Invariant

Pondora v2

Solo Audit Report v1.1

September 2, 2025

Contents

Revision table

Report version	Report name	Date	Report URL
1.1	New features	2025-09-02	Full report link
1.0	Main audit	2025-06-05	Full report link

1 Executive summary

THIS REPORT DOES NOT PROVIDE ANY WARRANTY OF QUALITY OR SECURITY OF THE AUDITED CODE and should be understood as a best efforts opinion of Invariant0 produced upon reviewing the materials provided to Invariant0. Invariant0 can only comment on the issues it discovers and Invariant0 does not guarantee discovering all the relevant issues. Invariant0 also disclaims all warranties or guarantees in relation to the report to the maximum extent permitted by the applicable law. This report is also subject to the full disclaimer in the appendix of this document, which you should read before reading the report.

Project overview

Pondora v2 brings smart account functionality to Cardano. A user can lock their assets in a *Pond* smart contract and approve a set of instructions called *intents*; they say how, when, and under what conditions a part of those funds may be used. Think of it as a programmable vault: you decide the rules on-the-go and let somebody else execute them for you when the conditions are right, potentially for a fee.

To see the whole project overview, please refer to the first audit revision here. This audit is only about certain changes to the base Pondora contracts and a few initial modules that were audited in the previous audit. The changes include:

- **The introduction of a temporary owner**. This is a new feature that allows a user to authorize a hot key called a temporary owner to act on behalf of the original owner for a limited time. It makes the UI a bit more smooth as the user doesn't have to sign messages manually. However, it is important to note that by doing that they are giving up some control over the funds locked in their smart account to that key. Even for those using hardware wallets, full access to any smart account authorizations are given, which means full access to any funds locked in the smart account — with the only limitation being the validity period.
- **The introduction of child intents**. This is a new feature that allows a user to authorize **strategies** of intents that depend on each other; an intent can be set to be authorized if and only if a specified parent intent is executed beforehand. And the chain can be arbitrarily long, even recursive.

  As noted in the code, parent modules making use of this feature should check

from within that their child is authorized in that transaction. If they do not do so, the strategy chain can be broken, making it unable to continue further without manual intervention. No modules that are currently in the codebase do this as of now. If the feature is used, the trust to not break the chain is up to the batcher. This will be more important to double-check once the Decentralized Order Routing Aggregator (*DORA*) is launched.

- **Unlock module can control accompanying Ada**. If the unlock module expects a payment in a token other than Ada, it can now assert that a specific Ada amount is present in the payment UTxO as well. Note: The Ada should **not** come from the same UTxO, though — as no Ada is controlled in UTxOs spent with a non-Ada label as is the case with paying in a non-Ada token.
- **Remake of off-chain auth message signing**. This is a technical change to cater to how different wallets handle CIP-8 standard of signing off-chain messages on Cardano.

Audit overview

I started the audit at commit *66550c595f* and it lasted from August 26, 2025 to September 2, 2025. The timeframe is inclusive of periods in which I was awaiting the implementation of fixes by the client. We interacted mostly on Discord. The team fixed all issues to my satisfaction, except for 1 minor finding that was acknowledged as intended. They do not represent security threats to the system alone, only if combined with other off-chain attack vectors. I found the team highly responsive and collaborative throughout the audit process, and the documentation provided was of good quality, which significantly aided my understanding of the system.

The scope of the audit was limited to the updates to the smart contract files only, done since the previous audit. I did not review any tests as part of this audit. I performed a design review along with a deep manual audit of the code and reported findings along with remediation suggestions to the team in a continuous fashion, allowing the time for a proper remediation that I reviewed afterwards. See more about our methodology in Methodology.

The commit *e608e92eac4126be0f6f7567aa5db378db522f0d* represents the final version of the code. The status of any issue in this report reflects its status at that commit. You can see all the files audited and their hashes in Audited files. The smart contract language used is Aiken and the contracts are intended to run on Cardano. To avoid any doubt, I did not audit Aiken itself or the external *merkle-patricia-forestry* library, which were assumed to function correctly.

