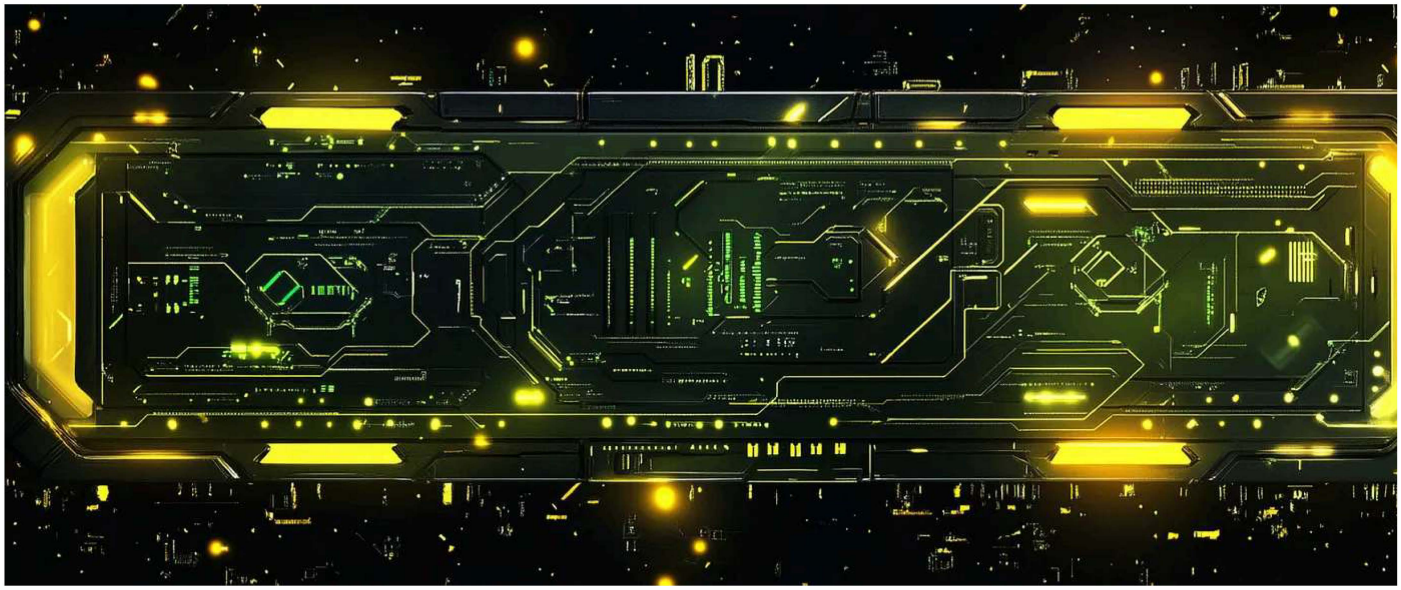
BORG OS – Under the Hood

The heart of the machine.....

GABRIEL SHAPIRO
DEC 10, 2024



The heart of the machine.....

Today MetaLeX is proud to present version 1 of BORG OS, a full software and legal suite designed to manufacture and design custom BORGs that are tailor made to create a full separation of powers between the real and digital world.

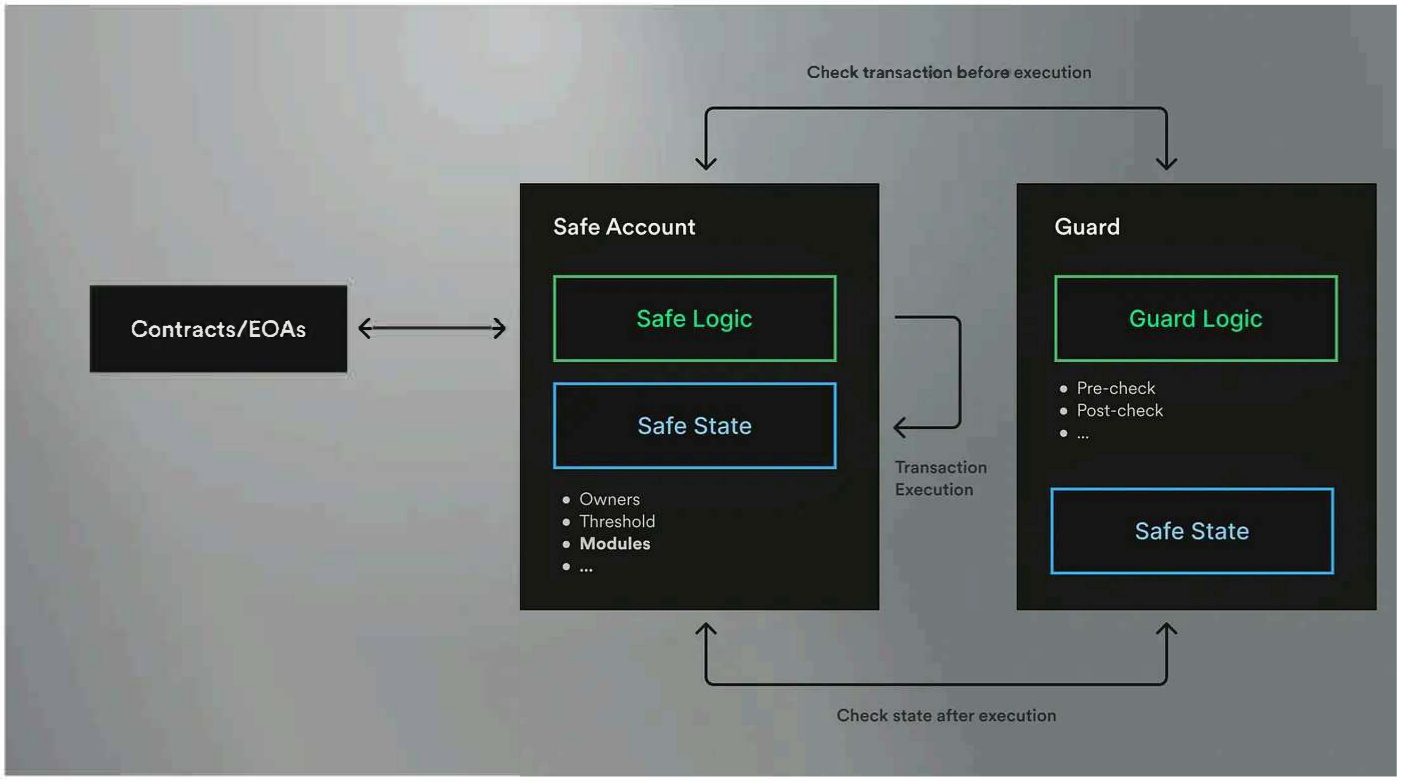
BORG OS v.1 provides a powerful and customizable system for establishing, monitoring, and operating governance-accountable, trust-minimized, transaction-automated and legally optimized multisig smart contracts. This article breaks down BORG OS's architecture and key features. Because the protocol is designed primarily for use with cy**B**ernetic **ORG**anizations—arrangements in which a multisig is

