Blockchain Specifications

Vincent Berthier

May 16th, 2024

Contents

1 Introduction

Bangk activities depend on the Solana blockchain [1], which uses some specific terms and language that are defined in 1.1 on top of some concepts specific to, particularly where tokens are concerned (*cf.* 1.1.3).

The Solana economic model, particularly those relevant to Bangk are presented in the 1.2 section to go over the differences between “Fees” (*cf.* 1.2.1) and “Rents” (*cf.* 1.2.2).

Tools and utilities used to interface with the Solana blockchain in general, and Bangk programs specifically will be presented at the end of this introduction in section 1.3.

1.1 Definitions

While most terms used for the Solana Blockchain are common to the crypto-universe, some are slightly different or entirely new to better take into account Solana specificities, particularly where “Accounts” (*cf.* 1.1.2) and “Programs” (*cf.* 1.1.1) are concerned.

In section 1.1.3 the tokens specific to Bangk will be presented. For a more in-depth understanding of those tokens, refer to the Bangk White Paper.

All other terms are defined in the Solana dictionary: `https://docs.solana.com/terminology`. We will only mention the “Lamport” which is defined as $\frac{1}{10^9}SOL$: it’s the unit used when talking about fees.

1.1.1 Program

A Solana Program is nothing more than what’s called a “Smart Contract” on other blockchains. It’s through them that all operations are performed on the blockchain.

There are two types of programs:

- Native - or sea-level - program: those are internal to the Solana ecosystem and are necessary to its day to day operations,
- “On-Chain” program: they are integrated on the blockchain, stored in accounts, and maintained by users.

Sea-level programs define all the atomic operations that can be performed on the Solana blockchain (*e.g.* to transfer SOL from one account to another, create accounts, *etc.*) while “On-Chain” programs use those atomic operations to perform more complex operations (*e.g.* in DeFi applications for example).

Bangk will deploy two programs on the Solana Blockchain: one to mint the utility tokens and handle the vesting period, the other one to deal with Stable and Security tokens (*cf.* 1.1.3).

1.1.2 Accounts

In the Solana blockchain, an “Account” represents a virtual file on the blockchain where data is stored. SOL wallets, programs, mints and many more are accounts. Generally, there are two types of accounts: those associated to a Keypair (private key + public key), and those who only have a public key.

On the Solana blockchain, a Keypair (composed of a public and private key) is generated on the elliptic curve ed25519 [2]. Those key represent an “identity” which are used to sign transac-

1. https://solana.com
2. See `https://www.rfc-editor.org/rfc/rfc8032` for the technical paper, or `https://fission.codes/blog/everything-you-wanted-to-know-about-elliptic-curve-cryptography/` for a higher level explanation

tions on the blockchain. Since most accounts are identities, and thus are not supposed to sign a transaction, they do not need to have a Keypair, only a public key to designate them.

Those public keys are derived from seeds that can be (mostly) anything, but such that for a given set of seeds, the same public key will always be generated. Should that public key fall on the ed25519 curve, it's slightly shifted with a "bump" to make sure it falls outside the curve (and thus doesn't have a private key).

There are two main families of such accounts. The most common one is called "Program Derived Address" (or PDA) and are used to store data required by on-chain programs. They can be seen as files or database tables, but token mints are also a type of PDA. One important thing to note is that using the tuple seeds + bump, a program can sign instructions operating on those PDAs.

The second family is the "Associated Token Address" (or ATA), which are effectively a sub-family of PDAs. Those accounts are associated to the tuple of User Wallet + Mint (there addresses are the seeds of the PDA). Since the generation of the address is deterministic, it ensures that all tokens of a given type are stored for each user in one and only one account. Contrary to 'normal' PDAs, ATAs are not owned by a program thus requiring that a user sign transactions impacting them.

Those accounts are all subject to fees to be stored on the blockchain (*cf.* 1.2.2). One important thing to note is that while PDAs can all be read from anywhere, only the program that owns them can perform write (or delete) operations on them. In the case of an ATA, only the user owning the ATA can burn or transfer out its tokens.

Bangk programs use those PDAs extensively: mints, investment records or ICO data is stored on PDAs while all tokens are stored in ATAs.

1.1.3 Tokens

In the crypto-universe, and *a fortiori* on the Solana blockchain, a token materializes goods that can be digitally transferred. They can represent currencies (all cryptocurrencies are tokens), but also pictures, deeds, *etc.*In the first case it's possible to transfer fractions of tokens while in the second it's only possible to transfer full tokens. In some cases there exists one and only one token, which cannot be divided: those are Non-Fungible Tokens (or NFT).

Bangk will use three different types of tokens:

Stable Those tokens are the stable coins equivalents to fiat currencies: the Euro (€) will be associated to the *Euro BANGK* (EUB), the US Dollar ($) to the *Dollar BANGK* (USB), *etc.*Those tokens are considered "stable" because they will never be worth more or less than exactly one unit of their respective fiat currency (or one of their sub-denomination, such that one EUB token is worth a euro cent).

Security Generally called *Bangk INVEST* (BST), those tokens are in fact a family of tokens representing parts of a project that are funded by buying tokens. By owing those tokens, user can receive at regular intervals dividends at a fixed or variable rate which will be paid in stable tokens.

Utility *Bangk COIN* (BGK) tokens are the closest Bangk will offer to 'regular' cryptocurrency tokens since, contrary to the first two, they will be offered in the usual trading platforms (*e.g.* Binance, eToro, Swissborg, *etc.*).

For a more in-depth presentation of those tokens, see Bangk 's white paper.

1.2 Usage costs

The Solana blockchain finances clusters creating / validating new blocks by applying fees for validating signatures for each transaction (*cf.* 1.2.1). Another thing to take into account, while not a 'cost' *per se* is rent exemptions (*cf.* 1.2.2) applied to keep data on the blockchain.

1.2.1 Fees

Solana transactions can only be done when fees are paid to finance computation costs on the validation clusters. Those fees can be direct or indirect.

Indirect fees To trigger transactions on the blockchain, it is necessary to send the request to a validation cluster through the RPC protocol. The cost of doing so depends entirely on the targeted cluster: Solana itself for example is completely free, but limited to at most 100 request per 10 seconds periods, and can at any moment change those limitations.

If more requests are needed (or guarantees that the limits will not change), the validation cluster needs to be self-hosted (which is a rather involved mechanism, although rewarding), or an external service must be used to get access to their own clusters, with costs that depend on that service (the most well known service for that is QuickNode[3]).

Direct fees When a transaction is a simple request (*e.g.* get the balance of a wallet, read the data of an account, *etc.*) only indirect fees need to be taken into account. In all other cases however, where a modification of the blockchain is necessary (*e.g.* to transfer tokens from one wallet to another, to modify the data of an account, *etc.*), direct fees need to be paid.

In all cases, those fees only depend on the number of signers for the transaction, such that $cost = N_{signers} \times 5'000 lamports$.

There is however another type of direct fees: *priorization fees*. While the Solana blockchain can perform a large number of instructions each second, it can happen that some validators have to choose which transactions to include in the next block, which means that some are ignored. This can lead to some transactions failing if the network is congested at a given moment. To prevent this, it's possible to add *Priority Fees* which are a 'reward' paid to the validator. Priority fees are defined in µLamports, and are multiplied by the number of Compute Units used by the transaction to get the total fee.

Since the importance of Priority Fees is heavily dependent on the state of the network, it is impossible to give a definite cost for them: in most cases, no priority fees will be just fine but when they are needed, there is no set amount that will be good in any situation (although 1'000 µLamports might be more than enough). The cost remains very limited still: at that level, the cost is *at most* increased by 200 Lamports (for a minimal transaction cost of 5'000 Lamports).