Summary of findings

During the audit, I identified and reported 1 critical, 2 major, 2 medium, 1 minor, and 1 informational findings. For details on severity and status classification, please refer to Classification. Please note, that the counts as well as further sections do not include any findings from the previous audits.

All findings were fixed prior to the final commit except for one minor finding that was acknowledged as expected:

- **POND2i-301**: Temporary owner can authorize intents not bound by his validity. A temporary owner is authorized for a limited time. If he signs an off-chain authorization for an intent, the validity is checked against the current time at the time it's being used. However, by creating an on-chain intent authorization UTxO while he is still in power, he can go around this as the on-chain intent authorization is valid unless cancelled.

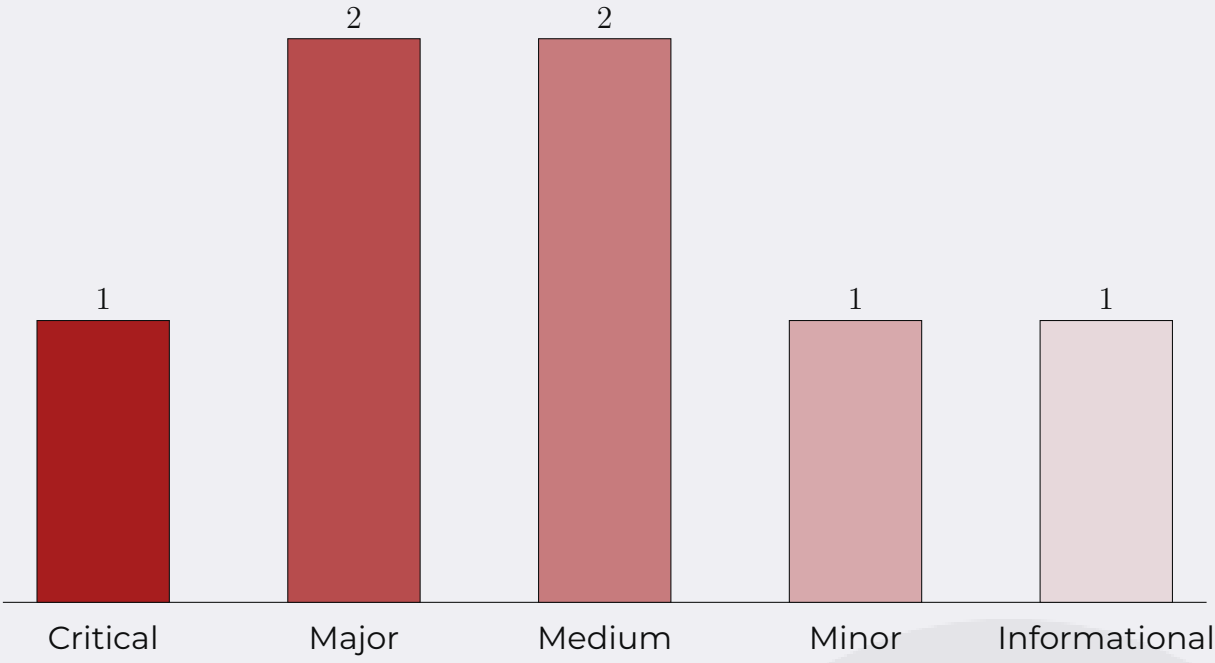
  This was acknowledged as expected, no relevant on-chain code changed. The client has implemented some security measures for temporary key management in their application layer which is outside the scope of this audit. However, from an on-chain perspective, temporary owners retain full authorization capabilities during their validity period and the authorization is valid unless cancelled.

  It is recommended that users set the temporary owner validity to be as short as practical and review all on-chain authorized intents before the temporary owner's expiration, as any intents must be created by that time. Users should promptly cancel any unexpected or potentially malicious authorizations they discover.

Overall, the reported findings can be broadly categorized as below:

- **Authorization and access control:** Critical authorization bypass allowing intents to be authorized on behalf of any user (POND2i-001), and design issues with temporary owner authorization validity (POND2i-301).
- **Child intent implementation issues:** Multiple issues in the child intent feature including minting failures (POND2i-102), incorrect authorization removal logic (POND2i-201), and insufficient control over token burning (POND2i-202).
- **Code implementation errors:** Timestamp validation error in the unlock module making it inexecutable when both validity bounds are set (POND2i-101).
- **Code quality and documentation:** Non-critical improvements for code style and documentation clarity (POND2i-401).

