MAKING PREDICTIONS WITH EXTERNAL MACHINE LEARNING MODELS PRESERVING INTELLECTUAL PROPERTY AND USAGE RIGHTS

ROCKET CAPITAL INVESTMENT PTE LTD

INTRODUCTION

In this document we discuss the problem of using external models in a machine learning competition and present the Proof of Concept of a decentralized technical solution; the POC preserves the intellectual property of the models and certifies the usage of the models by using strong cryptography and blockchain techniques.

In Section 1 we describe the Challenges that ML model trainers face when looking to share the insights from trained ML model without having to share the proprietary inner working of the ML model; then, in Section 2 we present an overview of the POC, and how it was conducted, in Section 3 we highlight the challenges faced that have been overcome, worked around or accepted over the course of the POC, and in Section 4 we describe the outcome of the implemented POC, with technical details about the various techniques used. In Section 5 we briefly discuss the expected benefits if this solution is deployed in Production. Finally, in Section 6 we present some additional considerations and possible future improvements of the presented POC.

1. ML COMPETITION

Rocket Capital Inv. (http://rocketcapital.ai/) leverages the power of two key technologies: Artificial Intelligence and Blockchain. Our overarching mission is to pioneer a novel Portfolio Theory by merging our in-house financial expertise with external machine learning predictions through a blockchain-based competition in the realm of financial markets.

Over the past few decades, we have witnessed an unprecedented technological race that has significantly impacted the financial sector. This transformation has ushered in heightened complexities and demanded a diverse skill set for effective fund management, creating an imbalance in resources between established financial giants and smaller enterprises.

Typically, traditional leading hedge funds rely on active management and internal competencies to drive their operations. Consequently, a company's ability to excel competitively has largely hinged on the proficiency of its staff in gathering and interpreting information.

The limited size of a company's workforce inherently constrains its competitive edge. To surmount these limitations, some financial institutions have begun to enlist external experts to complement their internal capabilities.

Rocket Capital Investment takes this approach a step further by establishing a platform that facilitates the curation and rewarding of contributions from a decentralized community.

THE YIEDL COMPETITION

The YIEDL Competition leverages a decentralized platform to source and incentivize the most advanced machine-learning applications for finance. In this initial implementation, participants submit their predictions and earn rewards based on the performance of their submissions.[1]

The Competition consists of a sequence of challenges. In the inaugural implementation, each challenge had a one-week duration. This has been running for just under 20 weeks and averaging over 100 submissions per week.

PREDICTIONS VS. MODELS

The solution [2] we've adopted thus far offers numerous advantages for our fund and for participants, who effectively become decentralized assets of the company.

Nonetheless, there are limitations to the current approach:

- We rely on participants to periodically send us predictions, making it suitable primarily for low-frequency financial activities and challenging to respond to rapid market fluctuations.
- We cannot query real-time market opportunities that may arise at any moment.

An alternative approach would be to request participants to provide machine learning models instead of predictions. This would enable us to access the models for on-demand use.

While this approach is common in many ML competition platforms, such as Kaggle, it comes with certain drawbacks that could impede its adoption in real-world settings like financial institutions:

- These competitions are typically centralized and verifying results and rewards in relation to all other participants can be challenging. In other words, participants must place their trust in the competition organizer. Our current solution overcomes this limitation, albeit limited to predictions in the current iteration.
- When users upload their models to these platforms, they essentially grant organizers access to view and replicate their models. Highly skilled predictors may find this unacceptable, potentially limiting engagement of true experts in financial market competitions. One solution would be to have participants publish their models as black boxes and use them as needed. However, this introduces a third issue.
- Participants should possibly not be informed about when and how their models are utilized by the organizer, as this could lead to frontrunning and other malicious trading activities after each model execution. For example, if the model is tasked with confirming specific real-time trading opportunities, participants could be alerted by the model to the trading opportunities in question and act accordingly.

In this paper, we describe the development of a Proof of Concept (POC) for a competition centered on ML models that addresses all three of the aforementioned problems.