1.2.2 Rents

Any account on the blockchain requires holding a sufficient balance to be kept there. The amount is linearly proportionate to the size of the account, such that the rent for 15ko of data amounts to 1SOL.

Some rent examples:

- Mint: 1 461 600 Lamports
- ATA : 2 039 280 Lamports

3. https://www.quicknode.com

Those amounts are effectively quite small, but should not be underestimated as they can grow quite a bit depending on the number of PDAs / ATAs that are used. Furthermore, since those amounts are blocked, their actual value is subject to the variation of the price of the SOL.

1.3 Utilities & Libraries

Solana programs first depend on tools delivered by the Solana Foundation (*e.g.* the CLI tools, the Rust Libraries such as *Solana_program*[4] or *spl_token*[5], *etc.*)

1.3.1 Borsh

Another important dependency is Borsh[6] which is used to (de)serialize data to send to the blockchain. This library is preferred to any other first because it's the *de facto* standard in the Solana ecosystem, but also because it benefits from formal specifications which is not the case of Serde[7] which is the other main option.

1.3.2 Anchor

Anchor[8] is a framework designed to write Solana On-Chain programs. As such, it offers a number of tools and standard practices that simplify development…but with some costs. While standardization has its pros and cons, and making stepping out of the beaten path harder can be mitigated, the fact that Anchor performs unnecessary operations limits the scale of the programs which can be a problem.

"Native" development can be a bit riskier (since there are no security checks by default), but it allows more freedom and the risks can be completely mitigated by following good practices[9] for example.

As such, we have made the choice to not use Anchor.

4. `https://docs.rs/solana-program/latest/solana_program/system_instruction/index.html`
5. `https://docs.rs/spl-token/latest/spl_token/instruction/index.html`
6. `https://borsh.io/`
7. `https://serde.rs/`
8. https://www.anchor-lang.com
9. Some of those can be found at `https://neodyme.io/en/blog/solana_common_pitfalls`

2 Accounts

As defined earlier (*cf.*1.1.2) un account stores all data used by on-chain Solana programs. Bangk will of course use generic account as is (*e.g.* ATAs to store token balances) or with some tweaks (*e.g.* Mints where extensions will be activated).

In addition to that, Bangk will use some specific PDAs which will be described below.

2.1 Mints

Mints used in Bangk 's programs will all use extensions from the SPL Token 2022 program [10] to at least embed metadata in the mints. All the mints will have least have:

- Name (between 5 and 128 characters),
- Symbol (between 3 and 5 characters),
- URI (between 5 and 128 characters).

2.1.1 Bangk Coin

Utility tokens' mint is bare bone, with nothing added to the default. Its metadata are:

- Name: Bangk COIN
- Symbol: BGK
- URI:

Added to the base size of the mint, that's a total of XXX bytes which implies a rent exemption of X XXX XXX Lamports (or 0,00XX SOL).

Address Seeds

- "BGK",
- Bangk Main public key.

Authorities

- Mint: None
- Freeze: Multisig 2/3
- Metadata: Multisig 2/3
- Close: None
- Permanent Delegate: N/A

2.1.2 Stable Coins

On top of the metadata (and metadata-pointer) extensions, stable coin mints will have the *Permanent Delegate* extensions activated (as a legal requirement). Besides that, the metadata of each stable coin mint will obviously be different.

10. `https://docs.rs/spl-token-2022/latest/spl_token_2022/`

Address Seeds

- Stable coin symbol,
- Bangk Main public key.

Authorities

- Mint: Bangk 's delegate
- Freeze: Multisig 2/3
- Metadata: Multisig 2/3
- Close: None
- Permanent Delegate: Bangk 's delegate

2.1.3 Security Tokens

Security token mints will have the same extensions activated as the stable coins one. In their case however, on top of Name-Symbol-URI a number of fields will be added to the metadata to register important information associated to the project:

- ATA: public key address of the stable coin ATA used by the project,
- rate: interest rate (either projected or fixed depending on the type of the project),
- last: timestamp of the last payment (0 if none have been made),
- next: timestamp of the next payment (0 if none are planned yet),
- value: value of one of the project's token in stable coins (in the currency used by the project),
- risk: risk associated with the project on a scale going from 1 (least risky) to 7 (most risky),
- period: payment periodicity of the project's dividends,
- status: current status of the project (Open, Live, Closed or Cancelled).

All the metadata fields are strings. Next and last are always defined as 10 characters, even for a value of 0.

Payment periodicity is defined as:

- 0 : Daily
- 1 : Weekly
- 2 : Monthly
- 3 : Quarterly
- 4 : Bi-annually
- 5 : Annually

Project status is defined as:

- 0 : Open, project currently looking for funding,
- 1 : Live, project has been funded and regularly pays dividends,
- 2 : Closed, project has ended,
- 3 : Cancelled, project has been cancelled in the funding phase.

Address Seeds

- Project symbol,
- Bangk Main public key.

Authorities

- Mint: Bangk 's delegate / None
- Freeze: Multisig 2/3
- Metadata: Multisig 2/3
- Close: Multisig 2/3
- Permanent Delegate: Bangk 's delegate

2.2 ICO settings

In this account, the first parameter will be a timestamp (i64) for the launch date of the coin post-ICO. After that, parameters for the unvesting period will be defined for each type of unvesting category. They will be stored as a HashMap with the key being the category type:

- 0: Team & Founders,
- 1: Advisors & Partners,
- 2: Private Sells,
- 3: Public Sells (weeks 1-11),
- 4: Public Sells (weeks 12-19),
- 5: Public Sells (weeks 20-26).

For each of those categories, the following parameters will be defined:

- Unvesting start in weeks (u8),
- Unvesting duration in weeks (u8),
- Initial unvesting release percentage (x1000) (u16),
- Weekly unvesting percentage (x1000) (u16),
- Final unvesting percentage (x1000) (u16),

In the case of the Advisors & Partners, the values will only be indicative as the exact start and duration of the unvesting can be negotiated individually, in which case the parameters defined in the following account will override the defaults.

Following that will be the Pubkey allowed to perform regular operations (such as create or update an investment) and the list of Pubkeys allowed to perform admin operations for the ICO.

2.3 ICO Investment

This account symbolizes a client's investment during the ICO. It contains a HashMap with the key being the type of the investment (*e.g.* private sell phase 1, team member, *etc.*) and the value a structure of:

- Number of allocated tokens (u64)
- Number of released tokens (u64)
- Custom start of the unvesting (u8)
- Custom length of the unvesting (u8)

Address Seeds

- Client's public key,
- BGK mint,
- Bangk Main public key.

2.4 Project's dividends tracker

Associated to a project, this PDA stores data regarding the payment of dividends to make sure that no user is paid twice for the same period and that none are skipped.

Table 1 – Data of the dividends tracker

Data	Type	Size
PDA type	u8	1
Bump	u8	1
Project's mint address	Address	32
Payment date	Timestamp (i64)	64
Number of clients	u32	32
Paid clients	u32	32

Address Seeds

- Project's mint,
- Bangk Main public key.

2.5 Client's investment

This PDA keeps track of one client's investment on a given project. Its purpose is mostly to make sure, in coordination with the previous one, that no client gets paid twice or skipped at any given round of dividends payment.

Address Seeds

- Client's public key,
- Project's mint,
- Bangk Main public key.

Table 2 – Data for client's investment

Data	Type	Size
PDA type	u8	1
Bump	u8	1
Client's public key	Address	32
Project's mint address	Address	32
Client's stable ATA	Address	32
Creation date	Timestamp (i64)	64
Last payment	Timestamp (i64)	64

3 Security

While the blockchain itself is extremely secure, frauds can still happen. Up to now however, any such cases was caused by user wallets being compromised. In order to mitigate that risk as much as possible and prevent single-point failures (*e.g.* Bangk 's private key gets lost) Bangk will secure most transactions using multi-signatures authorizations.

