirmation to access (read) the private key to authorise and sign the transaction.
B. compute a first cryptographic signature from the first private key;	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via a Wallet before using the App.

EXHIBIT 3

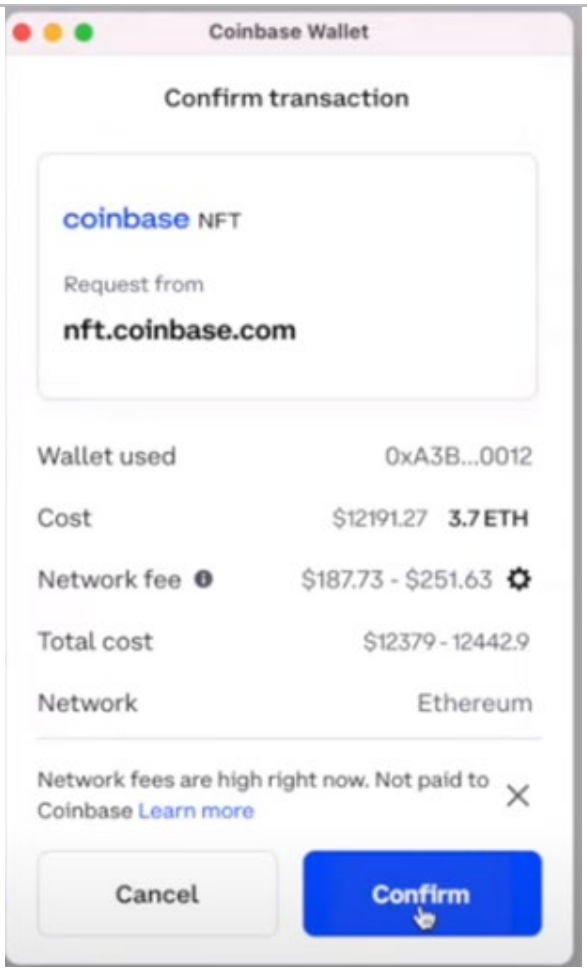
	 Coinbase Wallet requesting confirmation to access (read) the private key to authorise, calculate signature and sign the transaction. **eth_SignTransaction** web3.eth.signTransaction(transactionObject, address [, callback]) Signs a transaction. Parameters Object - The transaction data to sign. See web3.eth.sendTransaction() for more. String - Address to sign transaction with. Function - (optional) Optional callback, returns an error object as first parameter and the result as second. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#signtransaction
C. *create an inchoate data record comprising:*	While the Computing Device and the Second Client device are the same device, the Inchoate Transaction is created for the NFT seller to approve an account (typically a smart contract account) to transfer ownership of the NFT when the terms are met. The App (Coinbase Wallet) creates a transaction message to be sent to the Ethereum Network. RPC JSON API calls are defined to allow interaction with the Ethereum network. The App constructs the following message which is aligned with this standard. **eth_sendTransaction** (EIP1559) Creates new message call transaction or a contract creation, if the data field contains code.

EXHIBIT 3

		Object - The transaction object 1. from - String|Number: The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified. Or an address or index of a local wallet in web3.eth.accounts.wallet. 2. to - String: (optional) The destination address of the message, left undefined for a contract-creation transaction. 3. value - Number|String|BN|BigNumber: (optional) The value transferred for the transaction in wei, also the endowment if it's a contract-creation transaction. 4. gas - Number: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded). 5. gasPrice - Number|String|BN|BigNumber: (optional) The price of gas for this transaction in wei, defaults to web3.eth.gasPrice. 6. type - Number|String|BN|BigNumber: (optional) A positive unsigned 8-bit number between 0 and 0x7f that represents the type of the transaction. 7. maxFeePerGas - Number|String|BN: (optional, defaulted to (2 * block.baseFeePerGas) + maxPriorityFeePerGas) The maximum fee per gas that the transaction is willing to pay in total 8. maxPriorityFeePerGas - Number|String|BN (optional, defaulted to 1 Gwei) The maximum fee per gas to give miners to incentivize them to include the transaction (Priority fee) 9. accessList - List of hexstrings (optional) a list of addresses and storage keys that the transaction plans to access 10. data - String: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code. 11. nonce - Number: (optional) Integer of the nonce. This allows to overwrite your own pending transactions that use the same nonce. 12. chain - String: (optional) Defaults to mainnet. 13. hardfork - String: (optional) Defaults to london. Returns 3. DATA, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#sendtransaction The App (Coinbase Wallet) initiates a smart contract call populating the data field of the above message. From: https://eips.ethereum.org/EIPS/eip-721 `/// @notice Change or reaffirm the approved address for an NFT` `/// @dev The zero address indicates there is no approved address.` `Throws unless msg.sender is the current NFT owner, or an` `authorized operator of the current owner.` `/// @param _approved The new approved NFT controller` `/// @param _tokenId The NFT to approve` `function approve(address _approved, uint256 _tokenId) external payable;`
I.	*a commit input for receiving a commit data*	The commit input is the offer to sell the NFT indicated by an 'approve' function call. From: https://eips.ethereum.org/EIPS/eip-721

EXHIBIT 3

	from a commit transaction;	`/// @notice Change or reaffirm the approved address for an NFT` `/// @dev The zero address indicates there is no approved address.` `Throws unless` `msg.sender` `is the current NFT owner, or an authorized operator of the current owner.` `/// @param _approved The new approved NFT controller` `/// @param _tokenId The NFT to approve` **function** approve(**address** _approved, **uint256** _tokenId) **external payable;**
II.	*one or more outputs obtained from at least one of the first principal data or the second principal data, and a value data from at least one of the first data source or the second data source; and;*	The output data includes: • The signed acceptance of the offered NFT based on the value description of the Metadata URI link. • The disbursement amount to the seller and the originator based on the royalty fee definition of the NFT.
III.	*The first cryptographic signature; and*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via a Wallet before using the App.

EXHIBIT 3

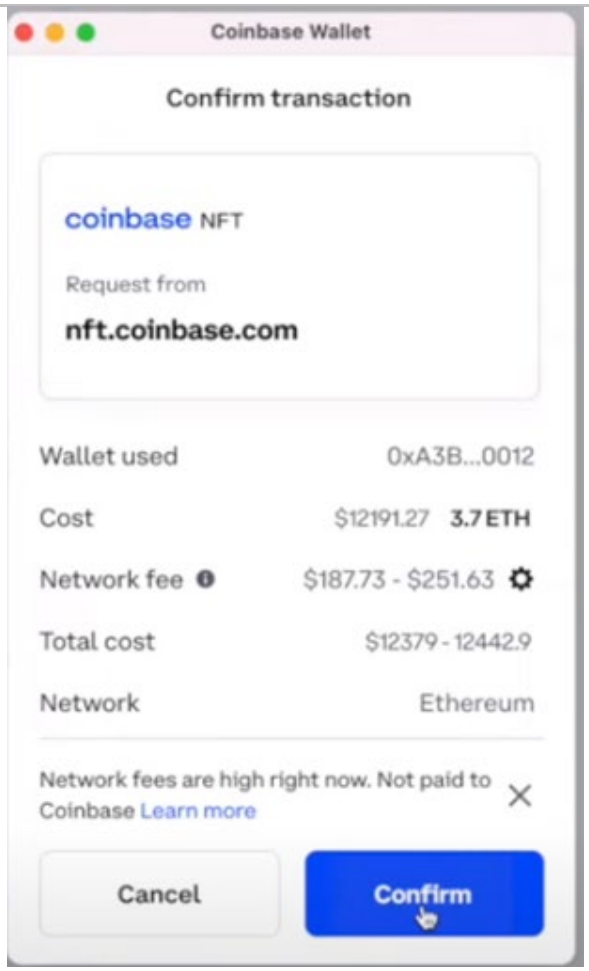
	 Coinbase Wallet requesting confirmation to access the private key to authorise and sign the transaction. **eth_SignTransaction** web3.eth.signTransaction(transactionObject, address [, callback]) Signs a transaction. Parameters Object - The transaction data to sign. See web3.eth.sendTransaction() for more. String - Address to sign transaction with. Function - (optional) Optional callback, returns an error object as first parameter and the result as second. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#signtransaction
D. *publish the inchoate data record to at least one of the first client device or the second client device;;*	The published Inchoate Data Record is the offered NFT for sale on a marketplace. The following is an example from the Coinbase NFT Marketplace.