'wrapped' by a legal entity and specific legal contract terms govern the use of that multisig—we generally refer to the multisigs operated through BORG OS as 'BORG

Background – SAFEs

BORG OS is built around the standard SAFE protocol (among the most secure, batt tested and audited protocols in all of crypto), leveraging the two important hooks fc extending that code: "Guards" and "Modules".

Guards.

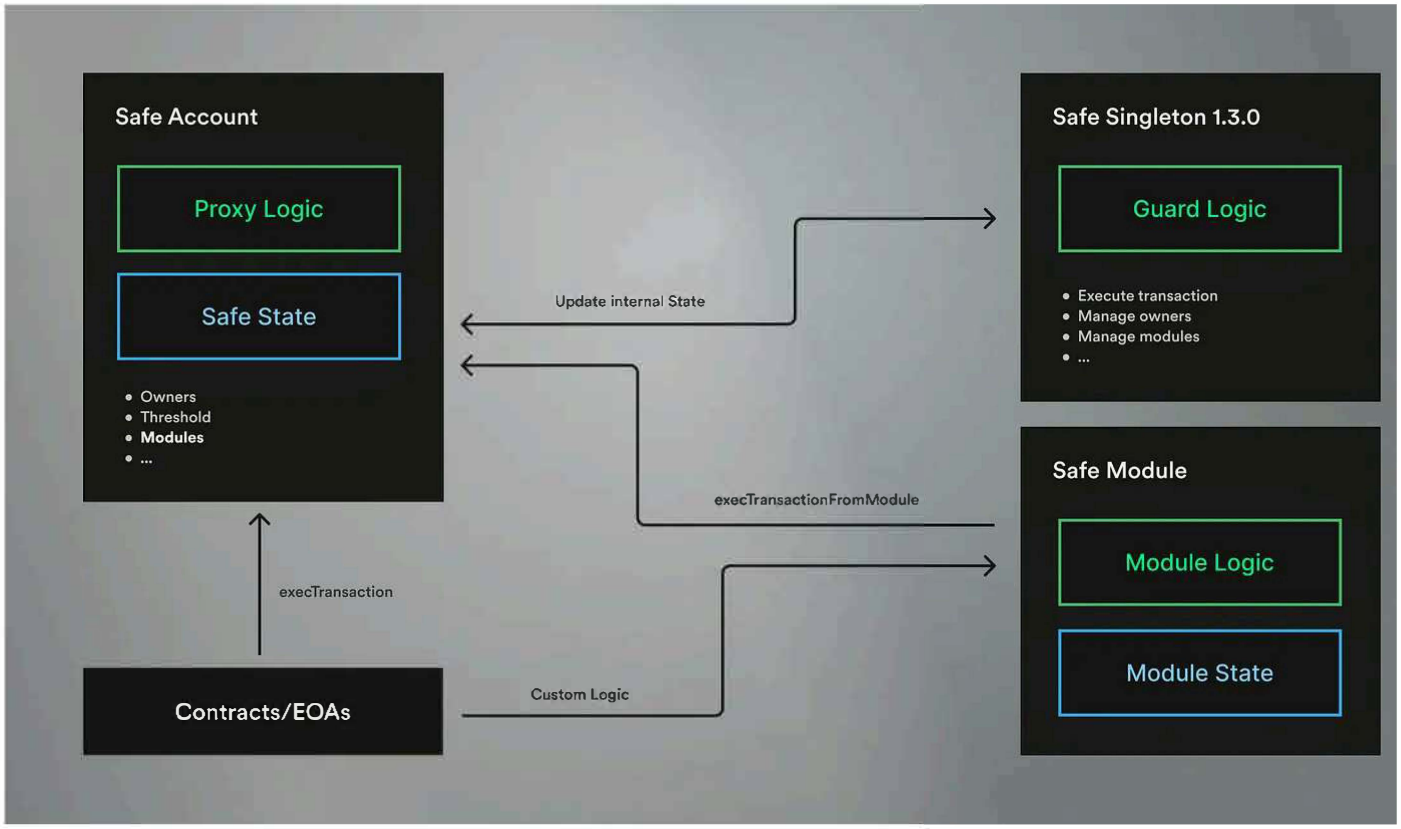


SAFE Guards are smart contracts that constrain the functioning of an otherwise standard SAFE, providing controlled access to specific recipients and contracts. Th is accomplished by imposing pre-transaction-checks and post-transaction-checks c the SAFE. Guards 'safeguard the SAFE,' as it were. If the checks are not satisfied by given transaction, the transaction will not execute (*aka* 'reversion').

A Guard is added to a SAFE by calling the setGuard() function of the SAFE with the requisite majority or plurality of signatures, parametrized to the smart contract address of the Guard. A SAFE can manage multiple Guards through an intermediar GuardManager contract. A typical use-case would be only allowing 'owners' of the SAFE (denominated by address) to execute a transaction (useful forMEV protection to only allow the SAFE to interact with certain whitelisted smart contracts via certa whitelisted functions on those smart contracts (useful to prevent accidental transactions or to limit the use of funds).

Guards constrain SAFEs.

Modules.