3.1 Multisig schemes

In the Solana ecosystem, there are three main schemes to use multi-signatures.

3.1.1 Off-line

The first way to do so is through an offline platform such as Snowflake [11] or Squads [12]. The default idea is that on the blockchain only one signer is needed for an instruction, but the associated Keypair is only known by the platform. To get it to sign a transaction, registered users need to create the transaction on the platform, then have allowed users authorize it. Once the required threshold has been reached, the instruction is sent on the blockchain.

The fact that it introduces a new platform for internal operations is less than optimal for Bangk 's needs as it makes the flow of operations more complex. On top of that, while it works very well for 'standard' instructions (*e.g.* program updates, token transfers, *etc.*) custom program instructions require ad-hoc development of 'plug-ins' into the platform. It would make sense for someone who doesn't already have (or want) a backend structure, but such is not the case for Bangk .

Despite that, there is still one operation that will be performed through one such platform: program update.

3.1.2 SPL Token 2022

The multisig scheme fully integrated with the SPL Token 2022 program allows to define multisig accounts wherever a owner / authority is expected: mint or freeze authorities, token owners, *etc.*It makes such operations easy to set up, use, and log directly on the blockchain. There is however one major drawback: once a multisig account has been defined, it *cannot* be changed. Thus should a Private Key associated with the multisig be compromised (*e.g.* lost, stolen, *etc.*) it will not be possible to remove it from the multisig definition.

While in some cases it's not much of an issue (if it's a token owner, just transfer the tokens to a new ATA with a new multisig owner), for things like mint authorities or delegation it's less ideal.

3.1.3 Program Controlled

In this setup, the program directly controls the multisig: it is the one checking the multisig from its own configuration (a set of keys registered in a PDA for example), and the program itself will then sign CPI instructions by setting the owner / delegates / authorities to be a PDA it has created.

By doing that, we can ensure that not only every instruction can only be performed if the required number of authorized signatures are present, but also that *only* the program itself can

11. https://snowflake.so
12. https://squads.so

perform critical CPI instructions. More importantly, the configuration *can* be updated: at any moment authorized signers can be added or removed without needing to change anything in any of the Mints / ATAs.

3.2 Instruction security

In Bangk 's program, we identify three levels of operations:

- Critical operations,
- Sensitive operations,
- Routine operations.

While they will all follow the same approach, the number of required signers (and their role) will be different.

3.2.1 Critical operations

Critical operations are instructions that, mishandled, can compromise the whole ecosystem or impact all users: update the configuration of the program, change a coin's metadata, *etc.*Considering the potential scale of the consequences of such malicious instructions, they will require three authorized signatures.

Which signatures are allowed will be defined in each program's configuration PDA, but the allowed ones are:

- Bangk 's API key
- Bangk 's president
- Bangk 's chief of operations
- Bangk 's administrator
- Bangk 's community

3.2.2 Sensitive operations

Sensitive operations are instructions that could cause issues limited in scope and severity, for example changing the state or update the interest rate of a project, freezing an account, perform large transfers. Those operations will require two authorized identities to sign the transaction.

In effect, what it means is that since one of these keys will be Bangk 's API key, the instruction will need to be signed by someone else officially. What the others authorized signatures will depend on the type of operation performed: the legal team will be the only one able to freeze / unfreeze an account, while updating an investment project will be done by the team in charge of those.

3.2.3 Routine operations

Routine operations are instructions that if mishandled will have a very small impact and that, more importantly, can easily be rolled back if necessary (*e.g.* triggering the payment of dividends for a project). Those operations will not use a multisig scheme in most cases in order to keep the flow of operations as smooth as possible. In some cases however (*e.g.* for large transfers), we may require an additional signature.

4 Design & Quality

One of the - if not *the* - most important things when implementing a complex program on the Solana blockchain - or any blockchain really - is to take security into account such that the program can only be executed by authorized identities and no unintended instructions can take place.

To that end, it is important to be aware of the common problems [13] to make sure to avoid them. To avoid most implementation issues, the code is heavily checked with Rust ecosystem tools and the program itself is extensively tested with the goal of maintaining a rate at or above 95% coverage.

4.1 Runtime checks

Those checks are performed at runtime, implemented through Rust macros. Their purposes are to check the integrity of the instructions.

check_signers This check ensures that the signer are valid and that there are as many as expected (from one for routine operations to three to critical instructions).

check_pda_owner Checks that the program owns a given PDA, preventing unknown PDAs to be substituted to the expected ones.

check_same_owner Checks that a given set of accounts (PDAs and/or ATAs) are all related to the same user. For example a stable coin ATA, a security token ATA and an investment tracker PDA are all related to the same client.

check_user Checks that an account belongs to a given user.

check_bangk_owner Checks that one or more accounts are all Bangk 's (for example stable coin ATAs used for exchange purposes).

check_mint_ata Checks that one or more ATA match the given mint.

check_investment Checks that a tuple of three accounts are a valid set of stable coin ATA, security token ATA and investment tracker.

check_project_status Checks that the current status of a project is the valid one for a given instruction.

check_system_program Checks that a given account is the expected system program.

check_spl_program Checks that a given account is the expected SPL Token 2022 program.

13. Some of them can be found here: `https://neodyme.io/en/blog/solana_common_pitfalls/`

check_ata_program Checks that a given account is the expected Associated Token Account program.

4.2 Rust tools

The Rust ecosystem provides a big number of tools to ensure the quality and security of a program. On top of cargo-fmt and cargo-spellcheck (to make sure formatting standard is enforced, and the comments are well written), we use:

cargo-clippy While the default parameters for clippy are decent enough [14], we have decided to use a much more restrictive configuration, so we have modified the default configuration to:

- disable *module_name_repetitions*
- enable all **nursery** checks,
- enable all **pedantic** checks,
- enable the following **restriction** checks:
  - absolute_paths
  - alloc_instead_of_core
  - allow_attributes
  - allow_attributes_without_reason
  - arithmetic_side_effects
  - assertions_on_result_states
  - as_underscore
  - big_endian_bytes
  - clone_on_ref_ptr
  - create_dir
  - dbg_macro
  - default_numeric_fallback
  - default_union_representation
  - deref_by_slicing
  - disallowed_script_idents
  - else_if_without_else
  - empty_drop
  - empty_structs_with_brackets
  - error_impl_error
  - exit
  - expect_used
  - filetype_is_file
  - float_cmp_const
  - fn_to_numeric_cast_any
  - format_push_string
  - get_unwrap
  - host_endian_bytes

14. They can be found at: `https://rust-lang.github.io/rust-clippy//master/index.html`

- if_then_some_else_none
- impl_trait_in_params
- indexing_slicing
- inline_asm_x86_att_syntax
- inline_asm_x86_intel_syntax
- integer_division
- iter_over_hash_type
- large_include_file
- let_underscore_must_use
- little_endian_bytes
- lossy_float_literal
- map_err_ignore
- mem_forget
- min_ident_chars
- missing_assert_message
- missing_asserts_for_indexing
- mixed_read_write_in_expression
- multiple_inherent_impl
- multiple_unsafe_ops_per_block
- mutex_atomic
- needless_raw_strings
- panic_in_result_fn
- panic
- partial_pub_fields
- print_stderr
- print_stdout
- pub_use
- pub_without_shorthand
- rc_buffer
- rc_mutex
- redundant_type_annotations
- rest_pat_in_fully_bound_structs
- same_name_method
- self_named_module_files
- semicolon_inside_block
- shadow_unrelated
- string_add
- string_slice
- string_to_string
- str_to_string
- suspicious_xor_used_as_pow
- tests_outside_test_module
- todo