EXHIBIT 3

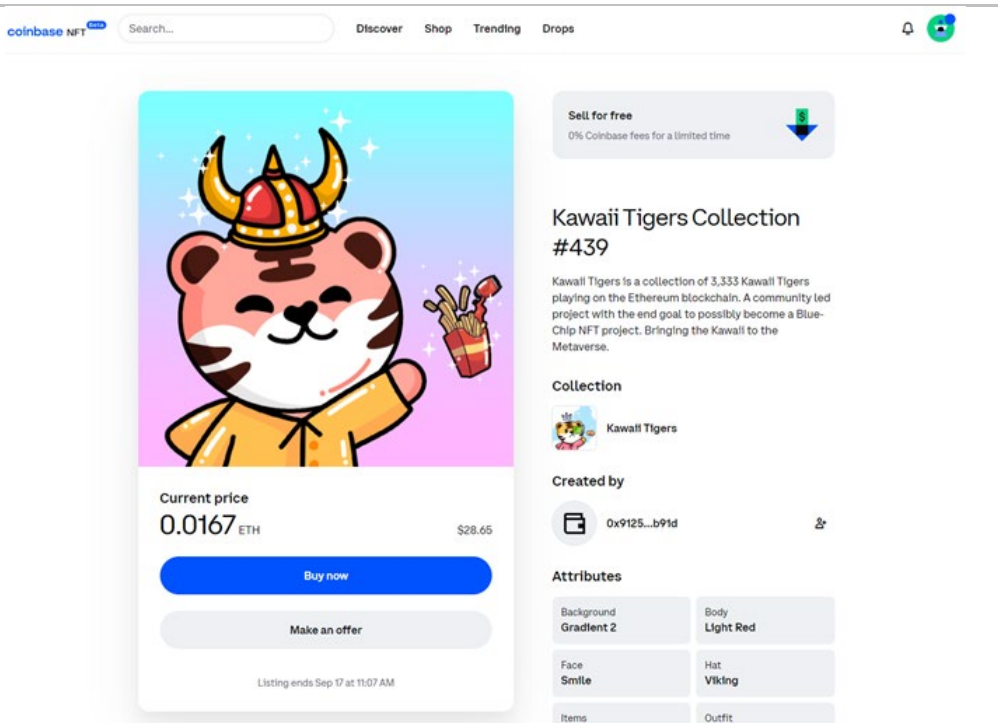
	 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439
7. b. the first client comprises: *i. a second memory for storing a second asymmetric key pair, the second asymmetric key pair comprising a second private key and a second public key.*	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed or offered) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed or offered) Ethereum supporting Archive Nodes and Light Nodes; and • the Coinbase (Ethereum compatible) wallets; Where both the Coinbase Nodes and the Coinbase Ethereum compatible end user wallet device are networked to have direct or indirect communication with each other. **Client Device** The First Client is the end user device that accepts an offer for an NFT. The Second Client is the end user device that authorises an NFT to be offered for sale. Three (3) Client instances have been identified that represent various implementations that exist. 1. Client is a device running an App (Coinbase Wallet) that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js. This device will also use a key storage wallet with EIP-191 or EIP-712 signing support (such as Metamask/Coinbase Wallet) to send an NFT transactions. The use of an EIP-1193 based Ethereum Provider API or EIP-191/EIP-712 signing support means the Client is considered to be part of the Ethereum Network. http://eips/ethereum.org/EIPS/eip-191 http://eips/ethereum.org/EIPS/eip-712 http://eips/ethereum.org/EIPS/eip-1193 2. The Computing Device and the Client are the same device where the Client is a Client Browser or Command Line Interface on a Coinbase Ethereum Node used to create, sign and submit NFT transactions to the Coinbase Ethereum Node for processing.

EXHIBIT 3

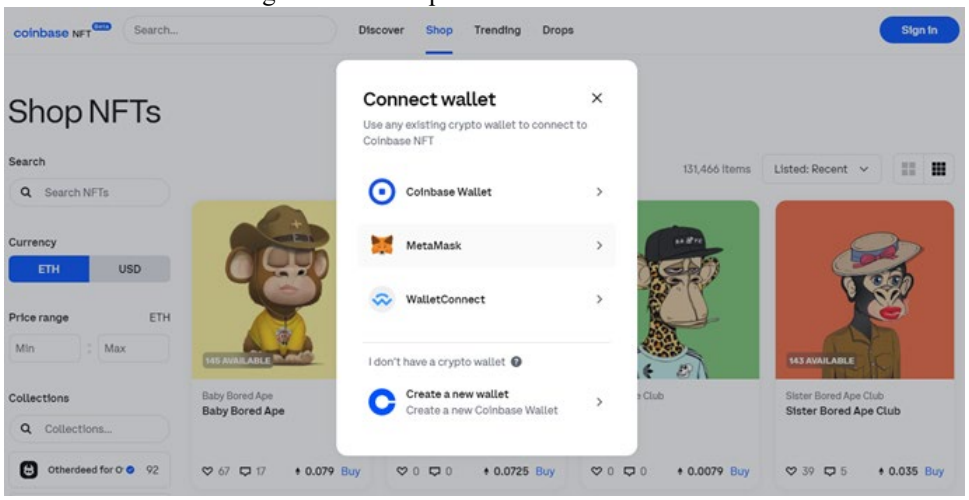
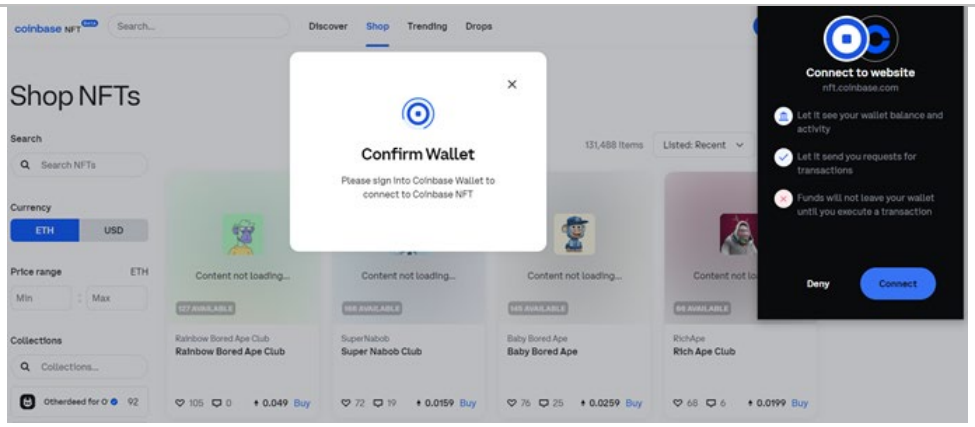
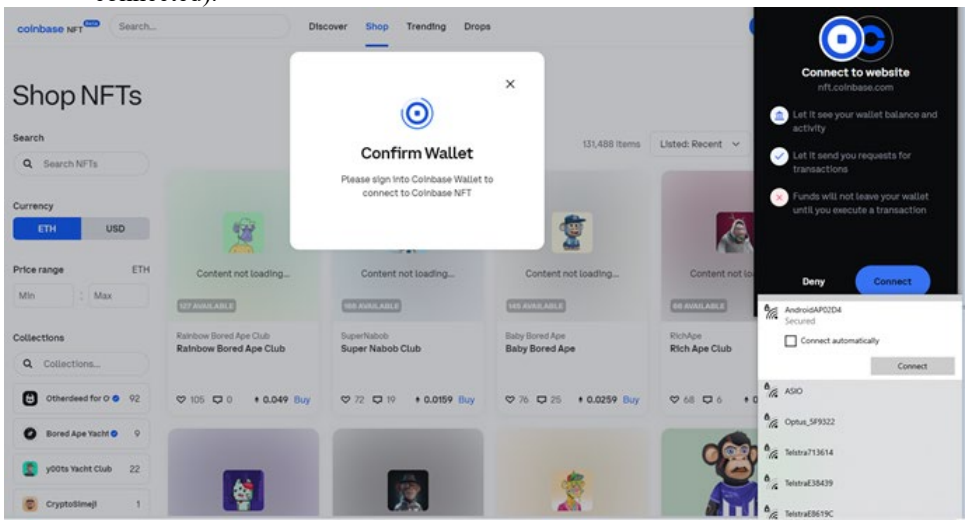
	3. Client is a device running an App (Coinbase Wallet) that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js.. This device will also use a key storage wallet. This device uses private key management in order to sign transactions Where the Client running the App (Coinbase Wallet) software, as part of the Facilitator, need a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer or mobile phone). • Transaction record sector (stores transactions that haven't been submitted to the blockchain yet) kept via the crypto software wallets Where the Client running the App (Coinbase Wallet) software, contains: • a first key pair sector which is generated and stored in the wallet software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the wallet software. • The wallet software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the Client interacts with the Eth Client Browser or Command Line Interface to transaction creation, contains: • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key. Where the Client interacts with App (Coinbase Wallet) with Private key management in order to sign transactions, contains: • a first key pair sector which is stored in the key management vault/software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the key management vault/software. • The key management vault/software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair.
ii. a second network interface; and	The Client device requires a network interface in order to connect a wallet to the marketplace to offer the NFT for sale. 1. Click on the ‘Sign In’ button top RHS.  2. Selecting ‘Coinbase Wallet’ prompts the user to connect to the website.