Modules are smart contracts that extend or modify the functioning of an otherwise standard SAFE. They can execute transactions on the SAFE that have not been individually approved by the requisite majority or plurality of signatures. A Module

added to a SAFE by calling the enableModule() function of the SAFE with the requi majority or plurality of signatures, parameterized to the smart contract address of t Module. A SAFE can manage multiple Modules through a ModuleManager contrac Typical use-cases would be giving another account a per-month 'allowance' it can spend out of the SAFE's assets, timelocking an upgrade authority that may be exercised by the SAFE over a DeFi protocol, or facilitating Moloch-style 'ragequit' functionality where a signer can quit the multisig with a proportional share of its assets.

Modules expand SAFEs.

BORGcore - The Heart of all BORGs

Every BORG on MetaLeX OS has a BORGcore (https://github.com/MetaLex-Tech/borg-core) smart contract as its heart. A BORGcore is a SAFE Guard combine with a standard ERC 4824 DAO interface.

BORG Modes

BORGcores have three modes: whitelist, blacklist, and unrestricted, each with uniq access rules and constraints, and each capable of supporting BORG implants (*see below*) and tracking the URIs of the legal documents that 'wrap' the BORG. For

security reasons, modes are immutable–thus, each BORG's mode is selected once a for all, at the time of deployment of the BORGcore contract.

Whitelist Mode

- In whitelist mode, all transactions are ***prohibited***–except those explicitly pre-permitted (i.e., whitelisted).
- Only approved recipients can receive native gas transfers, subject to individual transaction limits.
- Only whitelisted contracts can be interacted with, and each whitelisted contra methods can have specific parameter constraints (including types (uint, int, address, string, bytes, bool), value ranges, and exact matches).

Whitelist mode is the most conservative and trust-minimized mode for DAO-adjac BORGs. It enforces the rule that BORG transactions are *unauthorized by default.* On whitelisted transactions (or transaction types) with whitelisted accounts (or accoun types) are allowed—allowances that may be thought of as enforcing the BORG's 'policies.' In most recommended BORG configurations, these policies will be mandatory legal rules enforced by, or enforceable against, the legal entity 'wrapper' and its personnel.

As a result, the BORG can only transact with counterparties it's supposed to transa with, in ways, amounts, and at times, that are consistent with the legal rules of the legal entity 'wrapper'. In the case of a DAO-adjacent BORG, these legal rules will, i turn, reflect the expectations of, and be approved in advance by, the adjacent DAO. the case of a standalone BORG, these legal rules will reflect the expectations of, an approved in advance by, other classes of interested parties, such as stockholders, managers, a board of directors, etc.

Thus, in wishlist mode, the BORGcore creates a 'can't-be-evil' implementation of multisigs to constrain legal entities and their agents as a part of the core trust-minimization and 'can't-be-evil' ethos of crypto.

This BORG mode is most suitable for BORGs handling large amounts of money–fo example, a finBORG that is managing 'protocol-owned' or 'protocol-beneficial' valu by moving liquidity among various whitelisted liquidity pools and DeFi protocols. I may also be used for a Grants BORG that is subject to strict rate-limits or DAO vet or DAO co-approvals.

Blacklist Mode

- In blacklist mode, all transactions are ***permitted***–except those explicitly pre-prohibited (i.e., blacklisted).
- Recipients listed in the blacklist cannot receive native gas transfers.
- Blacklisted contracts cannot be interacted with, or methods of these contracts blocked unless they pass specified parameter constraints (including types (uint int, address, string, bytes, bool), value ranges, and exact matches).

Blacklist mode is suitable for DAO-adjacent BORGs that are somewhat more truste (or somewhat less risky, and therefore less trust-requiring) and therefore may have broad discretion and flexibility, but with respect to which the DAO wishes to constrain a short list of particularly risky, dangerous or prohibited transactions. Fo example, a group of friends managing a memecoin and is trusted to use presale proceeds well, but which requires a snapshot approval of the memecoin holders in order to increase the memecoin supply through a minting function on the token sm contract. In this case, the mint() function on that particular token contract would b blacklisted, and implants (explained below) would be used to only allow a mint() cal also approved by the required snapshot vote.

In both whitelist and blacklist mode, the BORG can be subject to cooldown period for relevant method calls to prevent the equivalent of DoS attacks. For example, a cooldown can prevent a BORG from spamming a DAO with vetoable transactions i order to raise the DAO's monitoring/voting costs in the hope of sneaking a transact through that the DAO would otherwise be likely to veto. This complements

recommended provisions in the BORG's legal docs that also prohibit such abusive activities.

Unrestricted Mode

- In unrestricted mode, all transactions and interactions are allowed without restrictions.
- No constraints are applied to recipients or contracts.
- All native gas transfers and contract interactions are permitted.

Unrestricted mode is suitable for highly 'trusted' BORGs where the multisig signer are intended to have full discretion over the BORG's activity, but where there is a desire to use BORG OS's implant functionality (*see* below) and/or simply to use MetaLeX's web interface and legal services to wrap an otherwise unmodified SAFE For example, this could enable a BORG whose main difference from an ordinary SA is just that, unlike in a standard SAFE< signers can now resign instantly and unilaterally–*see below* under 'ejectImplant.sol'.

BORGcore also manages access control via the `BorgAuth` contract, ensuring that only authorized addresses can modify the BORG's smart contracts. For DAO-adjace BORGs, this will typically be configured in one of three ways:

- BorgAuth is set to a null address–i.e., it is immutable.
- BorgAuth is set to the DAO (effectively, this makes the BORG into more of a subDAO-style arrangement, which is not legally recommended, but is possible)
- BorgAuth is set to a custom contract that requires co-approval of the DAO and BORG for changes.

The third option (approval of DAO and BORG) is the most recommended and typic and would mirror provisions in the BORG's alegal agreements that require DAO

approval for material amendments to the legal terms of the legal entity's governanc rules.

Directors & Guardians

BORGcore has the option to set "directors" of the BORG/multisig. This is meant to accommodate dual-class voting structures within a given BORG/multisig. An exam would be a 4/8 multisig with membership consisting of 3 persons who constitute th Board of Directors of the BORG and 5 persons who are not on the Board of Directo but are signers on the multisig for security reasons (essentially, increasing the num of keys and therefore raising cost of a coordinated wrench attack etc.). In this circumstance, we would require that at least a majority of the directors approve eve transaction, in addition to between one and two non-directors (which we refer to in the legal docs as "guardians"). This type of structure can also lower the administrative/process overhead for forming the legal entity, as directors have grea authority, more potential legal responsibility and therefore face higher KYC/AML/d diligence requirements than mere security contractors. The parameter is set via the 'directorsRequired' variable specifying a minimum number of directors that must approve every transaction.