- try_err
- undocumented_unsafe_blocks
- unimplemented
- unnecessary_safety_comment
- unnecessary_safety_doc
- unnecessary_self_imports
- unneeded_field_pattern
- unreachable
- unseparated_literal_suffix
- unwrap_in_result
- unwrap_used
- verbose_file_reads

cargo-audit Checks for vulnerabilities within the program's dependency. Any dependency detected must then be assessed. Those found at the moment are:

- Errors
  - RUSTSEC-2022-0093: a solana-sdk dependency with a fixed version
  - RUSTSEC-2024-0332: Solana dependency
  - RUSTSEC-2024-0336: Solana dependency
- Warnings
  - RUSTSEC-2021-0139: Solana dependency
  - RUSTSEC-2021-0145: Solana dependency
  - RUSTSEC-2023-0033: Solana dependency
  - RUSTSEC-2023-0042: Solana-runtime dependency

cargo-deny While it also checks for vulnerabilities, its main use here is to check that no copyleft licenses from dependencies contaminate the program. The allowed licenses are:

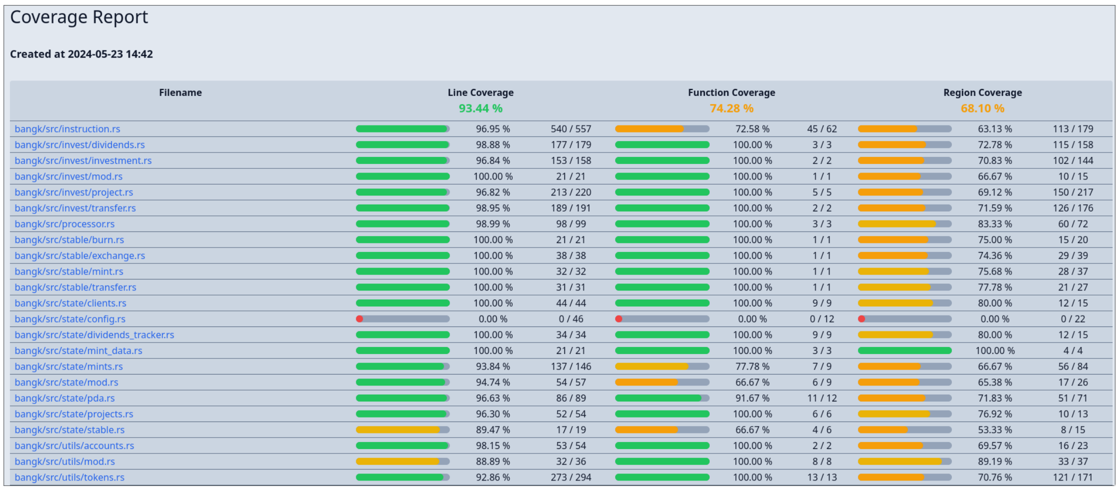
- Apache-2.0
- Apache-2.0 WITH LLVM-exception
- BUSL-1.1
- CC0-1.0
- MIT
- MPL-2.0
- BSL-1.0
- BSD-2-Clause
- BSD-3-Clause
- ISC
- Unlicense
- Zlib

With two exceptions:

- Unicode-DFS-2016, used by *unicode-ident*
- OpenSSL, used by *ring*

4.3 Unit tests

Finally, unit tests are run to ensure that everything works as intended and no regressions happen with a goal of reaching a line coverage of 95+%.



Coverage Report

Created at 2024-05-23 14:42

Filename	Line Coverage 93.44 %		Function Coverage 74.28 %		Region Coverage 68.10 %	
bangk/src/instruction.rs	96.95 %	540 / 557	72.58 %	45 / 62	63.13 %	113 / 179
bangk/src/invest/dividends.rs	98.88 %	177 / 179	100.00 %	3 / 3	72.78 %	115 / 158
bangk/src/invest/investment.rs	96.84 %	153 / 158	100.00 %	2 / 2	70.83 %	102 / 144
bangk/src/invest/mod.rs	100.00 %	21 / 21	100.00 %	1 / 1	66.67 %	10 / 15
bangk/src/invest/project.rs	96.82 %	213 / 220	100.00 %	5 / 5	69.12 %	150 / 217
bangk/src/invest/transfer.rs	98.95 %	189 / 191	100.00 %	2 / 2	71.59 %	126 / 176
bangk/src/processor.rs	98.99 %	98 / 99	100.00 %	3 / 3	83.33 %	60 / 72
bangk/src/stable/burn.rs	100.00 %	21 / 21	100.00 %	1 / 1	75.00 %	15 / 20
bangk/src/stable/exchange.rs	100.00 %	38 / 38	100.00 %	1 / 1	74.36 %	29 / 39
bangk/src/stable/mint.rs	100.00 %	32 / 32	100.00 %	1 / 1	75.68 %	28 / 37
bangk/src/stable/transfer.rs	100.00 %	31 / 31	100.00 %	1 / 1	77.78 %	21 / 27
bangk/src/state/clients.rs	100.00 %	44 / 44	100.00 %	9 / 9	80.00 %	12 / 15
bangk/src/state/config.rs	0.00 %	0 / 46	0.00 %	0 / 12	0.00 %	0 / 22
bangk/src/state/dividends_tracker.rs	100.00 %	34 / 34	100.00 %	9 / 9	80.00 %	12 / 15
bangk/src/state/mint_data.rs	100.00 %	21 / 21	100.00 %	3 / 3	100.00 %	4 / 4
bangk/src/state/mints.rs	93.84 %	137 / 146	77.78 %	7 / 9	66.67 %	56 / 84
bangk/src/state/mod.rs	94.74 %	54 / 57	66.67 %	6 / 9	65.38 %	17 / 26
bangk/src/state/pda.rs	96.63 %	86 / 89	91.67 %	11 / 12	71.83 %	51 / 71
bangk/src/state/projects.rs	96.30 %	52 / 54	100.00 %	6 / 6	76.92 %	10 / 13
bangk/src/state/stable.rs	89.47 %	17 / 19	66.67 %	4 / 6	53.33 %	8 / 15
bangk/src/utils/accounts.rs	98.15 %	53 / 54	100.00 %	2 / 2	69.57 %	16 / 23
bangk/src/utils/mod.rs	88.89 %	32 / 36	100.00 %	8 / 8	89.19 %	33 / 37
bangk/src/utils/tokens.rs	92.86 %	273 / 294	100.00 %	13 / 13	70.76 %	121 / 171

Figure 1 – Test coverage for Bangk Solana programs

5 Programs

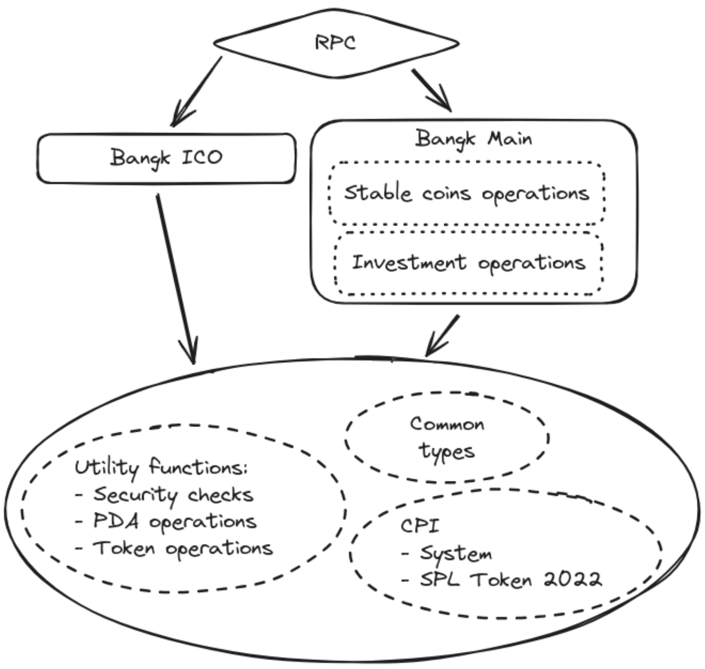


Figure 2 – General overview of Bangk 's programs

As the only way to interact with the blockchain, the programs are at the heart of Bangk 's processes.