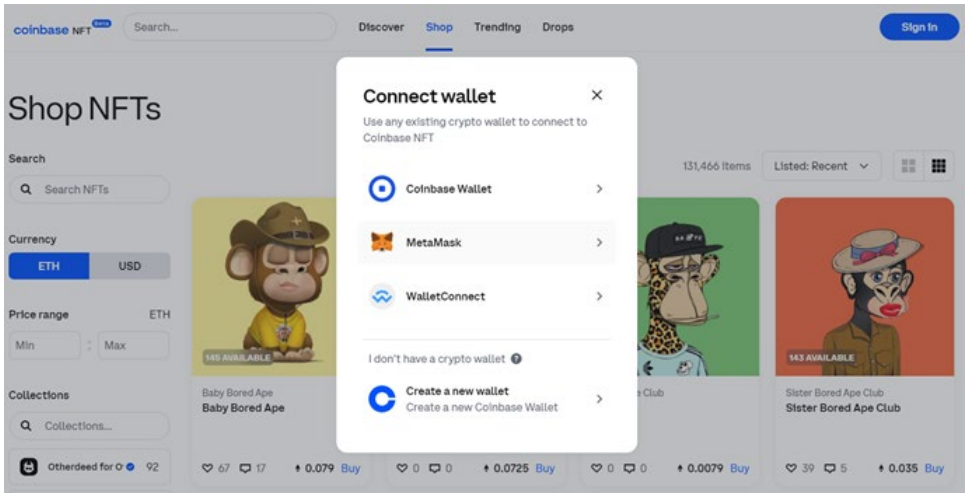
EXHIBIT 3



3. Attempting to connect without a network interface (as shown with no Wifi connected).

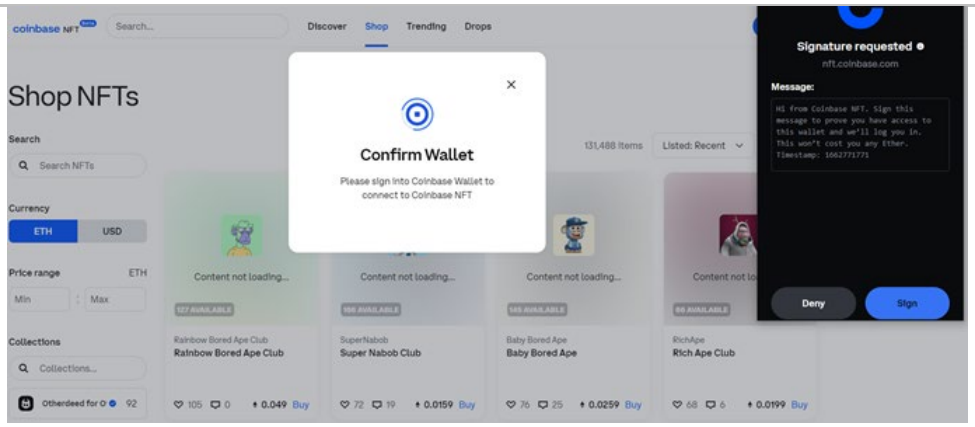


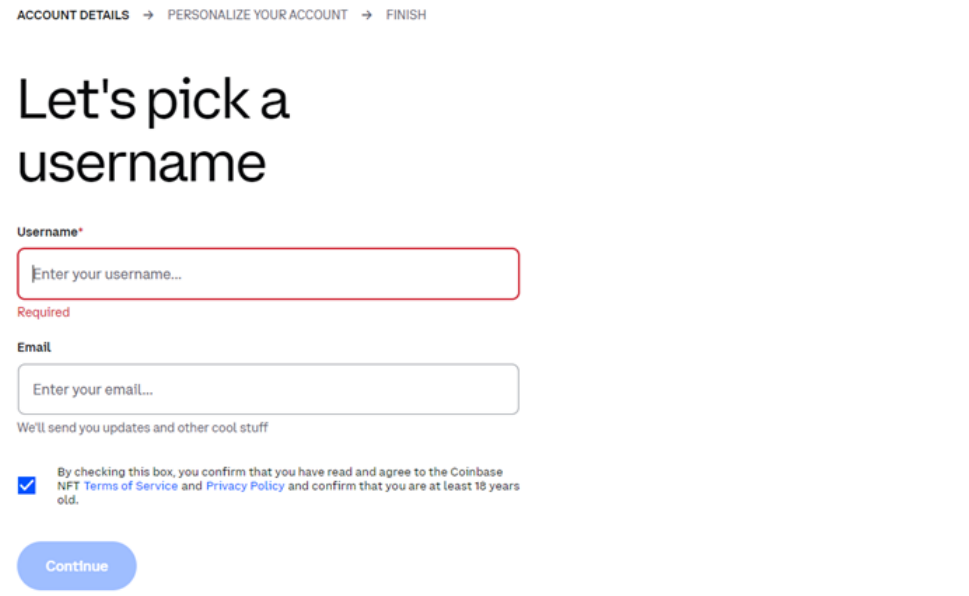
4. The user is not connected to the NFT marketplace and is instead requested to connect wallet.