BORG Implants - SAFE Customization Modules

Each BORG has cybernetic implants (https://github.com/MetaLex-Tech/borg-core/tree/main/src/implants) enhancing its abilities beyond those of a normal SAFE. Each Implant is a SAFE Module consisting of a ConditionManager, a BORGauth, a a set of more specific rules particular to that implant.

The BORGauth may, through a particular Implant, grant to some third party or extrinsic smart contract the authority to take actions such as vetoing a timelocked BORG transaction, revoking the BORG's funds, or adding and removing the BORG signers. For a DAO-adjacent BORG, the BORGauth would grant authorities over th BORG to the adjacent DAO. For a standalone BORG, the BORGauth would grant

authorities over the BORG to stockholders (represented through a voting contract c tokenized shares of stock) or a set of managers, board of directors, or similar body represented through another SAFE multisig.

The ConditionManager allows for the BORGauth's authority to be programmatical modulated. Custom smart contracts can be added to handle any conditional logic required, including: signature/multiparty approval, time, balance, and external orac inputs. This can accommodate any condition that is provable on chain or with a trusted oracle for unlocking milestones or releasing escrowed funds.

ConditionManager enables the relationship between the BORG and its adjacent authority to be *tuned* into a nuanced governmental 'checks/balances' dynamic rathe than the BORG being simplistically controlled by or subservient to the authority. F DAO-adjacent BORG, such as a Grants BORG, this might mean, for example, that t BORG can freely give grants up to some monthly cap (denominated in either dollar tokens), but that the DAO must co-approve, or can veto, grants exceeding that cap. a standalone BORG, such as a technology corporation, this might mean, for exampl that the BORG's signers are also the directors of the entity, that a smart contract fo recording votes of the tokenized shares is the BORGauth, and that each director se for a specific term-of-service (potentially 'staggered' against the terms of other directors) and is automatically removed if not reelected by tokenized share vote wit 30 days of the end of the term.

This architecture and its intended uses are a key differentiator between BORG OS a other DAO and SAFE meta-protocols. Many such protocols conceptualize DAO-adjacent multisigs as being "subDAOs," "minions," "avatars," "squads" or similar constructs to be treated much like subsidiaries, agents, or representatives of a DAO Still others treat such multisigs as being 'councils' that 'manage' the DAO via 'delegation.' While BORG OS can certainly facilitate those same kinds of relationsh the BORG design-philosophy encourages, and BORG OS facilitates, a more subtle check-and-balance dynamic between DAOs and BORGs—making them mostly

autonomous from each other while still being mutually accountable. This is critical implementing optimal legal strategies*.

*(*See* https://www.law.cornell.edu/wex/alter_ego. Such separation is also important from a tax perspective).

The first BORG-type fully implemented on BORG OS, GrantsBORGs, can feature t following Implants:

OptimisticGrantImplant.sol

- Enables the GrantsBORG to give out grants "optimistically" using funds grant by the adjacent DAO, subject to programmatic rate limitations and caps. The GrantsBORG may exceed the rate limitations and/or caps with DAO-co-approv or subject to a timelock + absence of DAO veto. In an ideal case, the legal entit wrapper would still constrain the optimistic grants to be within DAO-approve purposes (such as supporting the specific ecosystem involved) and to follow certain offchain rules (like not issuing a grant to the GrantsBORG personnel o their respective affiliates).

daoVETOGrantImplant.sol and vetoImplant.sol

- Enables the aforementioned timelock + DAO veto pattern for exceptions to the GrantsBORG's rate limits and caps. This also includes anti-DoS measures so t the BORG cannot overwhelm the DAO's veto capacity via 'spam' proposals (wh should ideally be complemented with anti-evasion legal rules in the BORG leg wrapper). Can also be configured to require pre-approval per token type and ha a per-grant limit. The two contracts accomplish this for full DAOs and snapsh style DAOs, respectively.

daoVoteGrantImplant.sol

- Enables the aforementioned DAO co-approval pattern for exceptions to the GrantsBORG's rate limits and caps.

