RGB - Bitcoin L2

#BitcoinL2

陣営、という表現をしたがそう単純なものではないだろう。

RGB Consortium陣営



- https://consortium.rgb.tech/
  - https://github.com/LNP-BP
  - https://www.lnp-bp.org/
    RGBを含むClient Side Validation技術やライブラリの標準化・開発を統括するスイスのNPO
- https://github.com/RGB-WG
  - https://rgb.tech/
    関係ないがRustのサイトと同じような見た目
  - https://www.rgbfaq.com/
  - XのRGB Consortium
  - XのMaxim Orlovsky

RGB Protocol Association陣営



- https://rgbprotocol.org/
  名前は「RGB Protocol Association」だがタイトルが「RGB Association」になっていてどっちが本名かよくわからない



- https://github.com/rgb-protocol
  - https://rgb.info/

XのRGB Hub

https://github.com/RGB-Tools

Iris Wallet で有名

その他

RGB - ChatGPT

RGB - ChatGPT2

RGB - ChatGPT3

RGB - Grok

その他

/nayuta/ご）RGB

RGBプロトコルのググりかた(2022年) - ドクセル

RGB Protocolとは何か - Recruit Tech Blog(2021年)

v0.12-rc3でregtestデモスクリプトを作った

PlanB Networkにあった記載の和訳を載せよう。

> RGBの目標は、ビットコイン上でスマートコントラクトを実装するためのインフラストラクチャを提供することです。「スマートコントラクト」とは、複数の当事者間で締結され、条項の執行に人手を介さずに、自動的に計算によって執行される契約を指します。言い換えれば、契約法は信頼できる第三者ではなく、ソフトウェアによって執行されるということです。

Taro がトークンに特化し、RGB はスマートコントラクトが使えるということか。
RGB は「Schema」という、アセットの挙動を規定するスマートコントラクトだそうだ。

Lightning Network での送受金も可能だし、ルーティングも使用できる。
ただオリジナルの LN に対して改造が必要だし、ルーティングも改造した LN である必要がある。
使用できるトークンも同じになっていないといけない。

最近の情報

RGB smart contracts(2025年7月18日)

RGBチームからforkしているプロジェクトが公式リリースっぽいv0.11.1というのを出しているが、それは不十分で品質も良くないので、forkするプロジェクトを否定するわけではないが forkされたプロジェクトまで管理していないので注意すること。

おそらくBitfinex

https://x.com/rgbconsortium/status/1946181777921286580

https://x.com/BitlightLabs/status/1946189619852173478

RGB smart contracts(2025年7月10日)

v0.12.0-rc2が2025年6月3日に出ている

その前はv0.11 beta 8で2024年9月6日。betaは取れない？

その前はv0.10 (stable)で2023年9月6日。

同じルールなら、v0.12 は2025年9月6日か？

zkSTARK圧縮のためのプロトコル準備

v0.10やv0.11では過去と非互換になる破壊的な変更が行われていたらしい

プロトコルの簡素化(今まで評判が悪かったらしい)

テスト工程の見直しによるセキュリティ向上

コンセンサスレベルの変更を凍結

Lnfi Network Integrates RGB Protocol To Enhance Lightning Network(2025年6月27日)

Lnfi Upgrade Completes Lightning Network's Value Stack with Full RGB Integration - by Lnfi Network - Jun, 2025 - Medium

Lnfi NetworkがRGBプロトコルを統合し、Bitcoin上でスケーラブルかつプログラマブルな資産の取引を可能にした。

Nostrを活用しているそうだ

LnfiはBitfinexが関係する会社のようで、上に書いたv0.11.1に関連しているらしい

https://atlas21.com/rgb-protocol-is-now-live-asset-tokenization-arrives-on-bitcoin-and-lightning-network/

Opinion: A Silent War is Unfolding on Bitcoin RGB - 吴说区块链 on Binance Square(2025年7月18日)

LNP/BP association とは別に RGB Protocol associationをBitfinexグループで構成するらしい(有料会員)

🔥 今までのRGB 側が v0.12 を出す準備のアナウンスをしたところで Bitfinex 側の RGB が v0.11.1 をメインネットリリースしたというアナウンスを出した。そのバージョンは Bitfinex側の開発チームでやっているので今までのチームのレビューを受けていない、というのが上の方に書いた7月18日の件だ。

今までの RGB側のコア開発者は Dr. Maxim Orlovsky 氏

RGBプロトコル、トークン化された資産USDTをビットコインに導入

First RGB Bridge Brings USDT From Ethereum To Bitcoin Via Lightning

✅ プロジェクト比較表

プロジェクト名	主な用途・特徴	現状
LNFi Network	Lightning統合型RGB資産の発行・取引	実環境稼働中
DIBA / Bitmask	NFT取引＋ウォレット	試験運用／市場提供間近
Bitlight Labs	BitcoinFiアプリ＋ウォレット開発	開発中
Infinitas	高速／高プライバシー型Bitcoinアプリ構築	初期展開フェーズ
Pandora Prime	GUIウォレット＆エクスプローラ	v0.12以降対応済み
RGB++ / Fiber	RGB拡張互換＋CKB連携、DEX基盤構築	CKB側で本格展開中

似たプロジェクトの現状

Counterparty

2024年初旬に開発が停止したがコア開発者が戻ってきて GitHubへの commitは行われるようになった(wikipedia)