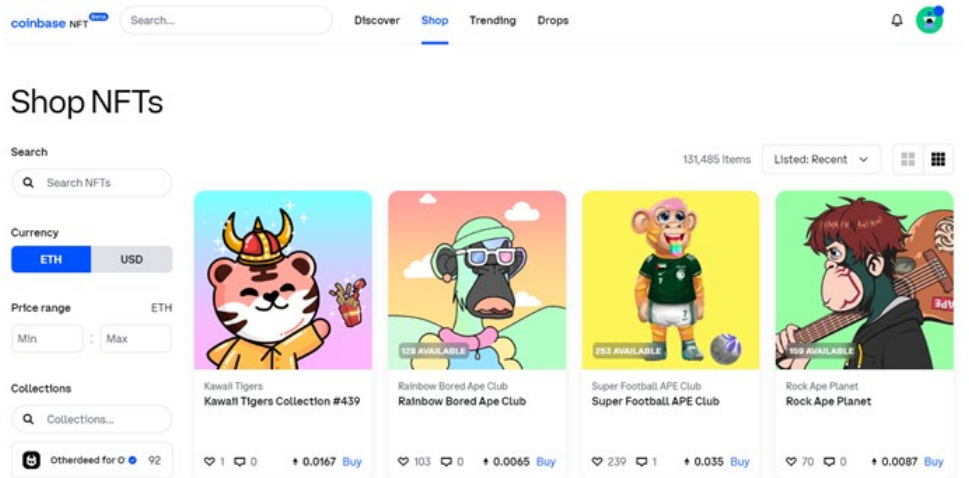
5. When connected to a network the user creates a username and password that is associated from the wallet signature.

EXHIBIT 3





6. The user is logged in as indicated at the top RHS icon.



iii.	*a second computer processor coupled to the second memory and*	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone.

EXHIBIT 3

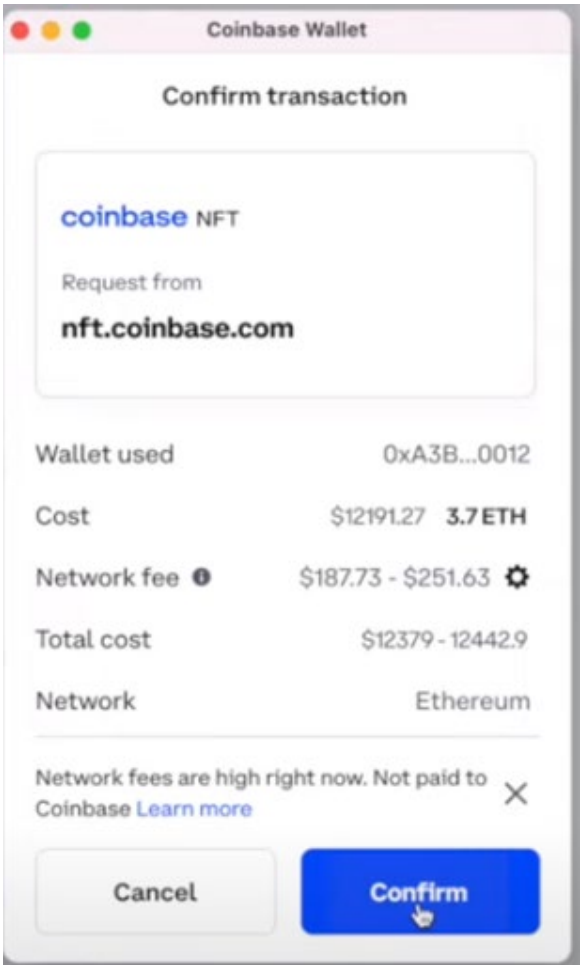
the second network interface, the second computer processor configured to:	
A. read the second private key from the second memory;	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via the Wallet private key either before using the App or for sign individual transactions. The private key is encrypted on the device and accessed for reading (and subsequently used to calculate a transaction signature) by the app/wallet from the passcode/protection mechanism of the app/wallet.  Coinbase Wallet requesting confirmation to access (read) the private key to authorise and sign the transaction.
B. read the inchoate data record	The published Inchoate Data Record is the offered NFT for sale on a marketplace. The following is an example from the Coinbase NFT Marketplace.

EXHIBIT 3

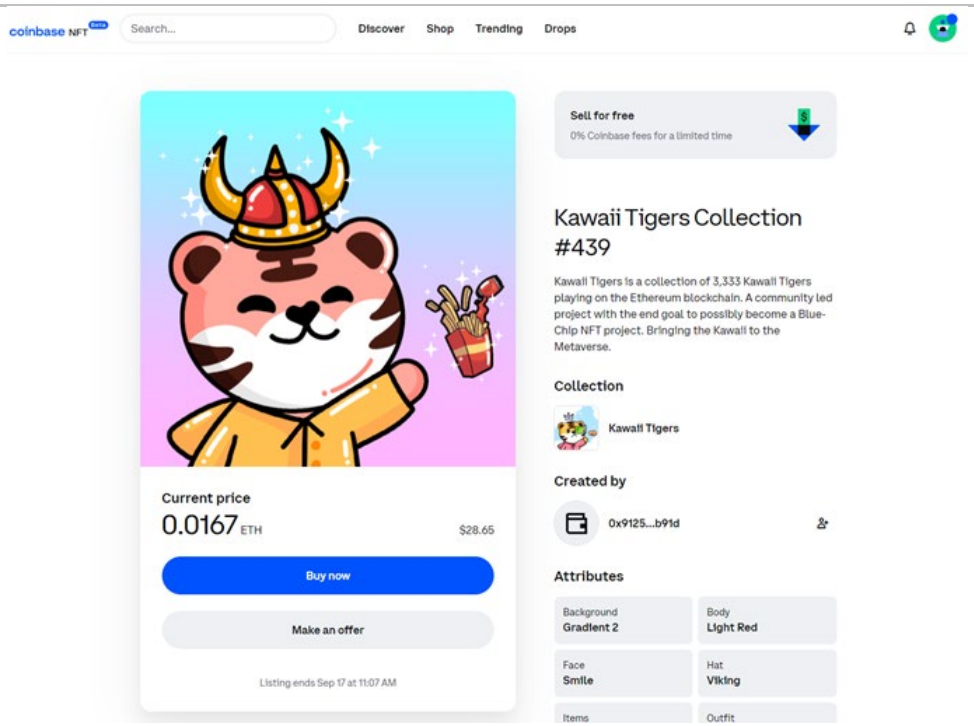
	 https://nft.coinbase.com/nft/ethereum/0x0a62be5f3552c4252f50edcd75de3aed831f9599/439
C. compute a second cryptographic signature from the second private key;	The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via a Wallet before using the App.

