Designing RGB Smart Contracts

with single-use-seals & client-side validation paradigms

LNP/BP Standards Association
Prepared & supervised by **Dr Maxim Orlovsky**, **Pandora Core AG**
Created with support from **Bitfinex**, **Fulgur Ventures** and **Poseidon Group**

Samples of RGB Smart Contracts

- Each asset (fungible or non-fungible) issued by somebody is a separate smart contract
- Each root identity (like “master identity key”) is a separate smart contract
- Each case of provable audit log on RGB (like a history of disease for a single patient at a hospital) is a separate smart contract

RGB Smart Contract consists of

- Single smart contract **Genesis** node
  - Created by issuer
  - Committing to Schema
  - No commitments in bitcoin transaction graph
- Branching tree of **State transition** nodes
  - Created by owners during state ownership transfers
  - Always committed into transaction graph
  - Linked to each other (up to genesis) with single-use-seals
- **Simplicity scripts**, taken from Schema, Genesis and direct upstream line of transitions
  - Extend each other according to parent node rules
    (i.e. Genesis can add scripts only when allowed by Schema, state transition – only when allowed by Schema AND Genesis AND all previous state transitions)

Schema

- Shared by many contracts, i.e. sort of a “**contract type**”
- Defines rules for client-side validation of smart contract nodes (i.e. genesis and state transition) at both per-node level and as an upstream DAG
- Wallets, exchanges, payment providers etc integrate RGB schemata, not particular smart contracts (for instance, they integrate Fungible Assets Schemas, not USDT or particular asset)
- Immutable for eternity by social consensus
- Not committed to bitcoin blockchain (b/c no reason to do so)
- Smart contract is created under certain schema when it includes a hash of the corresponding schema data+structure+scripts

Initial list of Schemas

- Can be vendor-defined
- To have a broad support by software, most common must be standardized

Subject	Schema name	LNP/BP Standard	Analog to
Fungible assets	RGB-20	LNPBP-20	ERC-20
Collectibles	RGB-21	LNPBP-21	ERC-721
Reputation/indentity	RGB-22	LNPBP-22	n/a
Audit log	RGB-23	LNPBP-23	n/a

Smart contracts

	“Ethereum-style”	RGB
• **Parties of the agreement**	loosely defined	issuer and current owners
• **Agreement:**	Bockchain-stored contract + ABI file	Client-stored contract genesis + state transitions
- **Current state**	blockchain-stored data: ★ publicly visible ★ non-confidential ★ non scalable ★ no 2nd layer support	client-stored data: ★ no chain analysis ★ confidential ★ scalable ★ 2nd layer support
- **State change rules**	custom EVM code	schema & simplicity script
- **Ownership rights**		bitcoin script
• **Mutability**	Pseudo-immutable: immutable in promice, censored my miners & creators in fact	Well-defined mutability rights at genesis & schema level by issuer Mutable by new owners within the scope of rules

1. There always must be an owner

- Smart contract state is not a “public good”
  (Ethereum/“blockchain” approach);
  it must always have a well-defined ownership (private, multisig…).
- RGB defines ownership by binding/assigning state to Bitcoin transaction outputs with single-use seals: whoever controls the output owns the associated state
- I.e. RGB leverages Bitcoin script security model and all its technologies (Schnorr/Taproot etc).

2. State ownership != state validation

- Ownership defines WHO can change the state
- Validation rules (client-side validation) define HOW it may change

2. State ownership != state validation

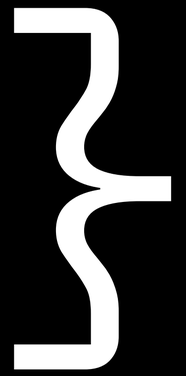
- Ownership controlled by Bitcoin script, at Bitcoin blockchain level (non-Turing complete)
- Validation rules controlled by RGB Schema with Simplicity script (Turing-complete)

This allows to avoid mistake done by “blockchain smart contracts” (Ethereum/EOS/Polkadot etc): mixing of layers & Turing completeness into non-scalable blockchain layer

Also it makes possible for smart contracts to operate on top of Layer 2 solutions (Lightning Network)

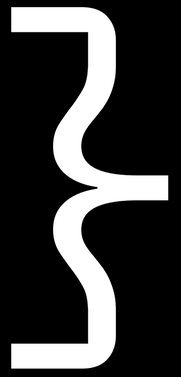
RGB Smart Contracts:

- Bitcoin script: bare, hashed and Taproot
  - Multisigs, state channels, swaps…
- Scriptless scripts: private, less footprint
- RGB
  - Using schema & simplicity language
  - No blockchain footprint
  - Confidential
  - Nearly fully Turing-complete



ownership:

single-use-seals



state validation

client-side validation

Thus, RGB smart contract is:

- Distributed system
- Where nobody has the complete view of the current state
- But it is still globally consistent (has consensus) because of:
  - Single-use seals based on bitcoin PoW
    (with possible LN as an intermediary)
  - Social consensus on the same client-side validation rules (Schema)
- Only owners has access to their owned state + a slice of state history DAG directly related to the owned state

Rights management under RGB smart contract

- RGB rights:
  Smart-contract defined types of actions, which can be taken only by a party owning some part of the smart contract state.
  - Ownership of the assets
  - Ownership of the identity
  - Right to inflate asset supply
  - Right to create child identities
  - Right to prune/prune assets
  - …

Rights management under RGB smart contract

- Types of allowed rights are **defined** at Schema.
  They named *state types*
  (b/c each right must have some current state/value, even if this is an “empty value”)
- Initial rights are **assigned** by the contract issuer in Genesis
- Rights can be **transferred** (together with the new state value) to new owners with *state transitions*
- Rights (state) **ownership** is controlled with bitcoin scripts via *single-use-seal mechanism*
- Who will be the next owner for particular right is always defined by the party that currently have that right (i.e. *state owner*)
- *Client-side validation*: rights transfers MUST be always **validated** backwards by the new owners according to validation rules.