OmniLayer

2025年のGitHubへのコミットはなさそう

LayerZeroベースのOmnichainに移行したり

概要

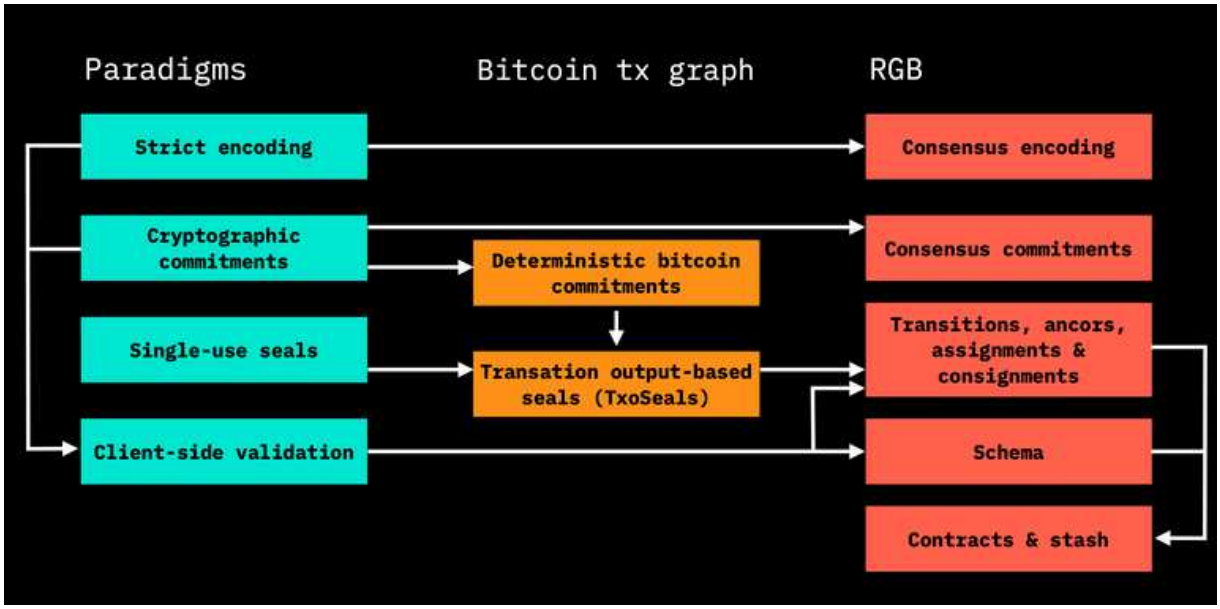
RGBは、BitcoinおよびLightning Network上で動作するスケーラブルかつプライベートなスマートコントラクトおよび資産発行プロトコルです。

Lightning Networkは必須ではないらしい。LNを使った場合は即時性がある。

Prime という RGB専用の L1というものも構想にあるらしい(Yellow Paper)

RGB のアーキテクチャパラダイムがあって、これをBitcoinに当てはめたのが RGBらしい。が、見ても分からん。

https://www.rgbfaq.com/rgb-paradigms#how-rgb-paradigms-are-interrelated



パラダイムはこちらの方がわかりやすいか。

分散コンピューティングの概念というところから説明している。

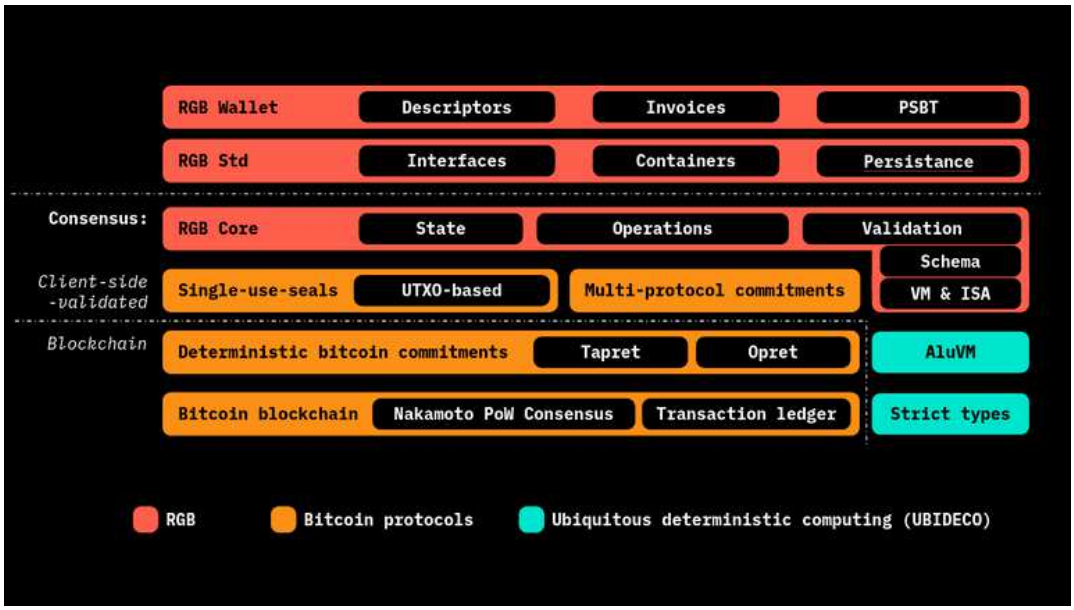
RGB-Documentation/distributed-computing-concepts/paradigms-of-distributed-computing.md at master · rgb-protocol/RGB-Documentation

(PlanBのRGBコースも内容は同じような感じだが)

RGBというか Client Side Validation はCAP定理でいうところの Consistency(一貫性)と Partition tolerance(分断耐性)を持つ(可用性Availabilityは持たない)