2 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
POND2i-001	Intents can be authorized on behalf of any other user	CRITICAL	RESOLVED
POND2i-101	Unlock module's start validity timestamp makes it inexecutable	MAJOR	RESOLVED
POND2i-102	Child intent tokens can not be minted as intended	MAJOR	RESOLVED
POND2i-201	Authorized child intent can be used to unauthorize authorizing hash	MEDIUM	RESOLVED
POND2i-202	Burning of child intent tokens is not controlled enough	MEDIUM	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
POND2i-301	Temporary owner can authorize intents not bound by his validity	MINOR	ACKNOWLEDGED
POND2i-401	Code style and documentation	INFORMATIONAL	RESOLVED

POND2i-001 Intents can be authorized on behalf of any other user

Category	Vulnerable commit	Severity	Status
Logical Issue	`66550c595f`	CRITICAL	RESOLVED

Description

The finding is about on-chain intent authorizations. Those are UTxOs holding authorized `root_hash` in their datums and holding intent tokens of the relevant owner that is said to have authorized the hashes. Due to a bug introduced to the `all_outputs_paid_to_script` function on line #296, it is possible to spend any owner's on-chain intent authorization's UTxO and replace the authorized hash with any other hash. That is, because it is no longer required for the `asset_name` of the intent auth token found among the outputs to match the owner that authorized the spend — the code passes even if the owners don't match.

As a result, all funds from all users that use on-chain intent authorizations can be emptied by an attacker.

Recommendation

To enforce that checks on line #286 s.a. the owner equality check hold for all intent tokens found among the transaction outputs, it is necessary to default to `False` on line #296, not `True`.

Resolution

The issue was resolved by the commit `a8beb56b0a` .

POND2i-101 Unlock module's start validity timestamp makes it inexecutable

Category	Vulnerable commit	Severity	Status
Code Issue	`66550c595f`	MAJOR	RESOLVED

Description

There is a new field added to the unlock module parameters called `valid_from`. There's also a `valid_until` timestamp in the parameters. If both are set, it is checked that the transaction's validity interval falls in between the `(valid_until, valid_until)` range instead of checking that it falls in the `(valid_from, valid_until)` interval. As the former interval allows just a single millisecond timestamp, it is unfeasible to execute the intent in reality.

Recommendation

Correct the error on the line #140 in the unlock module so that the `valid_from` timestamp is checked instead of the `valid_until` as the validity start.

Resolution

The issue was resolved by the commit `a8beb56b0a` .

POND2i-102 Child intent tokens can not be minted as intended

Category	Vulnerable commit	Severity	Status
Logical Issue	`66550c595f`	MAJOR	RESOLVED

Description

If a parent intent is observed, an authorized child intent token should be allowed to be minted. For it, it needs to observe an authorized `child_ref_root` and prove the child intent's inclusion in that merkle tree. However, as per the `all_outputs_paid_to_script` function's logic, the token can only be put to a UTxO authorizing the very same `child_ref_root` merkle root hash. As such, no new intents are authorized and the intended child intents can not be used.

Recommendation

I suggest correcting the logic in the `all_outputs_paid_to_script` function on the line #292 to allow for a new `child.root_hash` instead.

Resolution

The issue was resolved by the commit `a8beb56b0a` .

POND2i-201 Authorized child intent can be used to unauthorize authorizing hash

Category	Vulnerable commit	Severity	Status
Logical Issue	`66550c595f`	**MEDIUM**	**RESOLVED**

Description

If a child intent is authorized as verified in an observed "authorizing" `child_ref_root`, this child authorization can be used to revoke that `child_ref_root` — instead of the removal of the `child.root_hash`, which was likely the intended flow. As this can be done by anyone potentially, assuming the knowledge of the MPT structures, it can be used to remove authorizations and slow down execution.

Recommendation

If a child intent is authorized to be removed upon its parent's execution, allow the removal of `child.root_hash` on line #123 instead of the authorizing hash's removal.