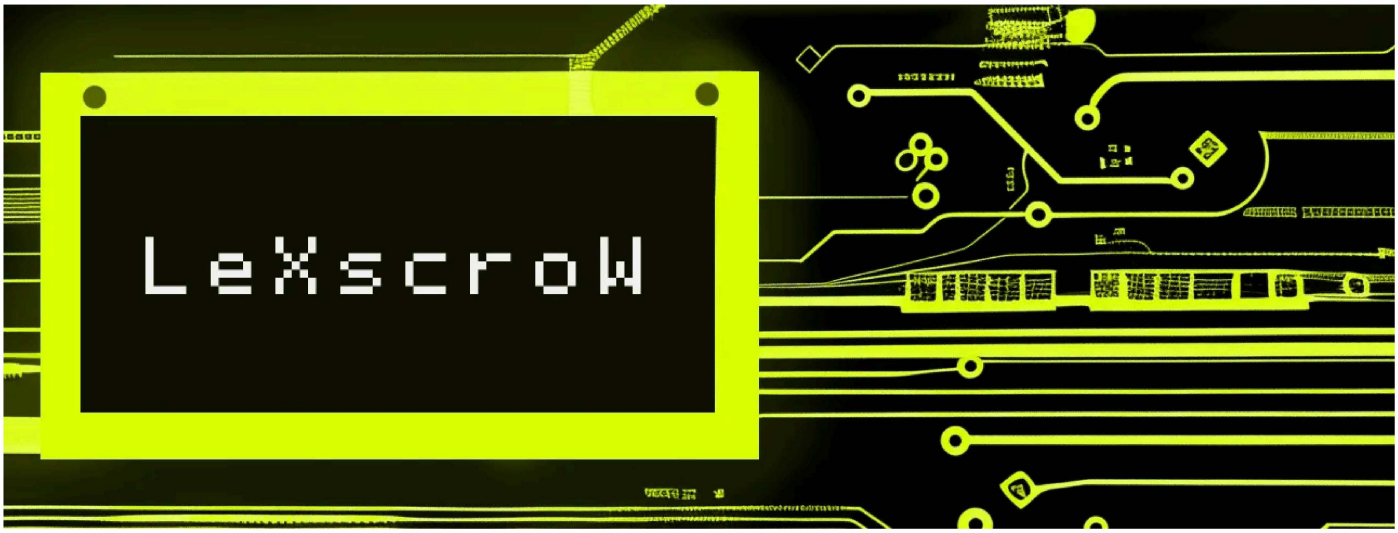
ejectImplant.sol

- Enables a GrantsBORG multisig signer to resign of their own accord or for the DAO to remove a GrantsBORG multisig signer. When combined with DAO control of a MetaVest instance used to pay GrantsBORG personnel, the resignation mechanic can be a powerful way for the DAO to exert *influence* ove the BORG without actually having shareholder-like authority to add and remov BORG members. *See* "MetaVest" and "Checks & Balances" below.

failSafe.sol

- Enables funds to revert to the DAO (or another address) on specified events—fo example, if the number of BORG signers falls below the minimum threshold needed for approval of actions on the SAFE. The destination address (typically DAO treasury) is set at deployment and immutable. Like with other Implants, t can be combined with the ConditionManager to allow for clawbacks triggered DAO approval alone, or DAO approval + number of other conditions or approv This can lead to some interesting strategies for responding to adverse events— "MetaVest" and "Game Theory for BORGs" below. However, DAO communitie and related BORGs should be careful to understand the risks of a DAO maintaining a ***diversified treasury***, such as falling under commodity pool or investment company regulations; therefore, this should be treated strictly as a security-of-funds measure and, if triggered, the DAO community should rally t create a replacement BORG to hold non-native assets that have reverted from t (likely defunct) BORG to the DAO.

LeXscroW - Cybernetic Escrows



LeXscroW (https://github.com/MetaLex-Tech/LeXscroW) is a critical component of BORG OS, designed to provide immutable, non-custodial, and flexibly-conditioned smart contract escrows. These escrows are built with the BORG OS team's experier with real-world deals in mind, and the uses of escrow agents within them. LeXscro enhances the BORG OS ecosystem by ensuring secure, automated, and legally optimized transactions involving BORGs—enforcing "deal logic" that is ordinarily entrusted to verbose legal agreements and manual processes, to the blockchain instead.

The core features common to all LeXscroWs include:

- *Ownerless Deployment*: Contracts are deployed without an 'owner,' ensuring that single entity can alter the conditions once the contract is active.
- *Immutable Conditions*: Execution conditions can include signatures, time constraints, oracle-fed data, and more. These conditions are immutable upon deployment, trust-minimizing enforcement of the deal logic.
- *Depositor Flexibility*: Contracts can specify depositing parties or allow any addre to deposit, supporting both negotiated agreements with specific prearranged counterparties and open offers (including with an option for the offeror to reje would-be accepting party, on a *post hoc* basis).

LeXscroW offers various smart escrow contract types, each tailored to specific transaction needs.

1. *DoubleTokenLexscroW*

- Bilateral Transactions: Designed for bilateral smart escrow transactions involv two different ERC20 tokens.
- Execution Conditions: The contract executes and releases tokens to both partie all conditions are met before expiration.

2. *TokenLexscroW*

- Unilateral Transactions: Facilitates unilateral smart escrow transactions for a single ERC20 token.
- Execution and Refunds: Tokens are released to the seller upon meeting conditi or refunded to the buyer if the contract expires without execution.
- Depositor Rejection: Sellers can reject depositors, triggering a withdrawal mechanism for rejected deposits.

3. *EthLexscroW*

- Native Token Transactions: Handles escrow for native gas tokens (e.g., ETH) w similar conditions and functionalities as TokenLexscroW.

LeXscroW integrates seamlessly with the BORG architecture within BORG OS, enhancing the trust-minimized, governance-accountable nature of BORG transactions. Each BORG on BORG OS can incorporate LeXscroW contracts to manage escrowed funds, ensuring that transactions comply with both the smart contract conditions and the legal rules governing the BORG's operations. This allo BORGs to be somewhat more flexible, while also remaining trust-minimized—transactions that could not have been anticipated when the DAO initially authorize

the BORG can still be implemented in a way that automatically enforces the relevar deal logic.

For instance, a GrantsBORG could utilize LeXscroW to handle the conversion of sc of its OpsBudget governance tokens into stablecoins via a TokenLeXscroW that off to sell the governance tokens at a specified discount to market price, with a specifie lockup–essentially a unilateral OTC token sale offer.

In addition to enhancing traditional BORG functions, LeXscroW significantly enhances the "deal technology" capabilities within BORG OS, powering secure and transparent transactions between various entities:

- BORG-to-BORG Transactions: LeXscroW enables BORGs to engage in comple transactions with one another, ensuring that funds are only released when spec conditions are met. This enhances trust and operational efficiency between different BORGs.
- DAO-to-BORG Transactions: DAOs can use LeXscroW to securely fund BORG ensuring that the funds are utilized according to predefined conditions and milestones. This mechanism supports governance and accountability within th DAO, while also providing operational autonomy to the BORG.
- DAO-to-DAO Transactions: LeXscroW facilitates secure and conditional transactions between different DAOs, enabling collaborative projects and fund initiatives. By ensuring that all conditions are met before funds are released, LeXscroW fosters trust and cooperation between separate DAOs.

Essentially it digitally bakes in a ruleset by which two entities can abide by, replicat the way regular companies and organizations craft deals and assurances while interacting with each other, but in a more trust-minimized and secure manner.

Interesting Use Cases

LeXscroW can be applied in various scenarios to enhance the functionality and trustworthiness of transactions within the BORG OS ecosystem:

1. Mergers and Acquisitions (M&A) Escrows:

- Scenario: Two DAOs or 'protocols' agree to merge, and the transaction involve complex exchange of tokens and assets.
- LeXscroW Solution: A DoubleTokenLexscroW contract can be set up to handle the token swap, with conditions that ensure all DAO votes, BORG actions, and other prerequisites are met before the tokens are exchanged. This ensures that both DAOs can trust the process without needing a central intermediary.