「シャード」というのはコントラクトの自分の部分(グローバルで全体を持つことに対して)で、RGBのスマコンは「各コントラクト」という感じでグローバルに持たない。

https://docs.rgb.info/annexes/rgb-library-map



「クライアントサイドバリデーション(Client Side Validation。CSV)」と「シングルユース

シール(Single-use Seal)」が特徴

クライアントサイドバリデーション

スマコンの状態や履歴はオフチェーンで扱う(グローバルでは持たない)。

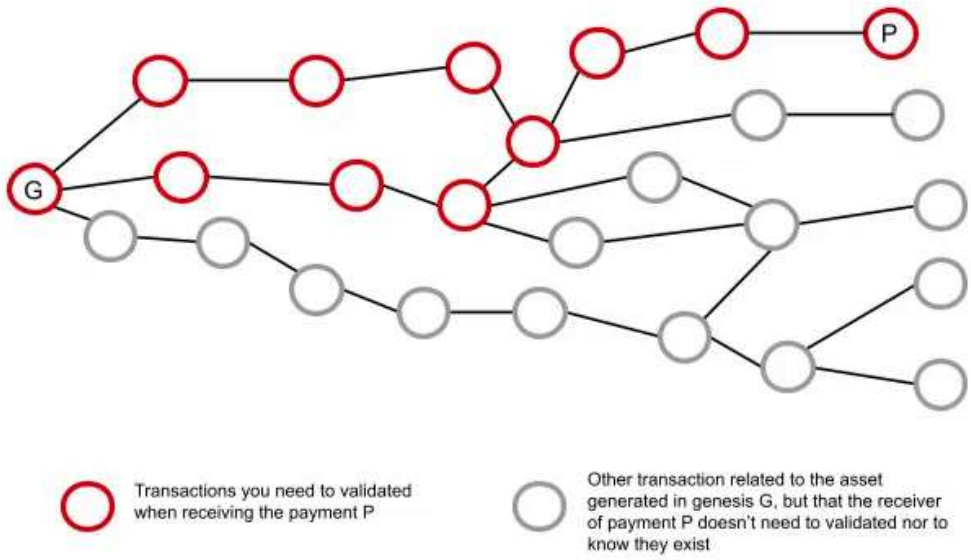
オンチェーンに書き込まないのでプライバシーが高い。

各人が持っている状態が正しいとか間違いとかを検証するのがクライアントサイドバリデーション。

検証の方式がCSV型というだけで、CSV型のやり方というものは無い？

「RGBは Single-use Seals 技術によって CSV型を実現している」などと呼ぶのだろうか

Genesis から受信者までの状態遷移＝トランザクションの履歴を検証すればよい(medium)



検証した状態遷移は集約されてLayer1に書き込む。二重支払いの防止に使う。

検証したデータのサブセットを stash と呼ぶ

RGBスマートコントラクトの整合性と履歴を維持するために参加者が必ず保持しなければならないクライアント側データのセット

stash は L1 と State Channel の両方で動作できる

stash はオンチェーンに載せないので各自でしっかり保存しておかねばならない

送金するときに相手に渡すデータや、スマコンのデータを cosignment と呼ぶ

スマコンのデータ: Contract Cosignements

送金のデータ: Transfer Cosignments

シングルユースシール (使い捨てシール)

RGBトークンの二重支払いを防止するために採用している方式

メッセージ `m` の中身を見せずに、メッセージ `h(m)` に関する特定のコミットメントが公開されていることを証明したい

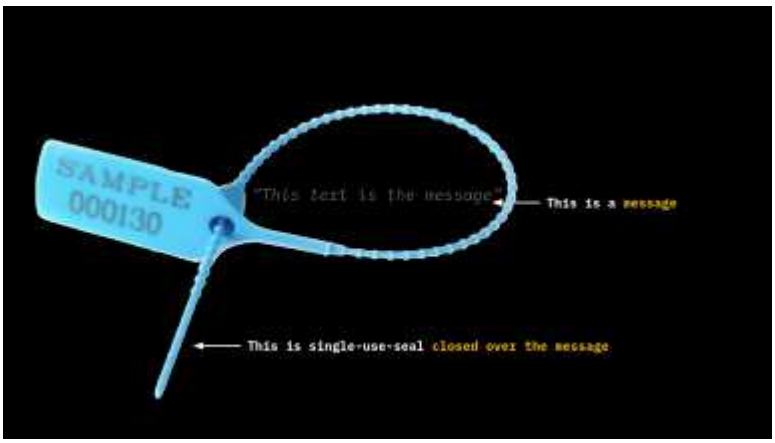
`h(m)` の代わりに競合する `h'(m)` のコミットメントが公開されていないことを証明したい