Resolution

The issue was resolved by the commit `a8beb56b0a` .

POND2i-202 Burning of child intent tokens is not controlled enough

Category	Vulnerable commit	Severity	Status
Design Issue	`66550c595f`	**MEDIUM**	**RESOLVED**

Description

A child intent can be authorized to be minted and burned whenever a specified parent intent executes. The minted quantity, as well as the actual burn of the token is not checked in the intents validator, though. As a result, depending on the implementation of the check in the parent intent (not present in the current codebase), it might be possible to:

- Mint multiple intent tokens and create multiple UTxOs authorizing the child intent, out of which it is enough to burn one if the parent intent expects the burn. That way, the child can be kept authorized even upon a valid single burn. Depending on the intent, this can damage the owner significantly.
- Spend the on-chain intent UTxO with a `Burn` action, but do not actually burn the token, just recreate the UTxO. This is possible as the `all_outputs_paid_to_script` function does not check the `child.action` when checking the output.

Recommendation

I suggest both implementing a tighter control of the minted and burned amounts to be equal to exactly +1 or -1 and a fix in the `all_outputs_paid_to_script` function whereas it would be checked that the `child.action` is set to `Mint` in case there's a token found among the transaction outputs.

Resolution

The issue was resolved in the commit `e608e92eac` .

POND2i-301 Temporary owner can authorize intents not bound by his validity

Category	Vulnerable commit	Severity	Status
Design Issue	*66550c595f*	MINOR	ACKNOWLEDGED

Description

A temporary owner is authorized to act on behalf of the original owner for a limited time. However, if he creates an on-chain intent authorizations for a merkle tree of intents while he is still in power, those are not bound by any kind of validity. As such, they are still authorized and can still be executed even after the authority of the temporary owner expires.

Recommendation

I suggest adding and enforcing an optional validity period to on-chain intent UTxOs — in case of temporary owners, they could not authorize something that would be valid beyond their own validity; and the same would apply to child intents that could not go beyond their parent's validity.

Resolution

The issue was acknowledged as expected, no relevant on-chain code changed. The client promised a clean UI showing all on-chain authorized intents so users can easily see if the key is being misused and cancel the authorization. Also, active monitoring of on-chain intents by temporary owners was encouraged and is being considered by the client.

POND2i-401 Code style and documentation

Category	Vulnerable commit	Severity	Status
Code Style	`66550c595f`	**INFORMATIONAL**	**RESOLVED**

Description

Across the codebase, there are a few codestyle and documentation suggestions:

- In some files, `auth` refers to both the `auth.ak` imported file and redeemer fields, I suggest making use of different names for the two.
- The documentation of the `verify_native_auth` function refers to signing MPT root hashes only. However, the function is generalized now to support signing of other data as well — for example in the child intent scenario where the message comprises of serialized transaction inputs and outputs.

Recommendation

Suggestions are included in the description.

Resolution

The issue was resolved by the commit `a8beb56b0a` .

A Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the agreement between Invariant0 LLC (Invariant0) and Pond Labs LLC (Client) (the Agreement), or the scope of services, and terms and conditions provided to the Client in connection with the Agreement, and shall be used only subject to and to the extent permitted by such terms and conditions. This report may not be transmitted, disclosed, referred to, modified by, or relied upon by any person for any purposes without Invariant0's prior written consent.

This report is not, nor should be considered, an endorsement, approval or disapproval of any particular project, team, code, technology, asset or anything else. This report is not, nor should be considered, an indication of the economics or value of any technology, product or asset created by any team or project that contracts Invariant0 to perform a smart contract assessment. This report does not provide any warranty or guarantee regarding the quality or nature of the technology analysed, nor does it provide any indication of the technology's proprietors, business, business model or legal compliance.

To the fullest extent permitted by law, Invariant0 disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. This report is provided on an as-is, where-is, and as-available basis. Invariant0 does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by Client or any third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services, assets and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Invariant0 will not be a party to or in any way be responsible for monitoring any transaction between you and client and/or any third-party providers of products or services.