All programs that do not require a modification of the state of the blockchain (*e.g.* query the token balance for a given account, read the investment state of a client, *etc.*) can be done through sea-level programs *via* an RPC API (*e.g.* QuickNode).

All operations performed on Bangk 's programs (or any on-chain program for that matter) expect three type of parameters:

1. One or more signatures: those are the Keypairs (Pubkey + private key) signing / authorizing the transaction. There can be more than one, but the first one given will be the one paying the transaction fees (which are proportional to the number of signatures to validate). Those accounts will be associated with the ✍symbol.
2. The accounts that will be accessed by the instruction, including any account that the instruction will create. An account that must be writable will be associated with the symbol 🖋. When those accounts are associated to a token, its type will be indicated next to it.
3. The parameters for the programs - serialized by Borsh. At a minimum, there will be an unsigned 8 bits integer defining the type of operation (*i.e.* the instruction) to perform.

One important thing to note: the way the Solana blockchain has been designed, transaction parameters cannot be of more than 1232 bytes all included. Considering that signatures take 64 bytes and addresses (so accounts) take 32 bytes, that places a hard limit on the number of instructions that can be done with a single transaction. While there exists a trick in the form of the Address Lookup Table [15] that can store up to 255 account addresses, the limit still exists. While this doesn't impact the program itself, some operations (*e.g.* payment of dividends) will need to be done by batch.

Transaction outputs are a tricky thing. In theory, the only thing an instruction can output is a flag signaling the success of the operation or its failure associated with an error code defining the type of error. There are however two other options:

- Logs: all programs can - and do - output logs. In theory, this allows tracing the behavior of the program, display some important information, *etc.*In practice however, logs formatting is computationally expensive, so those logs are limited.
- Each transaction owns a buffer for data passed from a program back to its caller, which can be used to write some data in the transaction state. This is limited, and needs to be decoded with Borsh, but it can be useful for some operations [16].

While there would be arguments both ways regarding splitting Bangk 's operations into different programs or have it all in one, we have decided to work with different programs where the operations can be isolated from the others. For example, BGK operations do not necessitate the use of Stable Coins or Security tokens, so they can be safely given their own programs. On the other hand, Security tokens operations make heavy use of the Stable Coins, so they both share the same program (doing otherwise would mean duplicating some security checks for example, which would be an unnecessary computation cost).

5.1 Bangk Coins

5.1.1 Initialization

This instruction can only be called once following the initial deployment of the program. It will set up the program's configuration PDA which contains the definition of the unvesting scheme as well as the initial set of keys allowed to sign routine, sensitive and critical operations.

A single signer is expected, the associated public key will be hard-coded in the program itself to ensure no one else but Bangk will be able to initialize the program.

Security checks performed:

- check_signers

The definition for each category of the unvesting plan is:

Table 3 – Unvesting plan category definition

Value	Type
ID	u8
Unvest delay after launch (days)	u16
Initial unvest percentage (factor x1000)	u16
Weekly unvest percentage (factor x1000)	u16
Final unvest percentage (factor x1000)	u16

Total size: 15 bytes. Compute Unit consumed: 44'069.

15. `https://docs.solana.com/de/developing/lookup-tables`
16. `https://docs.rs/solana-sdk/latest/solana_sdk/program/fn.set_return_data.html`

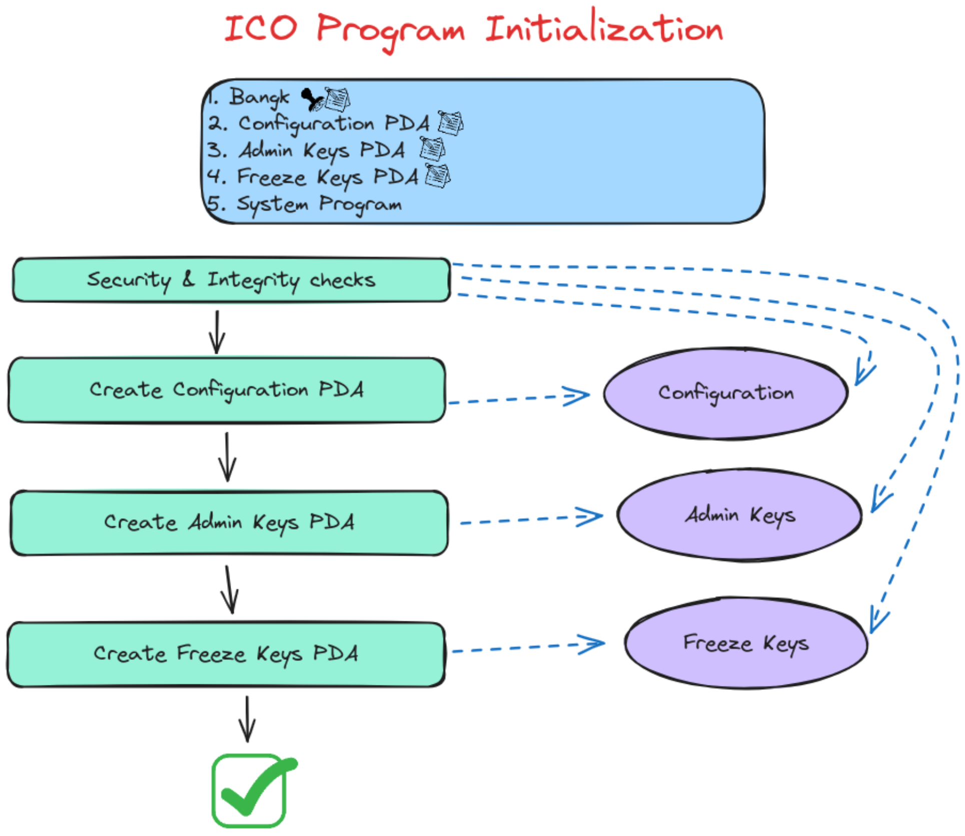


Figure 3 – BGK Program Initialization

Table 4 – Inputs for program initialization

	Value	Type
Accounts	Signer	Keypair
	Configuration PDA	Pubkey
	Admin Keys PDA	Pubkey
	System Program	Pubkey
Data	Instruction ID: 0	u8
	Unvest Team & Founder configuration	Unvest config
	Unvest Partners & Advisors	Unvest config
	Unvest private sales	Unvest config
	Unvest public sales 1	Unvest config
	Unvest public sales 2	Unvest config
	Unvest public sales 3	Unvest config
	Config PDA bump	u8
	API key	Pubkey
	Admin 1	Pubkey
	Admin 2	Pubkey
	Admin 3	Pubkey
	Admin 4	Pubkey

Total size: 509 bytes. Compute Units used: 44'069.