メッセージ `m` が特定の日付より前に存在するかどうかも確認したい

Single-use Seals - ChatGPT：概念のような記載なので RGBでの具体的な例は載っていない

そもそもの「シングルユースシール」という物理的なものについて

こんなやつらしい。タグみたいなもの？　このシールは「閉じる」という使い方をして、一度閉じたシールは再度開けたり閉めたりできないそうだ。開けたら無効になる。



一度剥がしたら跡が残るセキュリティシールみたいなもののようにも思えるが、これはどっちかといえば中身が見える封印みたいなものか？

Bitcoinの場合、シールの定義とシールの閉鎖の組み合わせがある。RGB は `TxO2` というスキーマを使う

シールの定義：outPoint

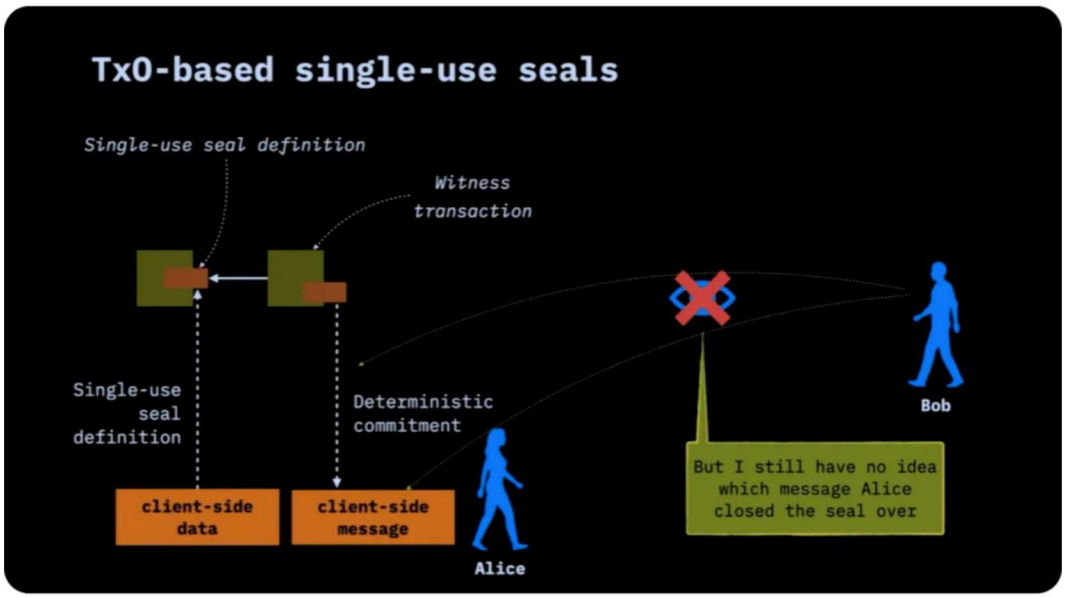
シールの閉鎖：シール定義した outPoint を消費するトランザクションで、メッセージにコミットメントが追加される

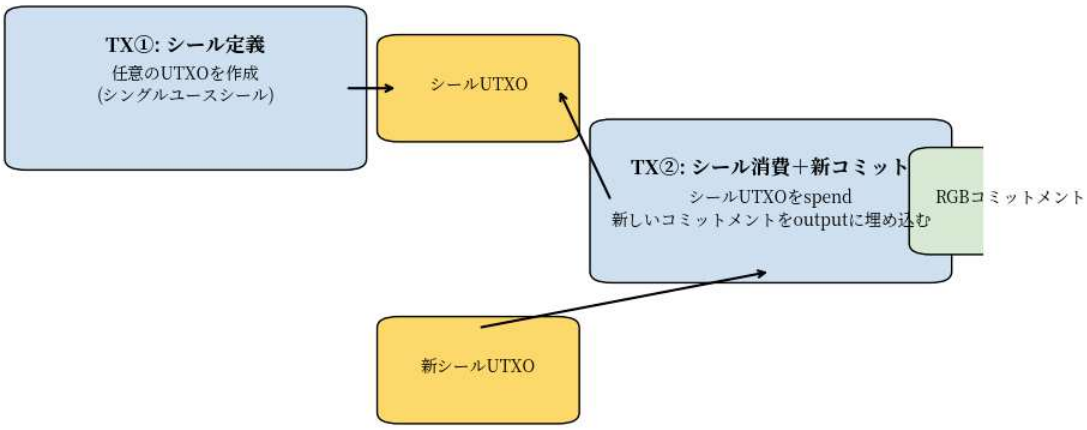
- Define the seal via a public key, and close it in an *output*;
- Define the seal with an *outpoint* and close it with an *output*;
- Define the seal via the value of a public key, and close it in a *input*;
- Define the seal via an *outpoint*, and close it in an *input*.

Schema Name	Seal Definition	Seal Closure	Additional Requirements	Main Application	Possible Commitment Schemes
PkO	Public Key Value	Transaction Output	P2(W)PKH	None at the moment	Keytweak, taptweak, opret
TxO2	Transaction Output	Transaction Output	Requires deterministic commitments on Bitcoin	RGBv1 (universal)	Keytweak, tapret, opret
PkI	Public Key Value	Transaction Input	Taproot only & not compatible with legacy wallets	Bitcoin-based identities	Sigtweak, witweak
TxO1	Transaction Output	Transaction Input	Taproot only & not compatible with legacy wallets	None at the moment	Sigtweak, witweak

シール定義は「この outPointを使う(この outPoint を私のシールにする)」と決めるだけで良い。UTXO のままだとシールは「オープン」な状態である。
シールを閉じる日にその outPoint からコミットメントを含んだトランザクションを展開する(Witness Transactionという名称だが segwit の witness とは関係ない)。

Witness Transaction が展開されると「何かイベントが起きた」ことになる
Witness Transaction を見ても、コミットメントは `OP_RETURN` や Tapscript に入っているだけなので知らない人は見ても分からない。オフチェーンでデータを送ってもらう。





Witness Transaction の outPoint が次のシール定義になる

TxO2 schemaの場合、コミットメントは「Keytweak(公開鍵)、Tapret (Tapscript)、Opret(`OP_RETURN`)」のいずれかに置かれる。

Opret の `OP_RETURN` 方式は検閲しやすいのであんまりなあという感じのようだ。Tapret の Scriptpath 方式を推している。(まだデータを見ていないが、コミットメントはデータだけなので redeem で公開しなくても良いはず。となると Merkle Tree に出てくるだけなのか？)

Tapret と Opret はわかるが、Keytweak の「鍵を調整する」はよくわからん

ときどき出てくる "MPC" は Multi Protocol Commitment の略

機能(NotebookLM)

close

シール `u` をメッセージ `m` に対してcloseすると証明 `w` が生成される

`m` は状態遷移をコミットするためのハッシュ値

オフチェーンで管理している現在の状態(アセットの所有者、数量、関連するトランザクション履歴など))をハッシュ値にする(`OP_RETURN` に埋め込むらしい)