This report should not be used in any way by anyone to make decisions around investment or involvement with any particular project, services or assets, especially not to make decisions to buy or sell any assets or products. This report provides general information and is not tailored to anyone's specific situation, its content, access, and/or usage thereof, including any associated services or materials, shall not

be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or other advice.

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Invariant0 prepared this report as an informational exercise documenting the due diligence involved in the course of development of the Client's smart contract only, and THIS REPORT MAKES NO CLAIMS OR GUARANTEES CONCERNING THE SMART CONTRACT'S OPERATION ON DEPLOYMENT OR POST-DEPLOYMENT. This report provides no opinion or guarantee on the security of the code, smart contracts, project, the related assets or anything else at the time of deployment or post deployment. Smart contracts can be invoked by anyone on the internet and as such carry substantial risk. INVARIANT0 HAS NO DUTY TO MONITOR CLIENT'S OPERATION OF THE PROJECT AND UPDATE THE REPORT ACCORDINGLY.

THE INFORMATION CONTAINED IN THIS REPORT MAY NOT BE COMPLETE NOR INCLUSIVE OF ALL VULNERABILITIES. This report is not comprehensive in scope, it excludes a number of components critical to the correct operation of this system. You agree that your access to and/or use of, including but not limited to, any associated services, products, protocols, platforms, content, assets, and materials will be at your sole risk. On its own, it cannot be considered a sufficient assessment of the correctness of the code or any technology. This report represents an extensive assessing process intending to help Client increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology, however blockchain technology and cryptographic assets present a high level of ongoing risk, including but not limited to unknown risks and flaws.

While Invariant0 has conducted an analysis to the best of its ability, it is Invariant0's recommendation to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring and/or other auditing and monitoring in line with the industry best practice. The possibility of human error in the manual review process is highly real, and Invariant0 recommends seeking multiple independent opinions on any claims which impact any functioning of the code, project, smart contracts, systems, technology or involvement of any funds or assets. INVARIANT0'S POSITION IS THAT EACH COMPANY AND INDIVIDUAL ARE RESPONSIBLE FOR THEIR OWN DUE DILIGENCE AND CONTINUOUS SECURITY.

B Audited files

The files and their hashes reflect the final state at commit *e608e92eac4126be0f6f7567aa5db378db522f0d* after all the fixes have been implemented.

SHA256 hash	Filename
76473...a79b6	lib/pondora/accounts.ak
72c27...59515	lib/pondora/benchmarks.ak
44a78...065f4	lib/pondora/messages.ak
e07b8...d01dc	lib/pondora/types.ak
41541...f3f7d	lib/pondora/utils.ak
26934...60009	validators/intents_native.ak
351a8...28a42	validators/modules/helper_nonce.ak
ac3c6...1a76e	validators/modules/module_defragment.ak
a83e3...48892	validators/modules/module_dependencies.ak
e8213...f04f7	validators/modules/module_unlock_fungible_outputs_for_payment.ak
cd1da...31e57	validators/pond_native.ak
20443...f7826	validators/pond.ak

Please note that we did not audit Aiken itself or the external *merkle-patricia-forestry* library used, which were assumed to function correctly.

C Methodology

At Invariant0, our agile methodology for performing security audits consists of several key phases:

1. Design reviews form the initial stage of our audits. The goal of the design review is to find larger issues which result in large changes to the code fast.
2. During the deep code audit, we verify the correctness of the given code and scrutinize it for potential vulnerabilities. We also verify the client's fixes for all discovered vulnerabilities. We provide our clients with status reports on a continuous basis providing them a clear up-to-date status of all the issues found so far.
3. We conclude the audit by handing over a final audit report which contains descriptions and resolutions for all the identified vulnerabilities as well as an executive summary featuring project overview, collaboration notes and a brief summary of findings found.

Throughout our entire audit process, we report issues as soon as they are found and verified. We communicate with the client for the duration of the whole audit. During our audits, we check several key properties of the code:

- Vulnerabilities in the code.
- Adherence of the code to the documented business logic.
- Potential issues in the design that are not vulnerabilities.
- Code quality and potential for improvement.