5.1.2 Mint creation and token minting

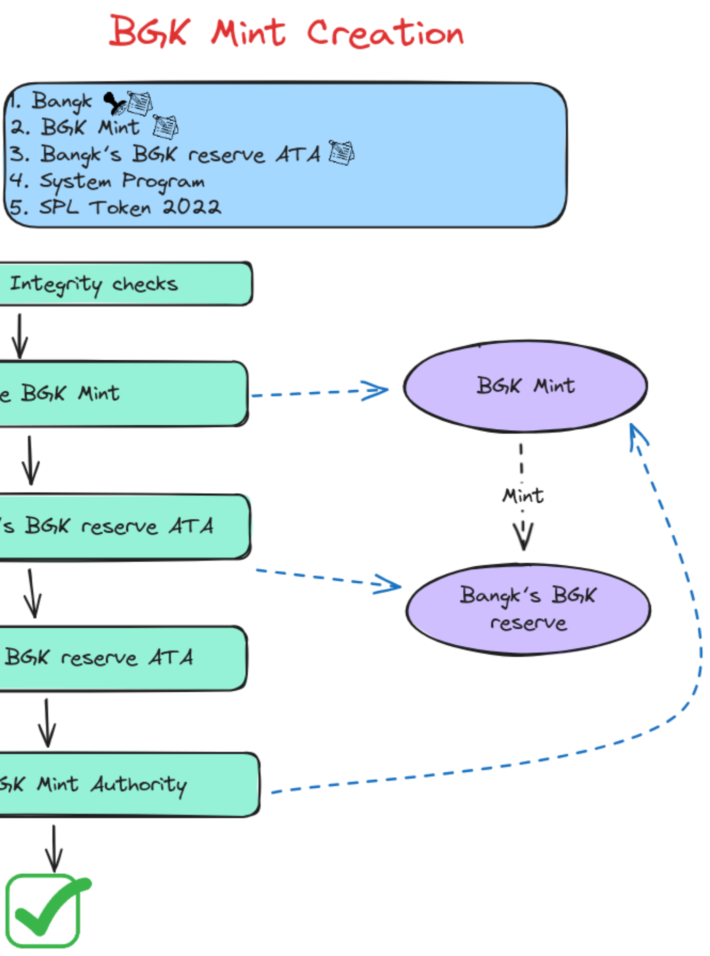

BGK Mint Creation
1. Bangk
2. BGK Mint
3. Bangk's BGK reserve ATA
4. System Program
5. SPL Token 2022
Security & Integrity checks
Create BGK Mint
Create Bangk's BGK reserve ATA
Mint 177m to BGK reserve ATA
Remove BGK Mint Authority
BGK Mint
Mint
Bangk's BGK reserve

Figure 4 – BGK Mint Creation & Minting

This instruction will:

1. Create the BGK mint,
2. Create Bangk 's reserve ATA, giving ownership to Bangk 's Admin Keys PDA,
3. Mint all 177m tokens to Bangk 's reserve ATA
4. Revoke the mint authority.

🪪 While a critical operation, this can only be done once and never again. As such, we will only require one signer, the same as the one who initialized the program (and thus whose pubkey is hard-coded in the program).

🔒 Security checks performed:

- check_signers

Table 5 – Inputs for mint creation & initial token minting

	Value	Type
Accounts	Admin 1	Keypair
	Admin 2	Keypair
	Admin 3	Keypair
	Admin Keys PDA	Pubkey
	BGK mint	Pubkey
	Bangk 's BGK ATA	Pubkey
	BGK Reserve ATA	Pubkey
	System Program	Pubkey
	SPL Token 2022 Program	Pubkey
Data	Instruction ID: 1	u8
	Mint Bump	u8

Total size: 418 bytes. Compute Unit consumed: 109'275.

5.1.3 Update admin keys

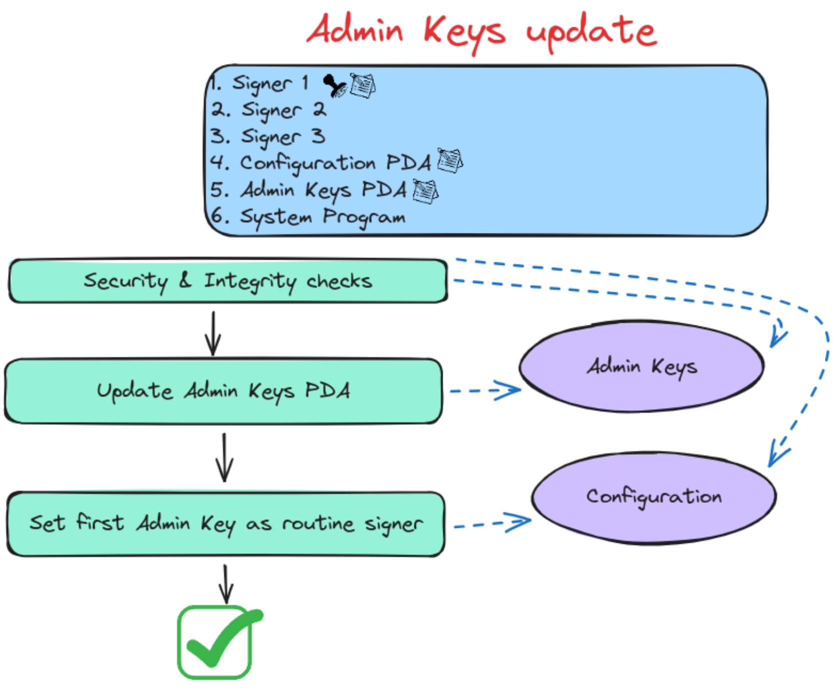


Figure 5 – BGK Program Admin Keys update

Should one of the keys allowed to perform critical tasks get compromised, this operation will ensure that it can be replaced by a new one.

Obviously a critical operation, this will require three authorized signers to perform the instruction.

Security checks performed:

- check_signers
- check_pda_owner for the configuration and admin keys PDA

Table 6 – Inputs for program configuration update

	Value	Type
Accounts	Admin 1	Keypair
	Admin 2	Keypair
	Admin 3	Keypair
	Admin Keys PDA	Pubkey
	System Program	Pubkey
Data	Instruction ID: 2	u8
	API key	Pubkey
	Admin 1	Pubkey
	Admin 2	Pubkey
	Admin 3	Pubkey
	Admin 4	Pubkey

Total size: 417 bytes. Compute Unit consumed: 10’118.

5.1.4 Investment

This routine operations materializes on the blockchain a user’s investment during the ICO. This instruction will do so by creating (or updating) a PDA containing details about all the user’s investments.

As a routine operation, this instruction will need to be signed only by the API key.

Security checks performed:

- check_signers
- check_pda_owner for the configuration and investment PDAs

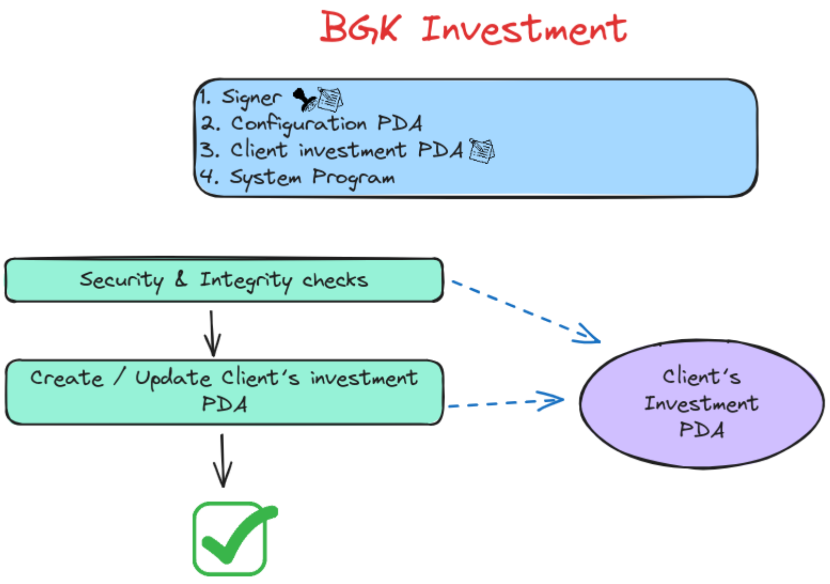


Figure 6 – Client's investment

Table 7 – Inputs for client investment

	Value	Type
Accounts	API key	Keypair
	Configuration PDA	Pubkey
	Admin MultiSig PDA	Pubkey
	Client investment PDA	Pubkey
	System Program	Pubkey
Data	Instruction ID: 3	u8
	Investment PDA bump	u8
	User wallet	Pubkey
	Investment type	u8
	Custom Unvesting	Unvest Config
	Number of tokens	u64

Total size: 250 bytes. Compute Unit consumed: 29'000.