シール定義で指定された元のコミットメントが履行される

❓わからん

UTXO が spent されると close になる

UTXO をシールと見立てる

UTXO が spent すると UTXO に戻ることはないので、これが Single-use(使い捨て)になる。

verify

証明 `w` が与えられたとき、シール `u` がメッセージ `m` に対してcloseされたことを検証する

❓こっちもわからん

とりあえず 1つ分のシールが検証できたとして、さらに遡ってその前の転送も

正しかったことを検証する。それを繰り返してアセットの発行元までたどる。
検証で使うオフチェーンのデータは送信者から直接P2Pで共有したり IPFSなどのリポジトリで共有したりする。

Iris Wallet は P2Pらしい

プロキシサーバーや Store and Forward という方式で共有することもあるらしい(Gemini情報)。ただ Store and Forward 方式の情報元は wikipedia だったのであやしい。

共有が P2P だった場合、最後に送受金したユーザだけが全体の履歴を持っていることになる。

ざっくり

やりたいこと：時系列で変化する情報を改ざんや検閲を避けつつL1に書込みたい
記録したい情報のハッシュ値と、UTXO の OutPoint をまとめてハッシュ値を取る。元データはオフチェーンで保存する。

次のUTXOってなに？

そのUTXOが spent になると、その Single-use Seal で表していた状態は消費されたというか終わったというか、とにかく意味をなさないものとなる。

spent されると次の Single-use Sealが現れるまで「状態が変わったのは確実だが不確定な状態」になる。

UTXOへの送金額に規定はない。

UTXO としてはつながっている必要がある。お釣りが出る場合は自分へ送金するのはBitcoinと同じ。

BitcoinとRGB

Single-use Seal と一言で言っていたが、4つも方法があった

「LNP/BP」という言葉が出てくるが、これは Bitcoin Protocol の上に Lightning Network Protocol がある、というような意味で、Association の名称にもなっている。

Lightning Network

https://github.com/rgb-protocol/RGB-Documentation/blob/master/rgb-over-lightning-network/lightning-network-compatibility.md

(まだあまり読んでいない)

stash はもともとオフチェーンでやりとりするので、オンチェーンでやっていたシールとコミットメントを何とかする。

そのなんとか仕方は、Commitment Transaction を改造して対応する、だった。

シールとコミットメントのところを Commitment Transaction でなんとかしているのだろう。

すなわち・・・

チャネルの相手も RGB 対応している

チャネルの相手も同じトークンを扱っている
ルーティングする経路もすべてそうなっている

という条件がいる。
これは無理やろう。。。
LSP 方式であればもう少しなんとかしやすいのか？
ここまでがんばるメリットはあるのか？
などなど考えてしまう。
GeminiでTaroとの比較もしたが、どちらもおなじくらいだ。Taroもなにがしかのオフチェーンデータが必要だそうな。
トークンに特化した Taro とスマコンに興味がある RGBというところか。

スマートコントラクト

https://rgb.tech/power-user/
情報がいくらか古い気がする。送金コマンドを `rgb transfer` と紹介しているが v0.12 にはない。rgb-sandbox にはあったのでv0.11.1 までは使われていたのかもしれない。

"consignment"という言葉が出てくる。委託荷物とか積み荷とかいう意味の単語。
ピア間で Client Side Validationされたデータを配布するパッケージを指すようだ。

2つのタイプがある。

- contract consignments
  - コントラクトの発行者からユーザへ配布する公開コントラクト情報
- transfer consignments
  - コントラクトユーザ間で状態遷移する際に使用する
  - invoice ってことかな？

「コントラクト(contract consignment)を stash に import する」というような言い方をする。
コマンド名も `import` だ。
状態(state)は Global と Owned があり、トークンそのものの情報が Global、自分の所有状態が Owned になるようだ。

状態遷移は invoice を使う。
invoice を発行する方を beneficiary(受益者)、invoice を実行する方を actor(実行者)と呼ぶ。
invoice には支払い情報が載っていないこともあるので、言葉として「支払い」よりも「実行」を使うそうだ。鍵の無効化や投票などが支払いにならない invoice である。
コマンド `invoice` はオプションが多い。

コントラクトは、contract consignment のバイナリファイルを `import` してから YAML ファイルを `issue` することで使えた。

contract consignment はおそらくこのツールで作るようだ。

pandora-prime/rgb-issuers: RGB contract issuers by Pandora Prime

基本形？があるようで以下の3つが載っていた。

- Fungible RGB20: `F`
- Non-fungible RGB21: `N`
- Uniquely Fungible RGB25: `U`

表は RGB20-FNA 以外は 2025年になっているので、v0.12系からの話か？

グローバルステートを持たないので、できる・できないがある。

RGB - ChatGPT3

「Uniswapの流動性プール」というのは、DEX AMM型みたいなやつだと思う。

🚫 限界・できないこと

1. **グローバル状態を前提にするアプリ**
   - Ethereum型の「Uniswapの流動性プール」や「全員で共有するオンチェーンオークション」など、**全ユーザーが同じ状態にアクセス・更新するタイプのアプリ**は難しい。
   - RGBは「当事者間の部分的合意」しか持たないので、グローバルな共有リソースを安全に扱えない。
2. **自動実行型のロジック**
   - Ethereumのように「誰かがトランザクションを投げなくても、コードが自動的に動いて状態を更新する」ことはできない。
   - RGBは **pull型（ユーザーが操作して初めて状態が進む）** に限定される。
3. **クロスコントラクト連携**
   - Yellow Paper I.0では「クロスコントラクト機能は未対応」と明記されている 📄。
   - 将来的にAtomic Swapなどの外部連携は可能だが、Ethereumのように「Aコントラクトから Bコントラクトを呼ぶ」といったシームレスな連携は難しい。
4. **ガバナンス型DAOの全員投票**
   - 全ユーザーが一つのグローバル投票結果を共有するような仕組みは直接実現できない。
   - 部分的には可能（例えばマルチシグ的な少人数の合意形成）。