EXHIBIT 3

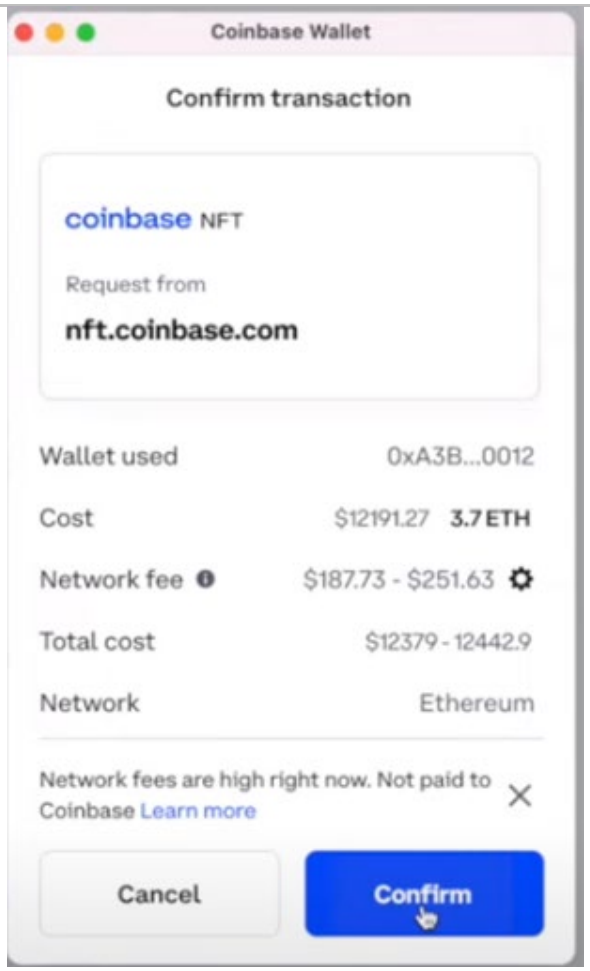
	 Coinbase Wallet requesting confirmation to access (read) the private key to authorise, calculate signature and sign the transaction. **eth_SignTransaction** web3.eth.signTransaction(transactionObject, address [, callback]) Signs a transaction. Parameters Object - The transaction data to sign. See web3.eth.sendTransaction() for more. String - Address to sign transaction with. Function - (optional) Optional callback, returns an error object as first parameter and the result as second. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#signtransaction
D. create a complete data record comprising: I. the commit input;	The App (Coinbase Wallet) initiates a smart contract call populating the data field of the above message. From: https://eips.ethereum.org/EIPS/eip-721 *ERC-721 standardizes a safe transfer function `safeTransferFrom` (overloaded with and without a `bytes` parameter) and an unsafe function `transferFrom`. Transfers may be initiated by:* • *The owner of an NFT*

EXHIBIT 3

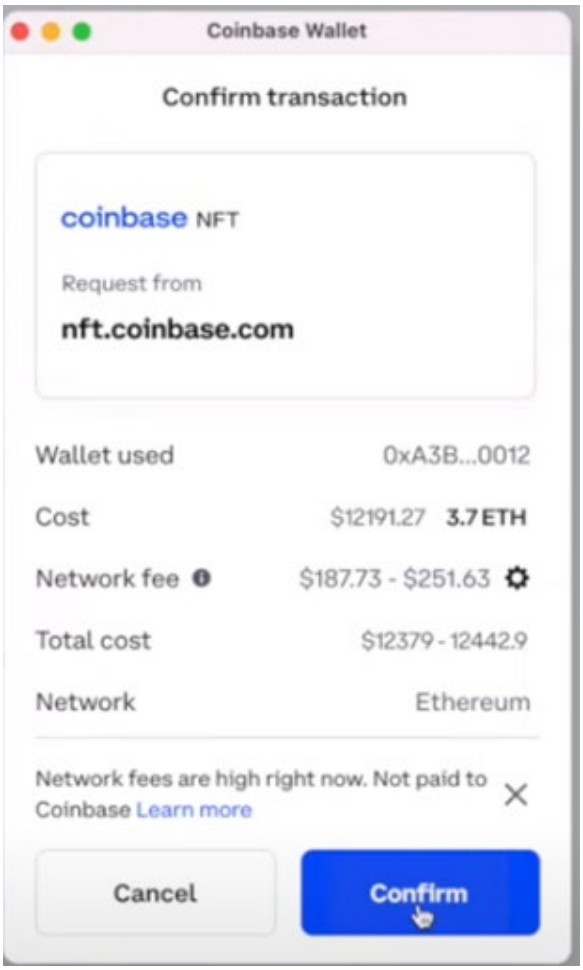
		• *The approved address of an NFT* • *An authorized operator of the current owner of an NFT*
II.	*the output data;*	The output data includes: • The signed acceptance of the offered NFT based on the value description of the Metadata URI link. • The disbursement amount to the seller and the originator based on the royalty fee definition of the NFT.
III.	*the first cryptographic signature, and*	The seller has signed the 'approve' smart contract function call to authorise transfer of ownership when the terms are met.
IV.	*the second cryptographic signature,*	The buyer signs the transaction. The hardware device, hosting the API software/libraries (used by an App), such as a computer or mobile phone. The App requires the user to authenticate themselves via a Wallet before using the App.  Coinbase Wallet requesting confirmation to access (read) the private key to authorise, calculate signature and sign the transaction. **eth_SignTransaction** web3.eth.signTransaction(transactionObject, address [, callback])

EXHIBIT 3

	Signs a transaction. Parameters Object - The transaction data to sign. See web3.eth.sendTransaction() for more. String - Address to sign transaction with. Function - (optional) Optional callback, returns an error object as first parameter and the result as second. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#signtransaction
E. create a transaction by submitting the complete data record to the transfer mechanism;	The NFT approved for sale is accepted by the buyer with the safeTransfer function call to transfer ownership to the buyer. The App (Coinbase Wallet) creates a transaction message to be sent to the Ethereum Network. RPC JSON API calls are defined to allow interaction with the Ethereum network. The App constructs the following message which is aligned with this standard. **eth_sendTransaction** (EIP1559) Creates new message call transaction or a contract creation, if the data field contains code. Object - The transaction object 1. from - String|Number: The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified. Or an address or index of a local wallet in web3.eth.accounts.wallet. 2. to - String: (optional) The destination address of the message, left undefined for a contract-creation transaction. 3. value - Number|String|BN|BigNumber: (optional) The value transferred for the transaction in wei, also the endowment if it's a contract-creation transaction. 4. gas - Number: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded). 5. gasPrice - Number|String|BN|BigNumber: (optional) The price of gas for this transaction in wei, defaults to web3.eth.gasPrice. 6. type - Number|String|BN|BigNumber: (optional) A positive unsigned 8-bit number between 0 and 0x7f that represents the type of the transaction. 7. maxFeePerGas - Number|String|BN: (optional, defaulted to (2 * block.baseFeePerGas) + maxPriorityFeePerGas) The maximum fee per gas that the transaction is willing to pay in total 8. maxPriorityFeePerGas - Number|String|BN (optional, defaulted to 1 Gwei) The maximum fee per gas to give miners to incentivize them to include the transaction (Priority fee) 9. accessList - List of hexstrings (optional) a list of addresses and storage keys that the transaction plans to access 10. data - String: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code. 11. nonce - Number: (optional) Integer of the nonce. This allows to overwrite your own pending transactions that use the same nonce. 12. chain - String: (optional) Defaults to mainnet. 13. hardfork - String: (optional) Defaults to london. Returns 4. DATA, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available. https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#sendtransaction