2. Trustless Token Swaps:

- Scenario: Two BORGs or DAOs wish to swap tokens directly without relying o centralized exchanges.
- LeXscroW Solution: Using DoubleTokenLexscroW, each party deposits their respective tokens into the contract. The swap executes only when both parties have deposited the required amounts and any additional conditions (such as oracle-confirmed price feeds) are satisfied. This creates a secure and trustless environment for token exchanges.

3. Project Funding Milestones:

- Scenario: A DAO funds a project managed by a BORG, with funds released bas on project milestones.
- LeXscroW Solution: A TokenLexscroW contract can be used to hold the funds, releasing them incrementally as predefined milestones are achieved and verifie either through on-chain events or trusted oracles. This ensures that the project progressing as planned before each tranche of funding is released.

4. Cross-DAO Collaborations:

- Scenario: Multiple DAOs collaborate on a joint venture, with each contributing funds or resources.
- LeXscroW Solution: EthLexscroW or TokenLexscroW contracts can be established to manage the pooled resources, releasing funds according to agree upon milestones, conditions that reflect the progress and contributions of each DAO or layered approvals (essentially approval by a virtual multisig-of-DAOs-a BORGs). This fosters cooperation and ensures transparency in the managemen shared resources.

5. Betting:

- Scenario: Two Twitter KOLs bet $1M USDC that ETH will hit $10,000 by the e of 2024.
- LeXscroW Solution: TokenLexscroW plugs into an oracle that tracks the price ETH, and automatically releases the funds to the winner based on the oracle pr

By integrating immutable, condition-driven escrow contracts, LeXscroW ensures tl transactions are secure, transparent, and efficient, thus reinforcing the reliability a functionality of BORG OS. This integration is particularly powerful in facilitating complex, trust-minimized transactions between BORGs, DAOs, and other entities, driving forward the vision of legally optimized, governance-accountable cyBernetic ORGanizations.

MetaVesT - Advanced Trust-Minimized Vesting

'Grants' can be tricky to implement. The typical notion of a 'grant' is giving a bunch money up-front to some trusted party, pursuant to a loose agreement that says they will use that money to fund some work that the grantor values. However, in practice this often turns out to be too 'trustful'—instead, it is common for grants to be paid tranches based on the satisfaction of various milestones, or to be earned/vested over time based on the expectation of a continuous period of work, where the grantor may determine to cut that work off at any time—more like an employment or independent contractor relationship. For very large grants, it may also be necessary or desirable to put transfer restrictions on tokens even after they are fully 'earned' so that not too many tokens are getting sold on the market all at once. Grant recipients may also have various structuring requests related to their tax optimization strategies, such as making the token grant into a token purchase option or deeming the tokens fully earned-up-front, but subject to a right of repurchase at a low price if conditions are not fulfilled.

Accordingly, when building-out the GrantsBORG use-case of BORG OS, we had to give thought to a number of issues related to grantees potentially earning tokens fr

the BORG over time rather than in a lump-sum:

- how could a GrantsBORG easily implement and enforce arrangements where grantees earn tokens based on length of service, milestones, or both?
- how could a GrantsBORG easily implement and enforce token-earning arrangements for its own personnel (grants 'council' members etc.)—including earning tokens over time, unlocking tokens over time, and earning tokens base on milestones?
- how could a DAO easily implement and enforce token-earning arrangements f the GrantsBORG?
- how could the above be made compatible with typical legal arrangements and t optimization strategies?
- how could we facilitate a trust equilibrium between grantor and grantee so tha neither has to trust one another completely (i.e., no 'rugging' is possible) but th is still flexibility to change terms etc. if the right people agree?
- what other kinds of uses may BORGs have for token agreements in the future (f example, handling token *unlocking* for SAFT or Token Warrant *investors*, or handling vesting and transfer restrictions for *tokenized equity securities*) and cou we accommodate all such use cases under the same protocol?

We looked at existing token 'vesting' protocols and realized they lacked many of the features needed to accommodate a rich response to these design questions. Thus, w born MetaVesT.

MetaVesT (https://github.com/MetaLex-Tech/MetaVesT) is a BORG-compatible and legally optimizable (especially, tax-optimizable) token vesting/lockup protocol for ERC20-compatible tokens. Like other token 'vesting' protocols (Hedgey, etc.), it supports basic token allocations that are streamed to a grantee over time. However, MetaVesT also supports far more complex and sophisticated token grant

arrangements that mirror how legal token agreements are drafted in the real world. These features include:

- dual-curve token release mechanisms (supporting token grants that "vest" (i.e., *earned*) and "unlock" (i.e., are released from transfer-restrictions) at different cadences, with each curve potentially having its own separate 'cliff';
- token option award and token warrant mechanisms (supporting the setting of a exercise price at time of grant and the later payment of such exercise price in stablecoins by the grantee to purchase the 'vested' tokens—important for tax structuring when grants are given after a token is already liquid and high-price
- restricted token award mechanisms (supporting the setting of a repurchase pri at time of grant and the later payment of such repurchase price in stablecoins b the grantor to repurchase the 'unvested' tokens—important for tax structuring when grants are given when a token is pre-liquid and low-priced);
- group amendment mechanics (majority-in-value of grantees + grantor can ame token grants for everyone in the same cohort—mirrors equity incentive plan amendment mechanics and venture investor SAFT/token warrant mechanics where not all employees/investors need to agree to the amendment for it to be effectuated);
- "can't be evil" /anti-rug vesting guarantees—when so configured, the only featu of the MetaVesT that can be unilaterally changed by the grantor is (where applicable) termination of vesting (mirroring "at will" termination of service of independent contractor)—all other changes require using the consensual amendment mechanic, just as they would for legal agreements in the real world (and, of course, vested tokens cannot be rugged by the grantor);
- pass-through DAO voting (unvested and/or locked tokens can still be staked an voted in a DAO)—important for restricted token award mechanisms where grantees are supposed to be the legal owners of tokens notwithstanding continuing lockups and/or repurchase rights;

- milestones—allowing vesting to be events-based rather than passage-of-time based; and
- DAO accountability—for example, allowing a DAO to either fire a worker or strongly encourage a worker to quit by terminating the vesting of the worker's token grants.