During our manual audits, we focus on a wide range of attacks; including but not limited to double satisfaction, theft of funds, violation of business requirements, token uniqueness attacks, faking timestamps, locking funds indefinitely, denial of service, unauthorized minting, and loss of staking rewards.

D Issue classification

Severity levels

The following table explains the different severities. In general, the darker the color, the more severe the issue.

Severity	Impact
CRITICAL	Theft of user funds, permanent freezing of funds, protocol insolvency, etc.
MAJOR	Theft of unclaimed yield, permanent freezing of unclaimed yield, temporary freezing of funds, etc.
MEDIUM	Smart contract unable to operate, partial theft of funds/yield, etc.
MINOR	Contract fails to deliver promised returns, but does not lose user funds.
INFORMATIONAL	Best practices, code style, readability, documentation, etc.

Resolution status

The following table explains the different resolution statuses. In general, the darker the color, the more attention the issue requires.

Resolution status	Description
RESOLVED	Fix applied.
PARTIALLY RESOLVED	Fix applied partially.
ACKNOWLEDGED	Acknowledged by the project to be fixed later or out of scope.
PENDING	Still waiting for a fix or an official response.

Categories of issues

The following table explains the different categories of issues.

Category	Description
Design Issue	High-level issues in the design. Often large in scope, requiring changes to the design or massive code changes to fix.
Logical Issue	Medium-sized issues, often in between the design and the implementation. The changes required in the design should be small-scaled (e.g. clarifying details), but they can affect the code significantly.
Code Issue	Small in size, fixable solely through the implementation. This category covers all sorts of bugs, deviations from specification, etc.
Code Style	Parts of the code that work properly but are possible sources of later issues (e.g. inconsistent naming, dead code).
Documentation	Small issues that relate to any part of the documentation (design specification, code documentation, or other audited documents). This category does not cover faulty design.
Optimization	Ideas on how to increase performance or decrease costs.

E Report revisions

This appendix contains the changelog of this report. Please note that the versions of the reports used here do not correspond with the audited application versions.

Report v1.1: New features

Revision date: 2025-09-02
Final commit: *e608e92eac4126be0f6f7567aa5db378db522f0d*
Report link: Full report link

I audited a few changes to the base Pondora contracts, s.a. the introduction of temporary owner, child intents, off-chain auth message remake, and more. To see the files audited, see Audited files.

Report v1.0: Main audit

Revision date: 2025-06-05
Final commit: *bf21576ffd7cca45f4e59ec0c9460ca63d361394*
Report link: Full report link

I conducted the audit of the base Pondora native contracts and a few initial modules including the payment unlock module, the defragmentation module, the service fee module, and the nonce helper. The files and revisions of this audit are included below.

SHA256 hash	Filename
76473...a79b6	lib/pondora/accounts.ak
91293...7df3b	lib/pondora/types.ak
d014c...265b4	lib/pondora/utils.ak

Continued on next page

SHA256 hash	Filename
20775...9112b	validators/intents_native.ak
351a8...28a42	validators/modules/helper_nonce.ak
ac3c6...1a76e	validators/modules/module_defragment.ak
a83e3...48892	validators/modules/module_dependencies.ak
4ab4a...21081	validators/modules/module_unlock_fungible_outputs_for_payment.ak
7b550...974eb	validators/pond_native.ak
20443...f7826	validators/pond.ak

F About us

Invariant0, the fresh new face of Vacuumlabs Auditing.

- You probably already know our team as former Vacuumlabs Auditing.
- We helped create WingRiders, currently the second largest decentralized exchange on Cardano (based on TVL).
- We prevented a massive $180M vulnerability in production.
- To date, we found and reported over 373+ vulnerabilities, delivered 21+ reports and thus protected over $150M in assets for our clients and their users.
- Our clients include *FluidTokens, Liqwid, WingRiders, Iagon*, and many more.

Our auditing team is chosen from the best.

- Talent from esteemed Cardano projects: WingRiders and NuFi.
- Rich experience across Google, traditional finance, trading and ethical hacking.
- Award-winning programmers from ACM ICPC, TopCoder and International Olympiad in Informatics.
- Driven by passion for program correctness, security, game theory and the blockchain technology.

Contact us:
info@invariant0.com