EXHIBIT 3

	The App (Coinbase Wallet) initiates a smart contract call populating the data field of the above message. From: https://eips.ethereum.org/EIPS/eip-721 *ERC-721 standardizes a safe transfer function `safeTransferFrom` (overloaded with and without a `bytes` parameter) and an unsafe function `transferFrom`. Transfers may be initiated by:* • *The owner of an NFT* • *The approved address of an NFT* • *An authorized operator of the current owner of an NFT*
7. c. the second client comprises: *i. a third memory for storing a third asymmetric key pair, the third asymmetric key pair comprising a third private key and a third public key;*	The **Computing Device | Facilitator** consists of: • the Coinbase (Owned, managed or offered) Ethereum Validator Full Nodes; and • the Coinbase (Owned, managed or offered) Ethereum supporting Archive Nodes and Light Nodes; and • the Coinbase (Ethereum compatible) wallets; Where both the Coinbase Nodes and the Coinbase Ethereum compatible end user wallet device are networked to have direct or indirect communication with each other. **Client Device** The First Client is the end user device that accepts an offer for an NFT. The Second Client is the end user device that authorises an NFT to be offered for sale. Three (3) Client instances have been identified that represent various implementations that exist. 1. Client is a device running an App (Coinbase Wallet) that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js. This device will also use a key storage wallet with EIP-191 or EIP-712 signing support (such as Metamask/Coinbase Wallet) to send an NFT transactions. The use of an EIP-1193 based Ethereum Provider API or EIP-191/EIP-712 signing support means the Client is considered to be part of the Ethereum Network. http://eips/ethereum.org/EIPS/eip-191 http://eips/ethereum.org/EIPS/eip-712 http://eips/ethereum.org/EIPS/eip-1193 2. The Computing Device and the Client are the same device where the Client is a Client Browser or Command Line Interface on a Coinbase Ethereum Node used to create, sign and submit NFT transactions to the Coinbase Ethereum Node for processing. 3. Client is a device running an App (Coinbase Wallet) that uses a Javascript Ethereum Provider API (as per EIP-1193) such as web3.js or ethers.js.. This device will also use a key storage wallet. This device uses private key management in order to sign transactions Where the Client running the App (Coinbase Wallet) software, as part of the Facilitator, need a computer hardware/software combination to run, namely: • Memory (RAM), used in the computing device (such as a computer or mobile phone). • Transaction record sector (stores transactions that haven't been submitted to the blockchain yet) kept via the crypto software wallets

EXHIBIT 3

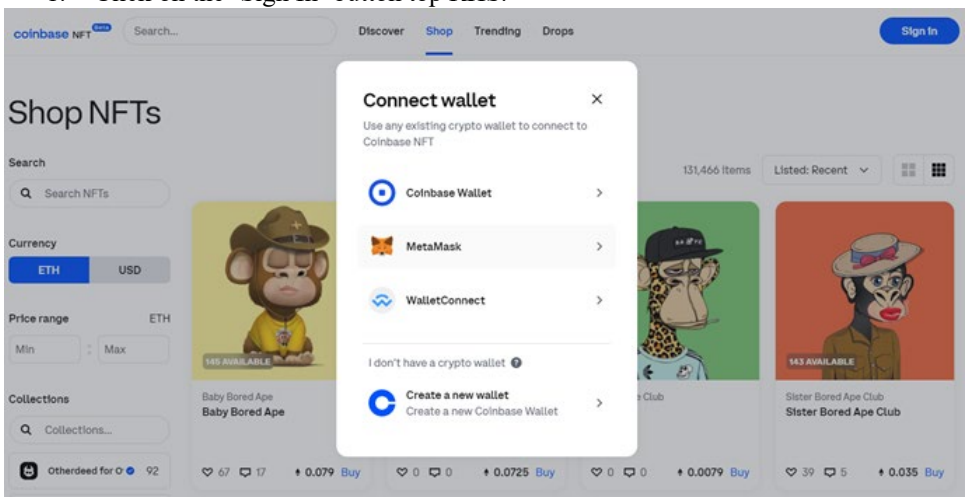
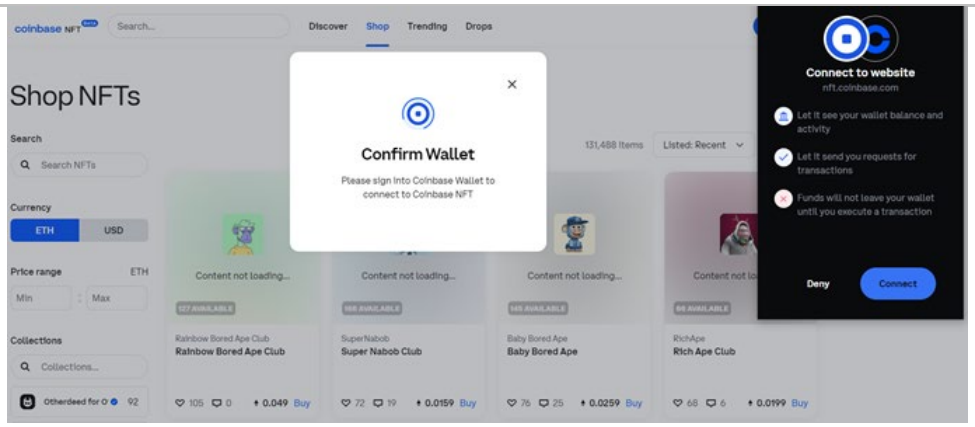
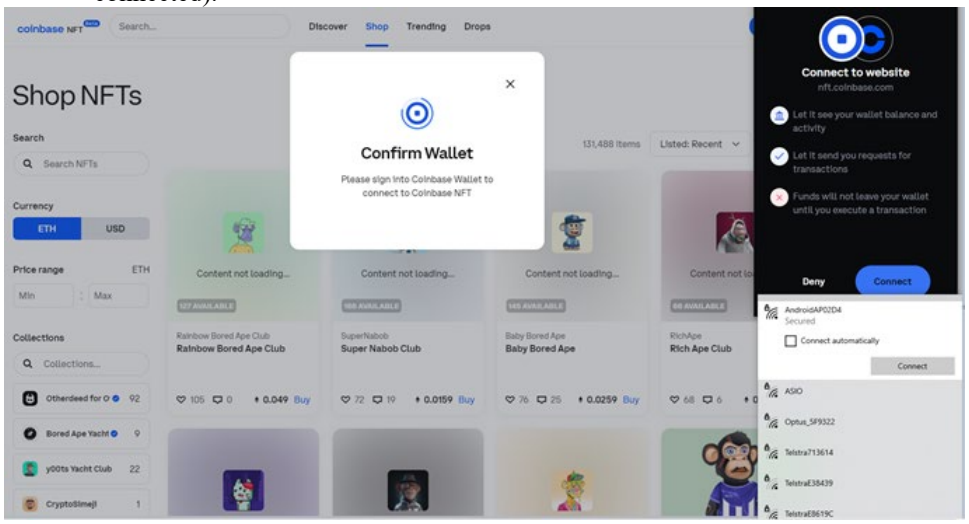
		Where the Client running the App (Coinbase Wallet) software, contains: • a first key pair sector which is generated and stored in the wallet software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the wallet software. • The wallet software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair. Where the Client interacts with the Eth Client Browser or Command Line Interface to transaction creation, contains: • a first key pair sector which is generated and stored on the device • The asymmetric key pair stored consists of a first private key and a first public key. Where the Client interacts with App (Coinbase Wallet) with Private key management in order to sign transactions, contains: • a first key pair sector which is stored in the key management vault/software • The asymmetric key pair generated and/or stored consists of a first private key and a first public key – all found and manipulated via the key management vault/software. • The key management vault/software connects via the public key or the key pair, and authorizes (signs) the transaction with the private key of the key pair.
ii.	*a third network interface; and*	The Client device requires a network interface in order to connect a wallet to the marketplace to offer the NFT for sale. 1. Click on the 'Sign In' button top RHS.  2. Selecting 'Coinbase Wallet' prompts the user to connect to the website.