5.1.5 Cancel

Even if it *should* not happen, it's always possible that we are required to cancel a user's investment. This is restricted to the period before the unvesting starts. It simply deletes the user's investment PDA.

While not a critical operation, this is not a routine one either. As such, it will require two signers.

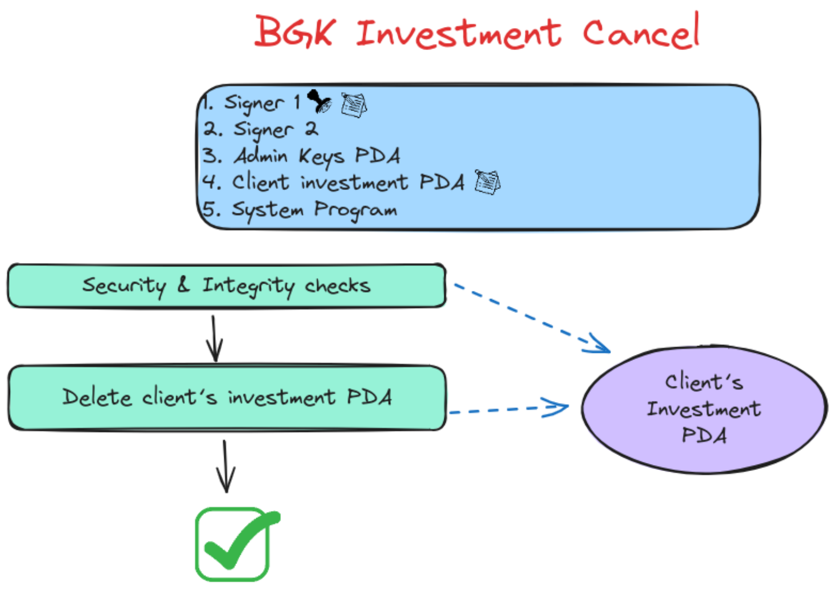


Figure 7 – Client's investment cancellation

🔒 Security checks performed:

- check_signers
- check_pda_owner for the configuration, admin keys and investment PDAs
- check_user to make sure the PDA to delete does belong to the user for whom we want to delete the investment.

Table 8 – Inputs for investment cancellation

	Value	Type
Accounts	Admin 1	Keypair
	Admin 2	Keypair
	Configuration PDA	Pubkey
	Admin MultiSig PDA	Pubkey
	Client invest account	Pubkey
	System Program	Pubkey
Data	Instruction ID: 4	u8
	User wallet	Pubkey

Total size: 289 bytes. Compute Unit consumed: 24'497.

5.1.6 Set launch date

This instruction starts by setting the BGK token's launch date in the configuration PDA. Then, it will create the investment ATA account where all tokens that are to be unvested will be stored before being transferred to the users.

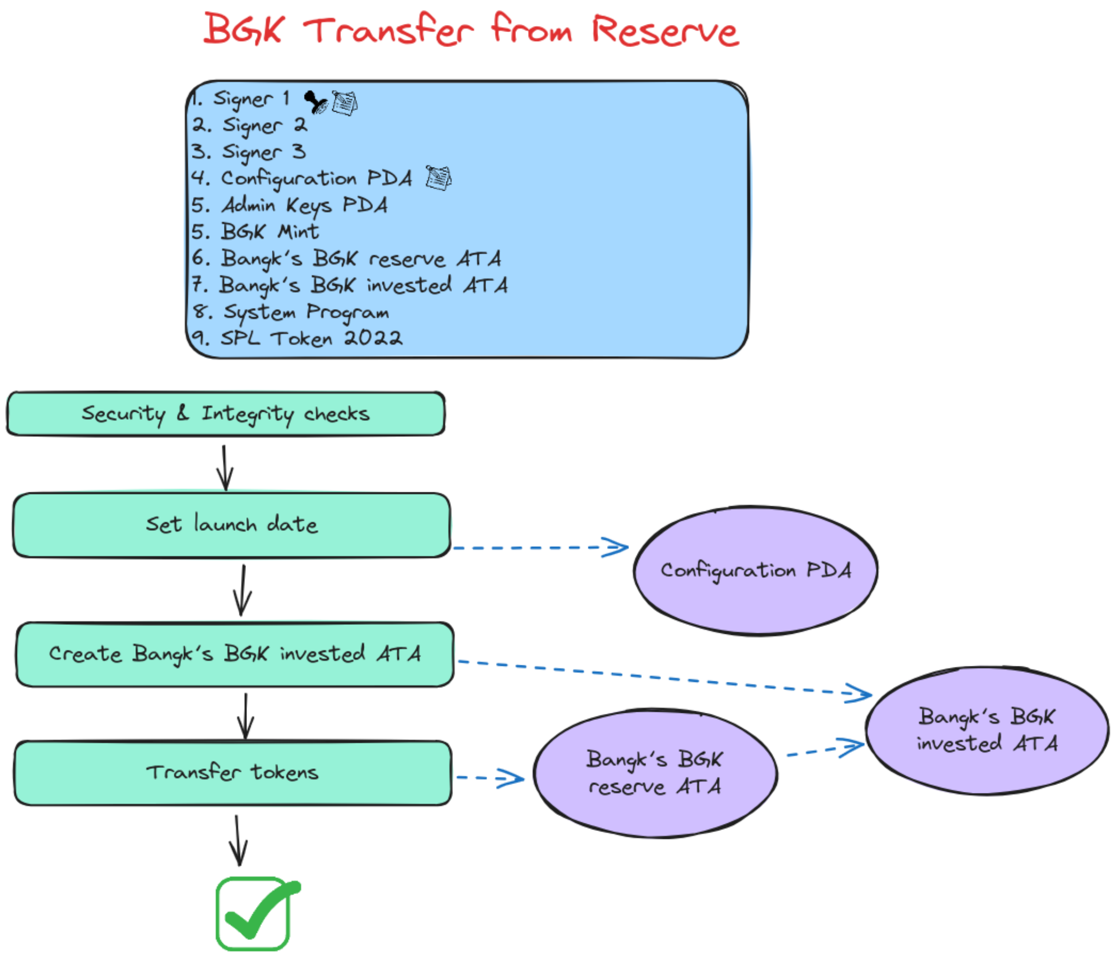


Figure 8 – BGK Launch

🪪 Obviously a critical operation here as all the unvesting depends on it: it will require three authorized signers.

🔒 Security checks performed:

- check_signers
- check_pda_owner for the configuration and admin keys PDAs

Table 9 – Inputs for BGK launch

	Value	Type
Accounts	Admin 1 ✍🖋	Keypair
	Admin 2 ✍	Keypair
	Admin 3 ✍	Keypair
	Configuration PDA 🖋	Pubkey
	Admin MultiSig PDA	Pubkey
	BGK Mint	Pubkey
	Bangk 's reserve BGK ATA 🖋	Pubkey
	Bangk 's BGK investment pool ATA 🖋	Pubkey
	System Program	Pubkey
	SPL Token 2022 Program	Pubkey
	Associated Token Account Program	Pubkey
Data	Instruction ID: 5	u8
	Release date	Timestamp (i64)
	Number of tokens to release	u64

Total size: 465 bytes. Compute Unit consumed: 73'490.

5.1.7 Vesting release

This instruction releases available unvested account to a user's ATA. It will start by checking *for each investment type* how many tokens the user is allowed to own at the time the instruction is executed and how many have already been released to him, before transferring the difference from Bangk 's investment ATA to the user's ATA. The check should *not* be done based on the number of tokens currently in the user's ATA.