NUMERAI SOLUTION

Numerai is a crowdsourced hedge fund that leverages data science and machine learning models submitted by a global community of data scientists to make financial predictions and investments. Numerai's latest initiative introduces a feature that allows participants to upload serialized models for their data science competition submissions [3]. Serialization simplifies the model submission process. However, it's important to understand that serialization enables the execution of these models beyond the competition's confines. Furthermore, with dedicated effort, individuals might reverse engineer the model back into a more understandable source code format and retrieve its weights. This means that the serialized model doesn't provide full protection for the intellectual property of participants' models. Consequently, participants should exercise caution when sharing serialized models to avoid unintentional exposure of their proprietary algorithms or techniques. (https://docs.numer.ai/numerai-tournament/submissions/model-uploads.)

2. OVERVIEW OF THE POC

Our primary objective is to establish a competition centered on ML models, with the following overarching steps and constraints.

STEPS:

- Participants submit their models.
- The organizer retains the capability to execute the models for predictions at any given time.
- The organizer remains unaware of the internal workings of the models.
- Participants remain uninformed regarding when the model is executed, and the specific inputs used.
- Participation and usage are governed by blockchain smart contracts, serving as an impartial third-party guarantee.
- All model executions are logged, enabling later verification of prediction scores through recomputation. This also facilitates validation of rewards allocated to participants.

CONSTRAINTS:

1. The organizer **MUST** be able to execute the models for predictions at any time.
2. The organizer **MUST NOT** be aware of the internal workings of the models and be able to use them outside of the competition.
3. The participants **MAY NOT** be aware of when the model is executed and what inputs were used.
4. At the time of execution, the model inputs and outputs **MUST** be known only to the organizer and model owner.
5. Model inputs and outputs **MUST** be verifiable afterwards.

From an abstract architectural perspective, the model operates as a black box created by the participant and employed by the organizer. The model represents a pure function from input data to predictions, devoid of side effects (additional input/output) or state (no capacity to store information). A model signature is created when the participant submits the model, and this signature is subsequently recorded on the blockchain for reference when all interactions with the model by the organizer are logged on the blockchain.

TECHNOLOGIES

Here we present several technical solutions that have been explored to date, highlighting the advantages and disadvantages of each. It is evident that, given the current state of technology, there is no flawless solution; however, a robust one can be devised through a careful selection of cloud solutions supported by blockchain technologies.

HOMOMORPHIC ENCRYPTION

Pros: completely secure.

Cons: extremely slow, not available for real-world models, by default only hides data. There exist some implementations that, to some extent, could be used to run basic ML models because they can express basic matrix multiplication and some additional operators used in many ML models, like for instance neural networks. However, due to the complexity and limited knowledge of the financial sector, probably the best models are the ones based on a high-level goal, like reinforcement learning or genetic algorithms, which require an underlying Turing-complete capabilities, which are currently out of reach in any practical implementation of homomorphic encryption.

Conclusion: technology not ready for this use case.

ML CUSTOM BLOCKCHAIN INTEROPERABLE WITH OTHER L2 BLOCKCHAINS

Pros: technically sound solutions.

Cons: there is no production-ready blockchain that offers at the same time the possibility to run ML models and to hide model internals; ML model variety is limited.

The requirements for decentralization such as fairness, trustlessness and transparency limit the ability to both run ML models and to hide ML model information. Even without considering hardware, ML models of different flavors will need different software environments to be run. This raises the bar for participation in the network, increasing centralization.

When a new ML model is added, the software environment may have to be re-configured, meaning each participant may need to add a new configuration environment for each ML model in the blockchain network. This might slow down the network to the point it is rendered unusable. Publishing an ML model on-chain will require making the code public to be run by other participants in the network, which goes against what we are trying to achieve (constraint 2).

There are a handful of ML-related blockchain platforms which, in general, are marketplaces that facilitate value transfer for a network of off-chain ML computation providers. These are useful in making sure that ML authors are duly compensated, but do not provide a secret channel between sender and receiver where the messages can later be reliably reconstructed and made public. (Constraint 5).

Conclusion: technology not ready.

MODEL/DATA OBFUSCATION

Pros: fast.

Cons: obfuscation could hide features useful to the ML models to extract information; also, it makes with obfuscation it is more difficult for participants to use their own specialized datasets for training; finally, the YIEDL competition explicitly declares the assets to be predicted to implement a complete certification of the competition process, so a complete obfuscation would be possible for the features but not for relevant parts of the competition question.

Conclusion: not suitable.

CONFIDENTIAL COMPUTING

Pros: available on several cloud providers (CGS, AWS, …), proven technology, fast.