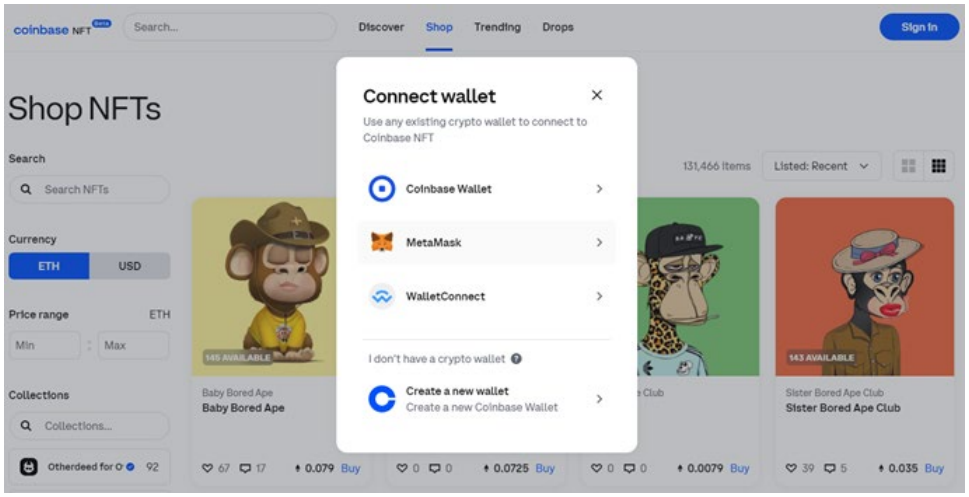
EXHIBIT 3



3. Attempting to connect without a network interface (as shown with no Wifi connected).

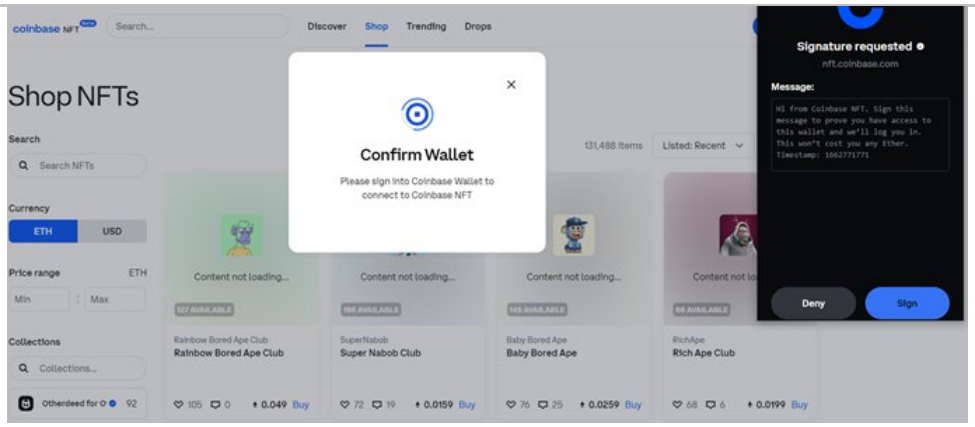


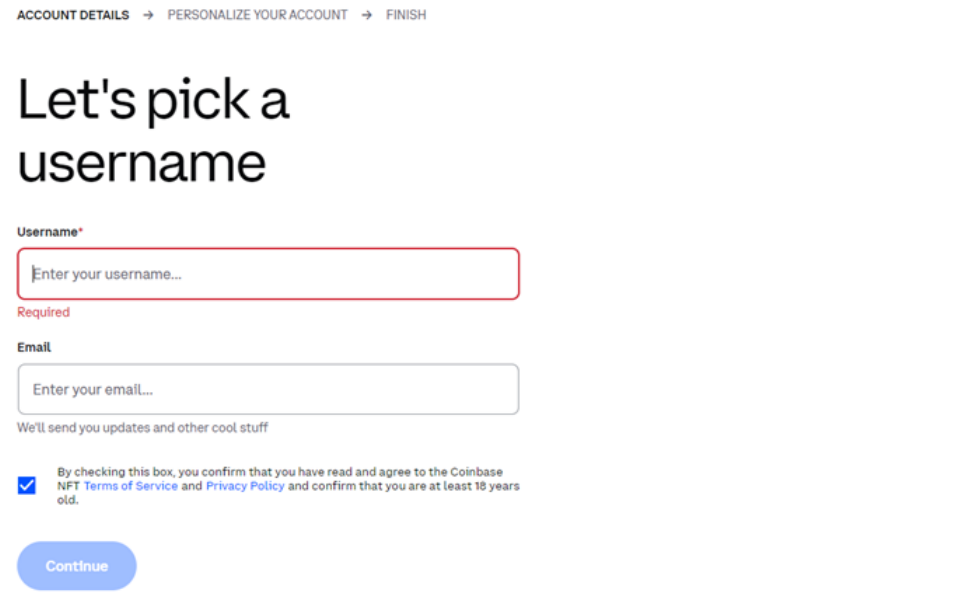
4. The user is not connected to the NFT marketplace and is instead requested to connect wallet.



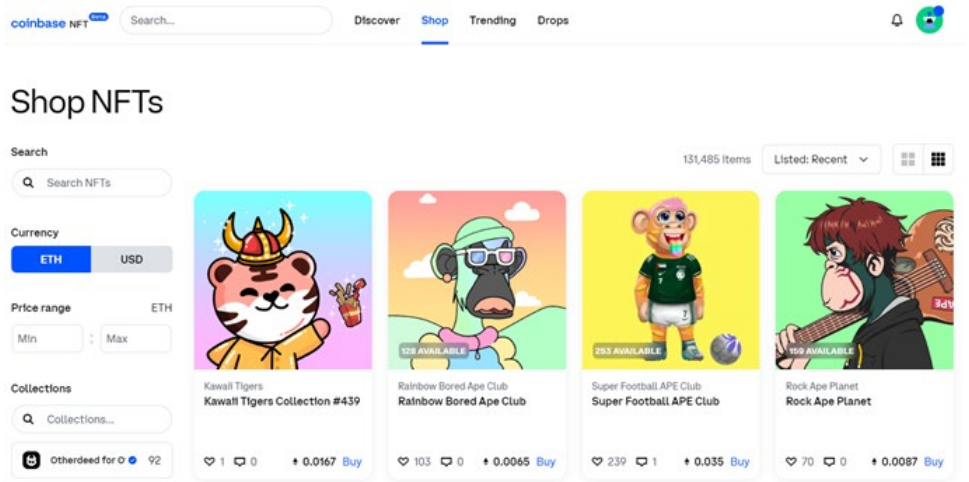
5. When connected to a network the user creates a username and password that is associated from the wallet signature.

EXHIBIT 3





6. The user is logged in as indicated at the top RHS icon.



iii.	*a third computer processor coupled to the third memory and the third*	The hardware device using the API software/libraries, such as a computer or mobile phone.

EXHIBIT 3

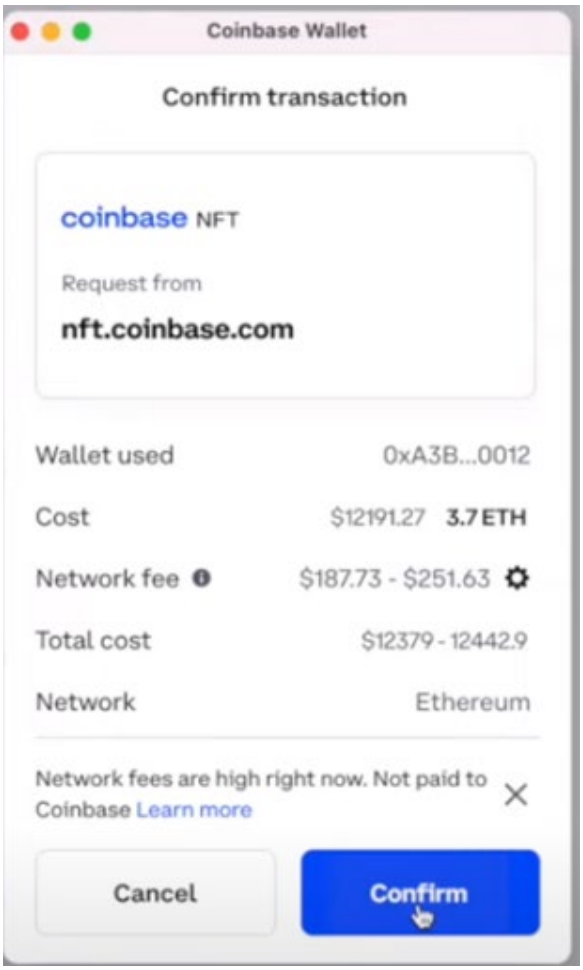
network interface, the third computer processor configured to read the third private key from the third memory; and	
wherein the at least one of the first client device or the second client device signs the inchoate data record and saves a copy of the inchoate data record on at least one of the first client device or the second client device.	The First Client requires the buyer to authenticate themselves via a Wallet before using the App. The wallet private key is used to sign the transaction for the EVM smart contract call.  Coinbase Wallet requesting confirmation to access the private key to authorise and sign the transaction. Note that it is not mandatory for the inchoate data record to be saved by the device as it is optional as described in the Patent general description. **[0095]** *11. The second client creates the complete commit transaction record by signing the inchoate commit transaction record and optionally saves a copy in non-transitory memory, the complete commit transaction record comprising:* **[0111]** *15. Optionally, the first client saves a copy of the complete commit transaction record in non-transitory memory.*