逆に何ができるのかというと、トークンとかNFTみたいなタイプ。

基本的に2者間のコントラクトを繰り返していく感じになる。

✅ RGBのスマートコントラクトでできること

1. **単一契約内のロジック実行**
   - 各コントラクトは「独立した部分的状態機械（PRiSM）」として動作。
   - 例えばトークン発行、資産移転、NFT、条件付き支払いなどは可能。
   - 状態の正当性は当事者同士でのみ検証される。
2. **ゼロ知識検証**
   - 状態遷移は zk-AluVM + zk-STARK で証明できる。
   - 計算内容を秘匿しつつ「正しく実行された」ことだけ証明できるため、プライバシーとスケーラビリティに優れる。
3. **所有権ベースのアクセス制御**
   - Single-use seal によって「誰がどの状態を更新できるか」を明確に制御できる。
   - これにより、金融商品や証券のような「権利ベースの契約表現」が得意。
4. **Bitcoinとの統合**
   - L1のUTXO（TxO seals）に紐付けて契約を管理するため、Bitcoinと強く結合した資産発行・移転が可能。
   - 例: RGB20（FT）、RGB21（NFT）、RGB23（証券トークン）。

2025/08/08

面倒なことに提供元が2つに分かれてしまったので、ここではオリジナルである RGB Consortium / LNP/BP Association / RGB-WG が提供している情報を参照する。基本はどちらも同じはずだが今後は分からんので分けておく。



LNP/BPの方

インストール

https://rgb.tech/install/

ローカルのregtestで動かすならRGB Nodeだろう。

と思ったが、ビルドの説明は command line のところしか書いていない。

ChatGPT によると、RGB-WG/rgb はノードとコマンドラインがセットになったようなアプリ、RGB-WG/rgb-node はノードとコマンドラインが分離した構成になっているらしい。

RGB-WG/rgb

https://github.com/RGB-WG/rgb

これで `$HOME/.cargo/bin/rgb` にインストールされた。 `0.12.0-rc.2` ではダメだった。

```
$ cargo install rgb-wallet --all-features --locked --version 0.12.0-rc.3
```

ソースコードからのビルドはこちらでよいのか。

```
$ cargo build --workspace
```

Unable to install RGB on my ubuntu machine · Issue #292 · RGB-WG/rgb

`./example/demo.sh` でデモが動くようなことを書いてあるが、ビルドに失敗したり、変更しても実行に失敗したりするので多少変更した。

https://github.com/hirokuma/rgb/tree/work/v0.12

- Alice, Bob: `init`
- Alice: `import <file>`
  - `RGB20-FNA.issuer` というバイナリファイルを読み込む
    - compiled/RGB20-Simplest-v0-AYkSrg.issuerと同じファイルだった。
  - 成功すると `examples/data/bitcoin.testnet/Fungible Non-inflatable Asset.7C15w3W1-L0T~zXw-Aeh5~kV-Zquz729-HXQFKQW-_5lX9O8.issuer` というファイルができた。
  - `rgb contracts` には出てこない
- Alice: `issue --wallet alice` (初回wallet)
  - コントラクト定義ファイル `DemoToken.yaml` をコンパイル
  - 成功すると `examples/data/bitcoin.testnet/DemoToken.nztjpjRX-_NVDDgd-EHENsqL-AKsGsyV-F6LzljB-X4WTv0Y.contract/` が作られ中にいろいろファイルが置かれていた。
  - これが stash ?
  - `rgb contracts` に出てきた。
- Alice: `backup`
  - `examples/DemoToken.rgb` が作られる
  - おそらく contract consignment だろう
- Bob: `accept --wallet bob` (初回wallet)
  - Alice が `backup` コマンドで作った `DemoToken.rgb` を与える
  - `-u` オプション(allow unknown contract)がないとエラーになる
  - `examples/data2/bitcoin.testnet/DemoToken.nztjpjRX-_NVDDgd-EHENsqL-AKsGsyV-F6LzljB-X4WTv0Y.contract/` 以下が作られる

Alice, Bob: `contracts`

どちらも同じ出力になった。

`examples/dataX/bitcoin.testnet/DemoToken.nztjpjRX-_NVDDgd-EHENsqL-AKsGsyV-F6LzljB-X4WTv0Y.contract/` にあるデータを読んでいるようで、ディレクトリを削除すると何も出力されない。

`contracts`

```
Contracts:
---
id: contract:nztjpjRX-_NVDDgd-EHENsqL-AKsGsyV-F6LzljB-X4WTv0Y
name: DemoToken
issuer: ssi:anonymous
timestamp: 2024-12-18T12:32:00Z
codex:
  id: 7C15w3W1-L0T~zXw-Aeh5~kV-Zquz729-HXQFKQW-_5lX9O8#heavy-cactus-quiet
  name: Fungible Non-inflatable Asset
  developer: dns:pandoraprime.ch
  timestamp: 2024-11-25T10:08:27Z
consensus: bitcoin
testnet: true
```