Right transfer/state transition validation rules

- **Defined by Schema** using two main instruments:
  - Schema structure (how rights can be decided among descendants)
  - Simplicity scripts (how the state of some rights may evolve).
    For instance, for assets script requires that a sum of outputs must be equal to a sum of outputs.
- Can be further **restricted** (and not extended) at the level of **genesis** and each state transitions
- **Validated by new owners** backwards up to the genesis within a particular subgraph
- Violation of the rights in one smart contract ownership branch does not affects smart contract integrity in other branch

Security measures

- Each right (i.e. state) does not have a direct access to the information on the state under other rights
- If required, rights can have a “shared state” using metadata; Schema and Genesis explicitly defines whether this is allowed
- State can be “hidden” (made confidential) with zero knowledge; which state MUST be hidden is defined by the Schema

Schema defines

- Types of metadata and their value restrictions
  (like max length for strings; max value for integers etc)
- Types of rights (i.e. state) and their value restrictions
- How rights transfers (state transitions) can be organized:
  - which metadata they must provide
  - which rights can be (or must be) transferred jointly
  - history validation rules for each of the rights, defined using Simplicity
    (like “sum of outputs must be equal to the sum of inputs)
- How these rules can be further limited - or extended - at the level of genesis and individual state transitions

Fungible Assets Schema

RGB-20 standard

Genesis metadata: asset definition

	Type & restrictions	Required	Notes
Ticker	String < 16 chars	Yes	
Name	String < 256 chars	Yes	
Description	String < 1024 chars	No	Arbitrary data
URL	*Removed since absent in Confidential Assets*		
Signature	*Must be part of the above layers*		
Precision	Integer, 0-18	Yes	Number of digits after the dot
Dust limit	Integer, 0-2^64	No	In smallest units (satoshi-like)
Issued supply	Integer, 0-2^64	Yes	Required b/c we use confidential amounts
Blinding factor	Integer, 0-2^64	Yes	
Total supply	Integer, 0-2^64	No	Valid only when inflation is allowed
Timestamp	32-bit integer	Yes	Unix timestamp

Questions

- We need to double-check that this is CA compliant
- What is the reasonable upper bound for “Dust limit” value?
- Do we need introduction of the timestamp?

State types (possible rights)

	State data	Genesis defines
Asset ownership	Confidential amount	One or more
Inflation *(secondary issues)*	No state data	None: inflation is not allowed
		One: inflation can be done by a single well-defined party controlling specific bitcoin UTXO
Burning / prunning	No state data	None: burning/prunning is not allowed
		Many: prunning can be done by a multiple parties with different combinations

Open questions

- Do we need to restrict inflation rights to only a single UTXO?
- Do we need the ability to prohibit Burning/prunning?
- Do we need to restrict burning rights to only a single UTXO?

State transition types

	Metadata	Closes seals	New state seals	Notes
Asset transfer	–	• Multiple asset ownership	• Multiple asset ownership	
Inflation *(secondary issuances)*	• Issued supply • Blinding factor	• Single issue right seal	• Multiple asset ownership • None inflation seals	Further inflation is not allowed
			• Multiple asset ownership • One inflation seal	Inflation is possible until “total supply” value from genesis is not exceeded
Burning / prunning	• Proof type? • Proof data?	• Single prunning right seal • Multiple asset issuance seals	• Multiple asset ownership • None prunning seals	From now on prunning/ burning is prohibited
			• Multiple asset ownership • One or many prunning seals	Many: prunning can be done by a multiple parties with different combinations

Why we can’t join prunning & inflation

- Information on the total amount of assets in circulation will be completely lost
- No way to limit the total supply without denying pruning procedure at the same time
- No chances to provide proof data

Open questions

- Do we need allow removal of Prunning rights at all?
- Do we need to restrict burning rights to only a single UTXO?
- Do we need to add proofs metadata to the prunning transition?

Burning / prunning options

1. Disallow procedure for all RGB-20 fungible assets
   (remove from Schema)

2. Leave as is

3. Add ability to add custom (of many possible types) proofs
   Allow asset issuers to mark them required in Genesis

4. Add epochs mechanism with eventual validation

5. Combination of 3 & 4

What are the inflation epochs?

- Each fact of asset issue (primary issue with genesis, secondary issues with inflation transitions) defines an “epoch seal”
- Issuer may prune each asset only once during the epoch
- After prunning of some% of all of the assets under some issue, the issuer has the right to open the new epoch
- When all assets are pruned, the epoch is closed and everybody can witness that no actual inflation were created
- Even before epoch is closed it is possible to everybody to do probabilistic estimation of the hidden inflation risk

Burning / prunning options

	Pros	Cons
Disallow	• “Simple solution” • Best fungibility	• No prunning in RGB-20
Leave as is	• Still “simple solution”	• Issuers may cheat and hide the inflation • Lowers fungibility
Add custom proofs	• Issuers can innovate on proof mechanisms	• Software vendors must support each type of proof validation, which may affect fungibility
Add epochs	• Good fungibility • Vendors do not need to update software for new type of proofs	• A lot of complexity
Add custom proofs AND epochs	• Good fungibility • Issuers can innovate on proof mechanisms	• Software vendors must support each type of proof validation, which may affect fungibility • A lot of complexity

My proposal

- Use proof epochs only