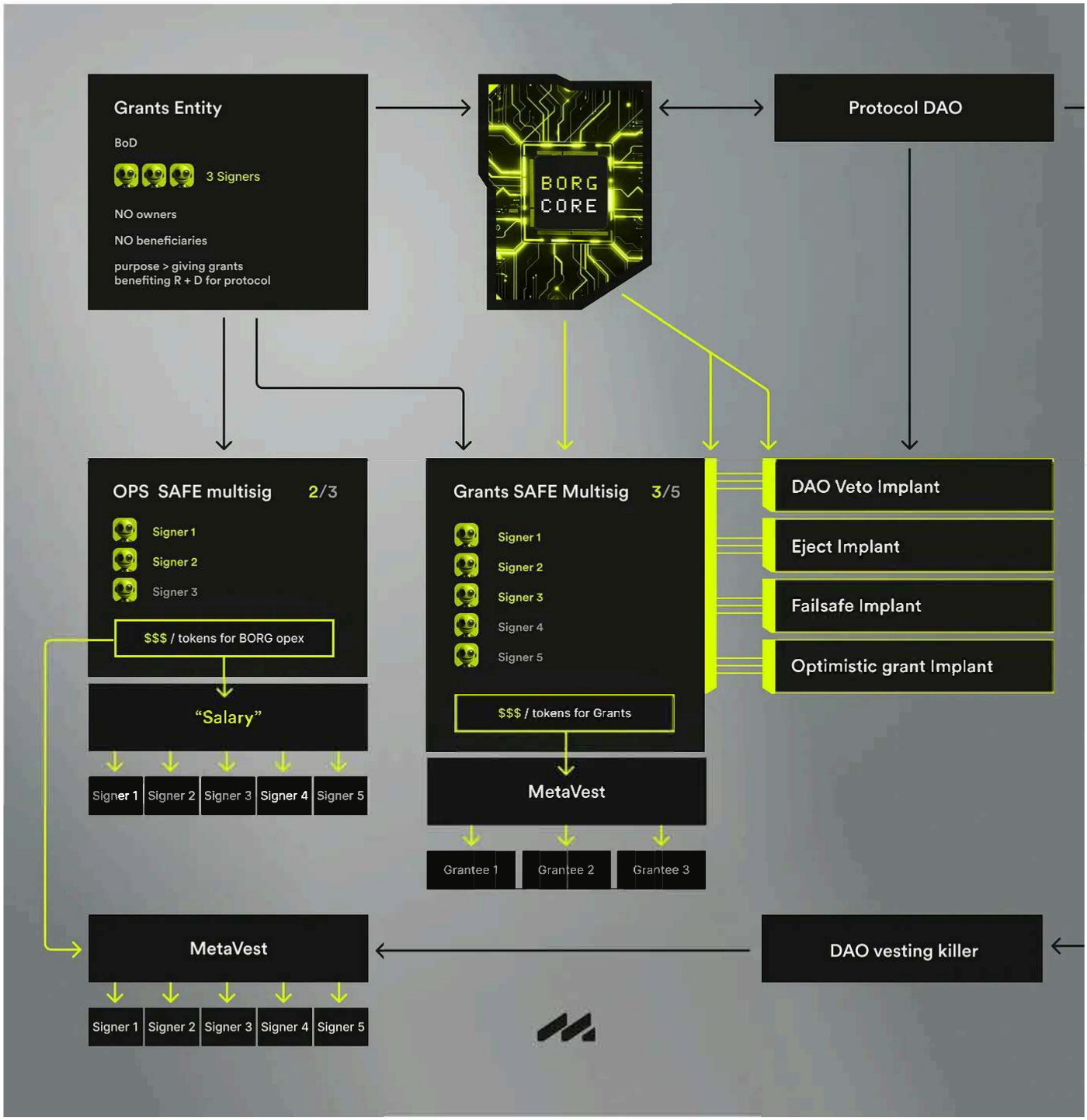
MetaVesT can be used both to mediate the vesting of BORG personnel into their compensation received from the DAO, and to mediate the vesting of grants from th GrantsBORG to grants recipients. For instance, a GrantsBORG can utilize MetaVe to handle the disbursement of a custom grant type with complex terms. By setting specific conditions for release, such as milestone achievements or time-based vesti or a combination of the two, the GrantsBORG can ensure that funds are only releas when the predefined criteria are met. This could include a mix of passage of time p some oracle condition—like a token hitting a specific price—being satisfied. This mechanism not only automates the grant disbursement process but also ensures compliance with the DAO-approved purposes and off-chain rules. MetaVesT may a be paired with LeXscroW for additional programmability.

As compared to other token vesting/unlocking protocols, MetaVesT has different features more optimized for the cybernetic law philosophy, DAOs and BORGs. Alsc unlike some other such protocols, it is both source-available and open-source. It is intended to be a DeFi-style and governance-based solution to legal token vesting/unlocking arrangements–most similar to Hedgey, but with additional featu optimized for our cybernetic law philosophy–as compared to a centralized 'administrator-style' solution such as Toku. We don't believe token vesting/unlocki should depend on the whims of any central authority–whether it be a 'devco' or a Sa company like MetaLeX. 'Can't be evil' is preferable to 'don't be evil.'

Example Configuration – GrantsBORG

BORG OS is a very feature-rich and ultra-customizable suite that lets you have essentially any customization a BORG would need. Having highlighted many of its features, it is now useful to lean into a more specific, opinionated implementation o specific DAO-adjacent BORG, discuss how it is likely to be set up both on the technical and legal sides, and explore the 'game theory' of the resulting check/balan dynamic between the BORG and its adjacent/sponsoring DAO.