EXHIBIT 3

wherein the transfer mechanism comprising a decentralized digital currency that comprises a distributed ledger that enables processing the transaction between the first client device and the second client device without the need of a trusted central authority,	Ether is used as the decentralised currency. The Ethereum network maintains a distributed ledger without the need for a trusted central authority. From the Ethereum yellow paper. *2.1. Value. In order to incentivise computation within the network, there needs to be an agreed method for transmitting value. To address this issue, Ethereum has an intrinsic currency, Ether, known also as ETH and sometimes referred to by the Old English ¯D.* https://ethereum.github.io/yellowpaper/paper.pdf
wherein the transaction is created by broadcasting the complete data record for transmitting and receiving among network participants in the computer network for recording in the distributed ledger, and	The NFT approved for sale is accepted by the buyer with the safeTransfer function call to transfer ownership to the buyer. The App (Coinbase Wallet) creates a transaction message to be sent to the Ethereum Network. RPC JSON API calls are defined to allow interaction with the Ethereum network. The App constructs the following message which is aligned with this standard. **eth_sendTransaction** (EIP1559) Creates new message call transaction or a contract creation, if the data field contains code. Object - The transaction object 1. from - String|Number: The address for the sending account. Uses the web3.eth.defaultAccount property, if not specified. Or an address or index of a local wallet in web3.eth.accounts.wallet. 2. to - String: (optional) The destination address of the message, left undefined for a contract-creation transaction. 3. value - Number|String|BN|BigNumber: (optional) The value transferred for the transaction in wei, also the endowment if it's a contract-creation transaction. 4. gas - Number: (optional, default: To-Be-Determined) The amount of gas to use for the transaction (unused gas is refunded). 5. gasPrice - Number|String|BN|BigNumber: (optional) The price of gas for this transaction in wei, defaults to web3.eth.gasPrice. 6. type - Number|String|BN|BigNumber: (optional) A positive unsigned 8-bit number between 0 and 0x7f that represents the type of the transaction. 7. maxFeePerGas - Number|String|BN: (optional, defaulted to (2 * block.baseFeePerGas) + maxPriorityFeePerGas) The maximum fee per gas that the transaction is willing to pay in total 8. maxPriorityFeePerGas - Number|String|BN (optional, defaulted to 1 Gwei) The maximum fee per gas to give miners to incentivize them to include the transaction (Priority fee) 9. accessList - List of hexstrings (optional) a list of addresses and storage keys that the transaction plans to access 10. data - String: (optional) Either a ABI byte string containing the data of the function call on a contract, or in the case of a contract-creation transaction the initialisation code. 11. nonce - Number: (optional) Integer of the nonce. This allows to overwrite your own pending transactions that use the same nonce. 12. chain - String: (optional) Defaults to mainnet. 13. hardfork - String: (optional) Defaults to london. Returns 5. DATA, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available.

EXHIBIT 3

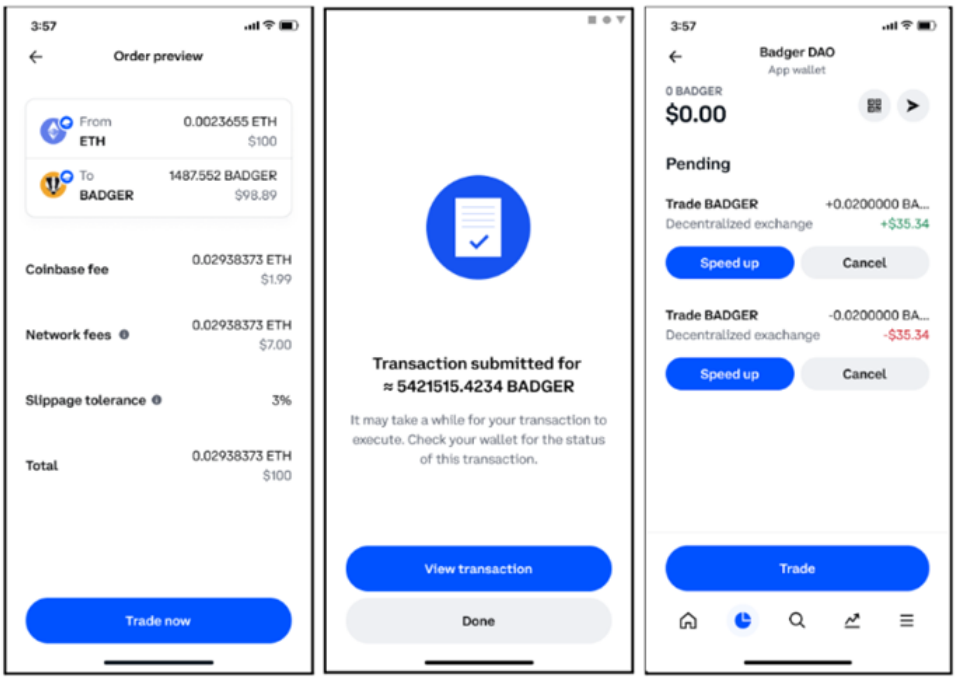
	https://web3js.readthedocs.io/en/v1.5.2/web3-eth.html#sendtransaction The App (Second Client as part of the Facilitator) initiates a smart contract call populating the data field of the above message. From: https://eips.ethereum.org/EIPS/eip-721 *ERC-721 standardizes a safe transfer function `safeTransferFrom` (overloaded with and without a `bytes` parameter) and an unsafe function `transferFrom`. Transfers may be initiated by:* • *The owner of an NFT* • *The approved address of an NFT* • *An authorized operator of the current owner of an NFT*
wherein at least one of the computer device, the first client device, or the second client device verifies the recording of the complete data record in the distributed ledger by observing an external state	The First Client verifies that the Complete Data Record (Transaction) has been recorded by the network as a complete transaction record. In the below image this is shown in the 'History' tab with the transaction shown as confirmed.  https://help.coinbase.com/en/coinbase/trading-and-funding/trade-on-dex/check-transaction-status When a transaction is sent, a transaction hash is received. This can be used to retrieve transaction details. Use the JSON RPC API command getTransactionByHash({transaction hash}) to retrieve the transaction details. The returned blockNumber should be non-null if the transaction has been mined and included into a block. Then call JSON RPC API eth_blockNumber to get the current block height. The number of confirmations is the eth_blockNumber result minus the eth_getTransaction blockNumber result. https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_blocknumber https://github.com/ethereum/wiki/wiki/JSON-RPC#getTransactionByHash

EXHIBIT 3