Cons: more suited to protect data and applications from real-time attacks and sniffing, the same set of keys are needed to store and use the model. With the current offering from cloud providers the entity who owns the billing account is also able to access the keys, so it does not guarantee the participants.

Conclusion: does not really solve the problem, can be used in addition to other techniques to strengthen the security.

CONTAINERS, CLOUD FUNCTIONS

Pros: available on several cloud providers, deployment and orchestration could be automated, internals could be hidden by a suitable orchestration architecture.

Cons: the orchestrator could probably access the model internals, and since the organizer will probably own the orchestrator, it is not secure enough; side effects (I/O and status) are not easily avoided.

Conclusion: could be used as a foundation layer but requires important efforts to satisfy the requirements.

GENERAL CLOUD ML SOLUTIONS (AMAZON SAGEMAKER, GOOGLE VERTEX AI, AZURE ML)

Pros: ML model implementation and deployment are easy, scalable solution, configurable access control, on-demand access, endpoint logging, private endpoints, monitoring and auditing.

Cons: while these solutions offer the possibility to share models and some fine-grained access control, they don't prevent the solution owner from gaining full control of the models. Hence, either it is the organizer is the owner, and can see the participant's models, thus invalidating constraint 2, or it is the participant paying, and then can see model executions, negating constraint 3.

Conclusion: Could be used with major efforts.

3. CHALLENGES

SOLUTION

A. The organizer retains the ability to execute the model in batch mode at any given moment, enabling real-time predictions (constraint 1).

B. The organizer does not know anything about the model's implementation and cannot use the model prediction capabilities outside of the competition process due to blockchain certification (constraint 2).

C. Both the inputs and outputs of each model invocation are securely stored in encrypted form on IPFS and the blockchain (addressing constraints 4 and 5).

D. Participants receive rewards based on their model's performance, determined by its accuracy in predicting outcomes for the provided inputs.

E. At a later stage, when the timing and specifics of model executions are no longer pertinent, decryption keys are released on the blockchain. This allows participants to verify if their rewards align with the declared scoring and reward policies.

It's worth noting that while points D, E, and F share similarities with the current YIEDL competition, adjustments are required to accommodate the added complexity of logging model inputs and outputs. The remaining points are specific to this Proof of Concept (POC)

SIGNIFICANT ASPECTS OF THE IMPLEMENTED SOLUTION

The implemented solution introduces several unique aspects:

- Preservation of Intellectual Property and Usage Separation: It addresses the broader issue of outsourcing code execution while safeguarding intellectual property and ensuring proper usage. This innovative approach distinctly separates ownership from usage, which is a significant departure from traditional models.
- Application to Outsourcing Financial Predictions: The solution can be effectively applied to the specific challenge of outsourcing financial predictions to multiple external ML/AI models. This empowers financial institutions to harness the expertise of external models while maintaining control over their proprietary data.
- Extension to Order Execution: With some additional efforts, the same model can be extended to address order execution challenges. It enables the outsourcing of strategies used to execute large orders without causing market disruption. This is especially valuable for achieving optimal execution conditions concerning price and slippage.
- Potential for Real-Time Portfolio Design: The model can also be extended to facilitate real-time portfolio design. This capability allows for dynamic and responsive adjustments to investment portfolios, aligning them with current market conditions and investment strategies.

In summary, this solution presents a groundbreaking approach to code execution outsourcing that has diverse applications in the financial sector, ranging from predictions to order execution and real-time portfolio management. Its ability to protect intellectual property and ensure proper usage sets it apart as a unique and valuable innovation in the industry.

4. OUTCOME

The constraints mentioned above lie largely in the communication domain. Constraints 2, 4 and 5 can be met if the way that the organizer, the participant, and the visibility of the communication to other participants is enforced and managed appropriately. For this POC, we introduce the following messaging protocol to meet these constraints and present our implementation outcomes.

MESSAGING PROTOCOL

OBJECTIVES

1. The contents of the message must initially be readable only by the sender and the receiver.
2. The contents of the message, including its timestamp, must be tamper-proof and publicly verifiable later.

PROBLEMS AND SOLUTIONS

1. We use the blockchain and a file system to publicly record messages in encrypted forms. These are time-stamped and cannot be modified once recorded. Later, the encryption-decryption keys can be published when there is a need to prove or verify the contents of the message.
2. The file reference (which may be something like a url) needs to be public to achieve verifiability. This makes the system susceptible to a third-party trying to claim the file reference as their own. There needs to be a mode of verification that the message sender is indeed the author of the file.
3. Public-private key pairs are not suitable for encrypting data of larger sizes. As such we utilize a scheme where a symmetric key is first generated and used to encrypt the data, then a public key is used to encrypt the symmetric key before sending.