Alice, Bob: `state -og --wallet`

`-o` : owned state、 `-g` : global state

`-a` : all なのだが、なぜか `-g` が必要らしい。

違いはほとんどなく、Aliceの方に `Owned.balance` の値が入っているだけである。

testnet4 の outPointが載っているのだが、そのamount=50,000 と `Owned.balance` の value=10,000 になっていて値は関係が無い

outPointは `DemoToken.yaml` に載っていた

Bob: `invoice --wallet --seal-only` : `AUTH_TOKEN` ←これは使っていない

`--seal-only` は Single-use Seal の作成のみで invoice の作成ではない

`AUTH_TOKEN` にランダム性はなさそう

`AUTH_TOKEN`

```
$ echo $AUTH_TOKEN
at:FWKMBVDc-NBk8dAZA-tYGt2HIP-MtzSRO17-8g3QvRIN-nuPjug
```

Bob: `invoice --wallet` : `INVOICE`

`INVOICE` の末尾は `AUTH_TOKEN` になっていた。

`INVOICE`

```
$ echo $INVOICE
contract:tb@contract:nztjpjRX-_NVDDgd-EHENsqL-AKsGsyV-F6LzljB-X4WTv0Y/10@at:FWKMBVDc-NBk8dAZA-tYGt2HIP-MtzSRO17-8g3QvRIN-nuPjug/
```

Alice: `pay --wallet`

Bobの作った invoice に Alice が支払うのだが、invoice 以外に保存する consignment を指定する。PSBT はオプションで、こちらも保存するファイル名を与える。

これだけではチェーンに展開されないようだ。 `finalize` コマンドを使って PSBTに署名して展開するのか？

PSBTファイルをbase64エンコードしてdecodepsbt


```
{
  "tx": {
    "txid":
"4f41811b210545aff4694981237d2cc1ce5d4334e874a24fd582305cb1d6f5fa",
    "hash":
"4f41811b210545aff4694981237d2cc1ce5d4334e874a24fd582305cb1d6f5fa",
    "version": 2,
    "size": 125,
    "vsize": 125,
    "weight": 500,
    "locktime": 0,
    "vin": [
      {
        "txid":
"b7116550736fbe5d3e234d0141c6bc8d1825f94da78514a3cede5674e9a5eae9",
        "vout": 1,
        "scriptSig": {
          "asm": "",
          "hex": ""
        },
        "sequence": 0
      }
    ],
    "vout": [
      {
        "value": 0.00000000,
        "n": 0,
        "scriptPubKey": {
          "asm": "OP_RETURN
4bf149c43891940177b1b28f31ff374357e1fe3635f31003f814ccdd1e3993a6",
          "desc":
"raw(6a204bf149c43891940177b1b28f31ff374357e1fe3635f31003f814ccdd1e3993a6)#zg
uvcyf0",
```

```
          "hex":
"6a204bf149c43891940177b1b28f31ff374357e1fe3635f31003f814ccdd1e3993a6",
          "type": "nulldata"
        }
      },
      {
        "value": 0.00049000,
        "n": 1,
        "scriptPubKey": {
          "asm": "0 46fa7f250d6d0b430c423cddd706262c12b5bb04",
          "desc":
"addr(bcrt1qgma87fgdd595xrzz8nwawp3x9sfttwcy322n78)#9kxv5fz6",
          "hex": "001446fa7f250d6d0b430c423cddd706262c12b5bb04",
          "address": "bcrt1qgma87fgdd595xrzz8nwawp3x9sfttwcy322n78",
          "type": "witness_v0_keyhash"
        }
      }
    ]
  },
  "global_xpubs": [
    {
      "xpub":
"tpubDCpZeo9QAvMetovHqqqMgzRnWU2Wy5cgZUG9g3v6SJLihmARTw2sSNga8xfZTqqAFJpoTgEn
VjwLjUEmf36GZ2gzba5w3GbjDbxLitzDGZA",
      "master_fingerprint": "cf05f8ea",
      "path": "m/84h/1h/0h"
    }
  ],
  "psbt_version": 0,
  "proprietary": [
  ],
  "unknown": {
  },
  "inputs": [
    {
      "witness_utxo": {
        "amount": 0.00050000,
        "scriptPubKey": {
          "asm": "0 b114525cdb2fd8de581b0a4a287d58e623736c46",
```

```
          "desc":
"addr(bcrt1qky29yhxm9lvdukqmpf9zsl2cuc3hxmzxhg7y32)#8skaqcqy",
          "hex": "0014b114525cdb2fd8de581b0a4a287d58e623736c46",
          "address": "bcrt1qky29yhxm9lvdukqmpf9zsl2cuc3hxmzxhg7y32",
          "type": "witness_v0_keyhash"
        }
      },
      "bip32_derivs": [
        {
          "pubkey":
"02032e391437f8202eb87d728a9c4d7fa35721acaaeb43a478e057bc3dbf37121a",
          "master_fingerprint": "cf05f8ea",
          "path": "m/84h/1h/0h/0/1"
        }
      ],
      "proprietary": [
        {
          "identifier": "4d4d42",
          "subtype": 1,
          "key":
"fc034d4d42019f3b63a63457f8d5430e07441c436ca8b00ab06b32545e8bce58c15f8593bf46
",
          "value":
"0bc4b7377e598f00491a85b484cacf8d498c346f82d4eca26ffdea49e6f37f98"
        }
      ]
    }
  ],
  "outputs": [
    {
      "proprietary": [
        {
          "identifier": "4d5043",
          "subtype": 0,
          "key":
"fc034d5043009f3b63a63457f8d5430e07441c436ca8b00ab06b32545e8bce58c15f8593bf46
",
          "value":
"a3285029f10fd7d5b96492796bed07434f899364dbc7c91b2eaade6961dbf7e9"
```

```
        },
        {
          "identifier": "4d5043",
          "subtype": 1,
          "key": "fc034d504301",
          "value": "e81aaf314b516bcf"
        },
        {
          "identifier": "4d5043",
          "subtype": 16,
          "key": "fc034d504310",
          "value":
"4bf149c43891940177b1b28f31ff374357e1fe3635f31003f814ccdd1e3993a6"
        },
        {
          "identifier": "4d5043",
          "subtype": 17,
          "key": "fc034d504311",
          "value":
"000300000800000000386a934216c653f3565826a9598217529282075c32bbcf013b38a4e35
96cbdadb0003962dae39e5537b44f5c6843beebfddba9d32965bcb29b5726fca867a7dfbd2e90
003372e6ae7f2ea583be8cd42a3a241a913a045a79665f731c8d4dff104c3ffdb9c0003125829
29d951b9f406c3e0c1975a97dfcbae912637ffaac8b092950645ab531a00039c498ccd371e9ca
d051bfb540cf5f7ba96d5ee7b08c030b680b0166d96048eb10003f2bb0d5058b769761ed70ede
9e71b7b30d455e36728b9b82adbec8fcf6b312950003de5b199a9be2c5c9ee0bf854d56cc0dae
48b9166dc4edbec4a5a7ca8bc6f271f019f3b63a63457f8d5430e07441c436ca8b00ab06b3254
5e8bce58c15f8593bf46a3285029f10fd7d5b96492796bed07434f899364dbc7c91b2eaade696
1dbf7e901e81aaf314b516bcf"
        },
        {
          "identifier": "4f50524554",
          "subtype": 1,
          "key": "fc054f5052455401",
          "value":
"4bf149c43891940177b1b28f31ff374357e1fe3635f31003f814ccdd1e3993a6"
        }
      ]
    },
    {
```

```
        "bip32_derivs": [
          {
            "pubkey":
"0395e64e94e5381c0cc4594d98b4f6f65887796b86c412cce1bc188165c0d3cfa1",
            "master_fingerprint": "cf05f8ea",
            "path": "m/84h/1h/0h/1/1"
          }
        ]
      }
    ],
    "fee": 0.00001000
  }
```