For this purpose, we will explore what we'd view as a typical/recommended DAO-adjacent GrantsBORG set up, depicted below. ***NOTE:*** *This is just one 'opinionated' configuration we think is likely suitable for most projects' purposes; we can of course work clients to configure many alternative configurations, according to their individual commun needs and governance philosophies. We* ***highly*** *recommend considering the needs of a BOR and customizing its setup rather than using this as a template*



More verbally, the basic setup of a standard DAO-adjacent Grants BORG would be:

1. a DAO:

- governing some onchain smart contract system—let's assume, for the moment, DeFi system—and

- in control of some source of 'money'—be it premined governance tokens, the governance token mint function, or tokens in a diversified 'DAO treasury' (***note*** this DAO could itself be 'legally wrapped' or not; it does not matter to the BOR model);

2. a memberless, beneficiary-less, no-tax, Caymans Foundation that, by virtue of the rules in its Memorandum of Association, Articles, and Bylaws, is expressly and sole devoted to giving grants in support of the DeFi system—a Grants Foundation;

3. 'money' (in the form of tokens) to be donated by the DAO to the Grants Foundati via an approved DAO governance proposal (or a series of approved DAO governanc proposals);

4. a division of that money into conceptual buckets—one intended to constitute the Grants Foundation's operational budget for some period of time (say, two years)—a another intended to be used by the Grants Foundation fund the grants;

5. a separate SAFE multisig to hold each of these buckets—the Ops Multisig and th Grants Multisig;

6. legal documentation establishing that the Grants Foundation is the owner of the multisigs and the tokens they 'control' (or, more colloquially, 'hold');

7. election (or at least acceptance) of the initial signers of these multisigs by the DA as part of the aforementioned proposals;

8. documentation establishing that these signers, when it comes to these multisigs related activities, are working for the Grants Foundation;

9. documentation making all or a subset of the signers into directors of the Grants Foundation (***note***—this is optional—alternatively, the Grants Foundation can have o or more 'professional directors,' though this adds expense and complexity, and the SAFE signers can be mere service providers of the Grants Foundation);

10. a target runtime for the Grants BORG before it would likely need a 'top-up' of funding from the DAO—let us suppose in this case the Grants BORG is meant to ru for two years;

11. a target 'strategy' for the Grants BORG—let us say in this case it's to make lots a lots of small 'bootstrapping' grants to promising builders;

12. connecting the Grants Multisig to BORG OS:

- BORGcore + OptimisticGrantImplant will allow the Grants BORG to operate freely within its determined rate limit—so, for example, if it had $50M worth o funds meant to last two years and was meant to divide that into roughly 200 gra of roughly $250k each, it could freely and 'optimistically' grant up to 4 grants o to $250k each per month without needing any DAO approval or being subject t DAO veto;
- BorgCore + OptimisticGrantImplant + DAO Veto Implant will allow the Grant BORG to exceed its rate limits, subject to a timelock and DAO veto in each cas
- BORGCore + ejectImplant + failsafeImplant will allow multisig signers to resig from their roles, and if too many do so, will enable the funds to revert to the DA (more on the reasons for this below);
- MetaVesT will handle both:
  - the vesting of grant amounts to the grantees; and
  - the vesting of material governance incentive awards to the Grants BORG personnel (more on the reasoning for this below) (***note***--their ordinary cash compensation can be handled discretionarily out of the Ops Multisig).

Checks & Balances

MetaLeX's design philosophy emphasizes check-and-balance dynamics using a mix cryptoeconomic incentives and legal structures. Let's consider in more detail how these shake out in the sample configuration summarized above:

- The BORG is pretty much free to do its thing for two years, as long as it sticks the rules the DAO wanted—give lots of modest bootstrapping grants to projec that might benefit the DeFi system governed by the DAO, spread out evenly ov the two-year period.
- The BORG has an Ops budget (decided in advance by the DAO) and needs to m that last the two years, but can basically do whatever it wants with it, as long as the uses supports the BORGs grant-giving mandate—so, it can use this to pay personnel, in whatever way is determined by the Board of Directors of the Gra Foundation. The DAO does not need to approve every little compensation or operational spending decision;
- We assume in this case that the 'big money' for the Grants BORG personnel—their main upside and the reason they are spending at least two years on giving out grants for this ecosystem—is their upside in the governance token of the DAO. For this reason, the incentive token awards are set aside in a MetaVesT contract, where the vesting of each individual personnel:
    - can be tax-optimized as token options, restricted token awards or RTUs;
    - can be terminated by the Board of Directors (this would presumably be because the person has been 'fired from' or has 'quit' the Grants BORG); an
    - can be terminated by the DAO—although the DAO does not have the powe hire and fire the BORG's personnel, this can be seen as a strong 'invitation leave' from the DAO to one or more Grants BORG personnel—and since th Grants BORG Personnel can, in fact, unilaterally resign from the multisig v the ejectImplant, this could be quite effective.
- In case of abandonment of the GrantsBORG (which could be triggered due to attrition among the GrantsBORG personnel, or even by an *en masse* resignation the event of the GrantsBORG getting hit by some adverse legal action), the fun automatically revert to the DAO via failSafeImplant—this means the GrantsBC personnel always have an 'out', but can exercise it without personally making a

decision about where the money goes; it's already been made for them, and programmed onchain.

Under this structure, the initial structure-specific legal rules and smart contracts–establish a baseline rule-enforcement mechanism, while the DAO acts rather like a fuzzier-logic accountability oracle to handle edge-cases. Together, these structures ensures that the BORG is following its own rules.

Conclusion & Future Releases

While MetaLeX v1.0 is now complete, the small number of use cases and configurations we have mentioned here (mainly focused on GrantsBORGs) are just start of unlocking its potential. In the coming weeks and months, we will aim to release more and more 'out-of-the-box' configurations of BORG OS, paired with a customized UI for each one on our web app–securityBORGs, finBORGs, lexBORGs ventureBORGs and more. Additionally, we will be releasing standalone LeXscroW MetaVesT apps with some commonly used patterns pre-configured–such as a 'cloutScroW for KOL twitter bets and SAFT and token contributor grant configurations for MetaVesT.

Links

- BORGcore: https://github.com/MetaLex-Tech/borg-core
- LeXscroW: https://github.com/MetaLex-Tech/LeXscroW
- MetaVesT: https://github.com/MetaLex-Tech/MetaVesT
- Mixbytes Audit: https://github.com/mixbytes/audits_public/tree/master/MetaLeX
- Zellic Audit (MetaVesT only):

https://reports.zellic.io/publications/metavest

Cybernetic law is now.

4 Likes

Discussion about this post

Comments Restacks

Write a comment...