🪪 This operation is a routine one, so only one signer will be required.

🔒 Security checks performed:

- check_signers
- check_pda_owner for the configuration and investment PDAs
- check_user to make sure the PDA belongs to the user owning the ATA where the tokens will be released.

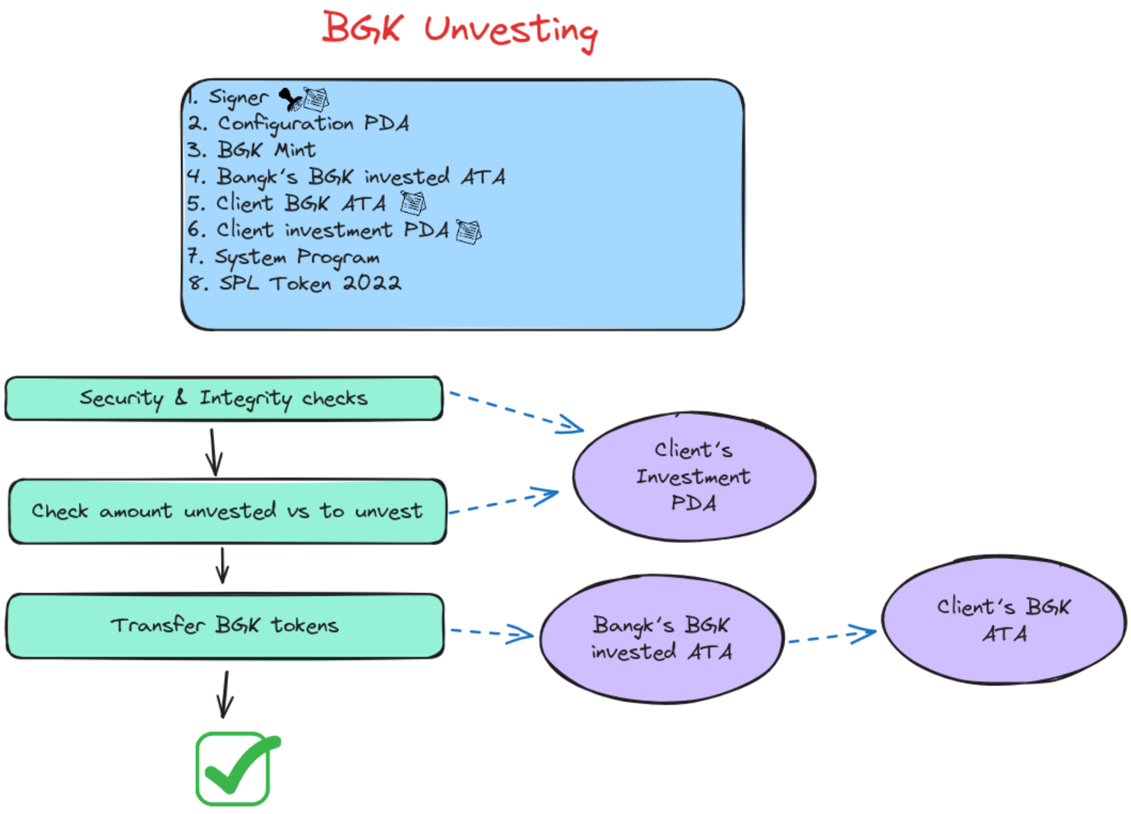


Figure 9 – BGK Unvesting

Table 10 – Inputs for BGK Release

	Value	Type
Accounts	Payer ✍	Keypair
	Configuration PDA	Pubkey
	BGK mint	Pubkey
	BGK investment pool ATA ✒	Pubkey
	Bangk 's BGK investment pool ATA ✒	Pubkey
	User wallet	Pubkey
	Client's investment PDA ✒	Pubkey
	Client's BGK ATA ✒	Pubkey
	System Program	Pubkey
	SPL Token 2022 Program	Pubkey
	Associated Token Account Program	Pubkey
Data	Instruction ID: 6	u8
	User wallet	Pubkey

Total size: 417 bytes. Compute Unit consumed: 61'000 (heavily depends on the number of different unvesting rules to check, and if there are tokens to unvest or not).

5.1.8 Transfer from reserve

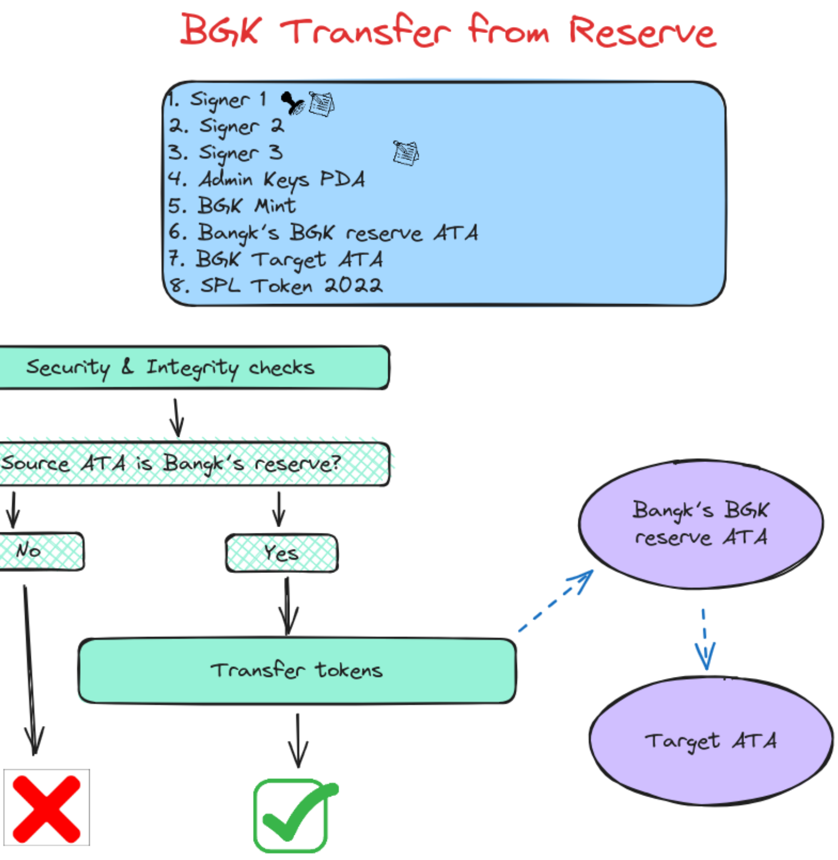


Figure 10 – BGK Transfer from Reserve

The standard transfer instruction (which is a direct call to the SPL Token 2022 program) is not to be used to transfer tokens away from Bangk 's reserve ATA since it's considered a critical operation and thus can only be accessed through Bangk 's MultiSig. In order to transfer tokens from that account, this instruction must be used.

🪪 Considered a critical operation, this instruction will need three authorized signatures.

🔒 Security checks performed:

- check_signers
- check_pda_owner for the configuration and admin keys PDA

Table 11 – Inputs for Bangk 's reserve BGK tokens transfer

	Value	Type
Accounts	Admin 1 ✍ 🖋	Keypair
	Admin 2 ✍	Keypair
	Admin 3 ✍	Keypair
	Admin MultiSig PDA	Pubkey
	BGK mint	Pubkey
	Bangk 's BGK reserve ATA source 🖋	Pubkey
	User Wallet	Pubkey
	BGK tokens ATA target 🖋	Pubkey
	System Program	Pubkey
	SPL Token 2022 Program	Pubkey
	Associated Token Account Program	Pubkey
Data	Instruction ID: 7	u8
	Number of tokens	u64

Total size: 457 bytes. Compute Unit consumed: 21'913 (if target ATA exists) or 51'232 (otherwise).