Alice: `state -goa`

`-a` を付けないと "tentative"(一時的) な balance が出力されなかった

`9990` なので自分に対するお釣りか？

outPoint も出力されていて、 `pay` したPSBTファイルを展開したときのTXIDと一致していた

Bob: `accept --wallet`

今回は Alice が `pay` したときの transfer consignment を使う

Bob: `state -go`

こちらに `-a` をつけると、Alice の `state` で出力された tentative な方も出力された

つけない場合でも Bob への tentative な balance である 10 が出力された

TXID は今までにないものだが、testnet4 には展開済みだった。まだ UTXO なので次のシール？

いろいろ

ウォレット作成 `rgb create <NAME> <DESCRIPTOR> <flag?>`

`<NAME>` : `DATADIR/bitcoin.testnet/<NAME>/` が作られ、その下にファイルが置かれる

`<DESCRIPTOR>` : `bitcoin-cli listdescriptors false | grep "pub"` で extended pubkey を出力し、サンプルにあわせるなら `wpkh()` の desc を探す。たぶん 2つあるので、最初にある方でいいんじゃないかな？ 引数にするのは `wpkh()` の中身だけである。

`<flag?>` : `--tapret-key-only` か `--wpkh` のどちらかがいるようだ。 `<DESCRIPTOR>` で `wpkh()` を選んだからここは `--wpkh` にした。

これで抽出できそう

```
$ bitcoin-cli listdescriptors false | jq -r '[.descriptors[] | select(.desc |
test("^wpkh")) | .desc][0] | sub("^wpkh\\("; "") | sub("\\)#.*$"; "")'
$ $RGB create carol
```

```
"[959f8be5/84h/1h/0h]tpubDCAiniUxuXW9VKrEckNqCoF2TnXNkFcetRC5h6R7w7Bc71L3X82meHP
sVDdXVFhLDg379nAXviQPL6fm1j2E8Ze5SCEtLu7pvjj4yEFMEbZ/1/*" --wpkh
```

contract/トークンの発行

コントラクト定義ファイルというYAMLを作る(他のフォーマットもあるそうだ)。DemoToken
https://rgb.tech/power-user/#issue

コントラクトのベースみたいなものがある。それは AluVMで動くもので `rgb` コマンドではコンパイルされたバイナリファイルを与えている(`rgb import`)。

RGB-WG/rgb-node

こちらがノード？
https://github.com/RGB-WG/rgb-node

ノードは2つの動作モードがある

スタンドアローンのサーバー

組み込みサービス

Rust の crate は 3つ(括弧内はバイナリ名)

`rgb-node` (`rgbd`)

`rgb-client` (`rgb-cli`)

`rgb-rpc`

リリースバージョンは2年前だが、Rustの Cargo.toml を更新しているようだ。

RGB Protocol Associationの方

サンドボックスがあって権藤さんも動かしたことがあるようだったので、こちらを試す。。。
何をしているか分からないが、動きはした。
GitHub - rgb-protocol/rgb-sandbox at a4063996d3fe77516e609a6fed9c30d0770cbfcb
ログからすると、このバージョンは RGB v0.11.1系と思われる。

> Installing bp-wallet v0.11.1-alpha.2+unreviewed
> Installed package `rgb-cmd v0.11.1-rc.4` (executable `rgb`)

RGB-WG は v0.11.1系は alpha までで、beta や rc はリリースせず v0.12.0 をリリースしたとブログにも書いている。
alpha ではあるのだが、RGB Protocol Association のほうで紹介されていたので