COMPONENTS

Smart Contract (C): The smart contract lives on the blockchain, a public network. The smart contract can enforce logic to restrict who can make requests or responses. In addition, the smart contract can, in future versions, facilitate rewards or penalties in the form of blockchain tokens.

File System (FS): The File System is used to store larger amounts of data that may not be suitable for recording on the blockchain. File references to data (which are suitable for recording on the blockchain) are used to identify and retrieve data from the file system.

File systems such as IPFS are suitable for our use case.

A NOTE ON TERMINOLOGY

There are 3 types of keys used in our protocol.

1. Symmetric Key. This is a key used for encrypting larger pieces of data. The same key is used for decryption.
2. Public-Private Key Pair. This is a pair of keys (public and private) that are used for encrypting smaller pieces of data. One is used for encryption and the other for decryption. The private key is kept secret by the owner of this key pair.

3. Blockchain Wallet Key Pair. This is similar to the public-private key pair. The wallet address, essentially an account ID on the blockchain, is derived from the public key. The private key is kept secret by the owner of the wallet address.

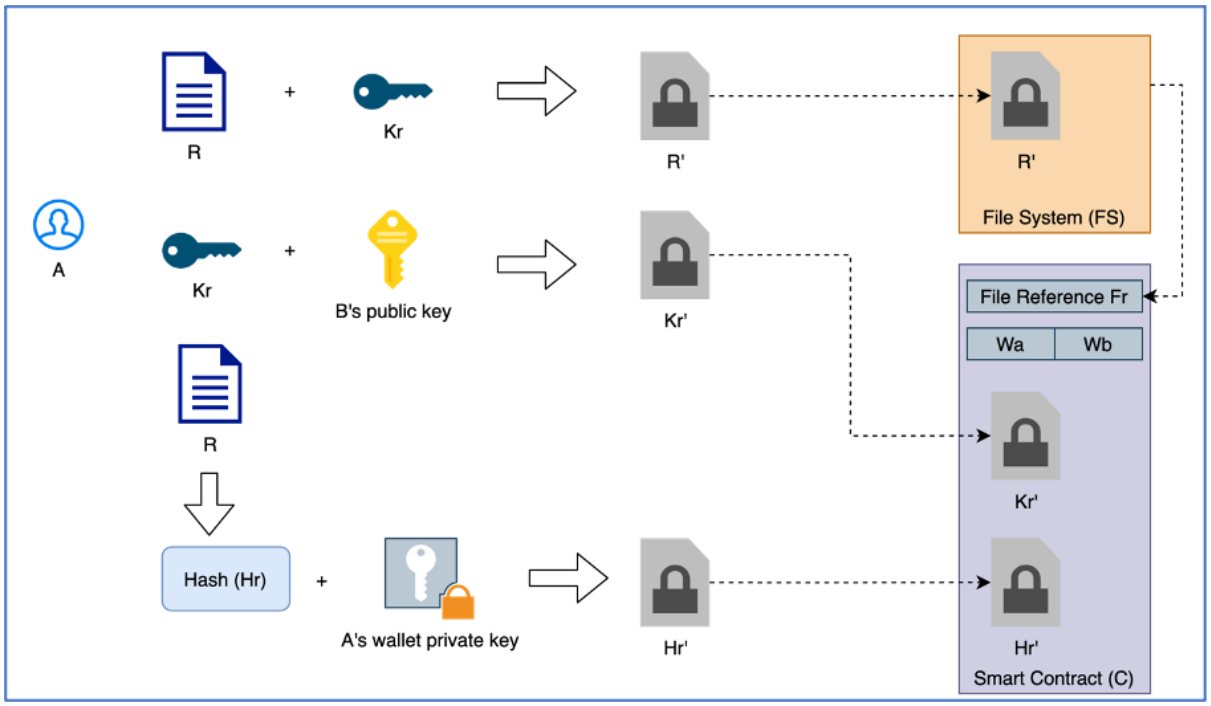
INITIAL CONDITIONS

For the following explanation, we consider a requester, A, and a responder, B.

The public keys of A and B are both made publicly available.

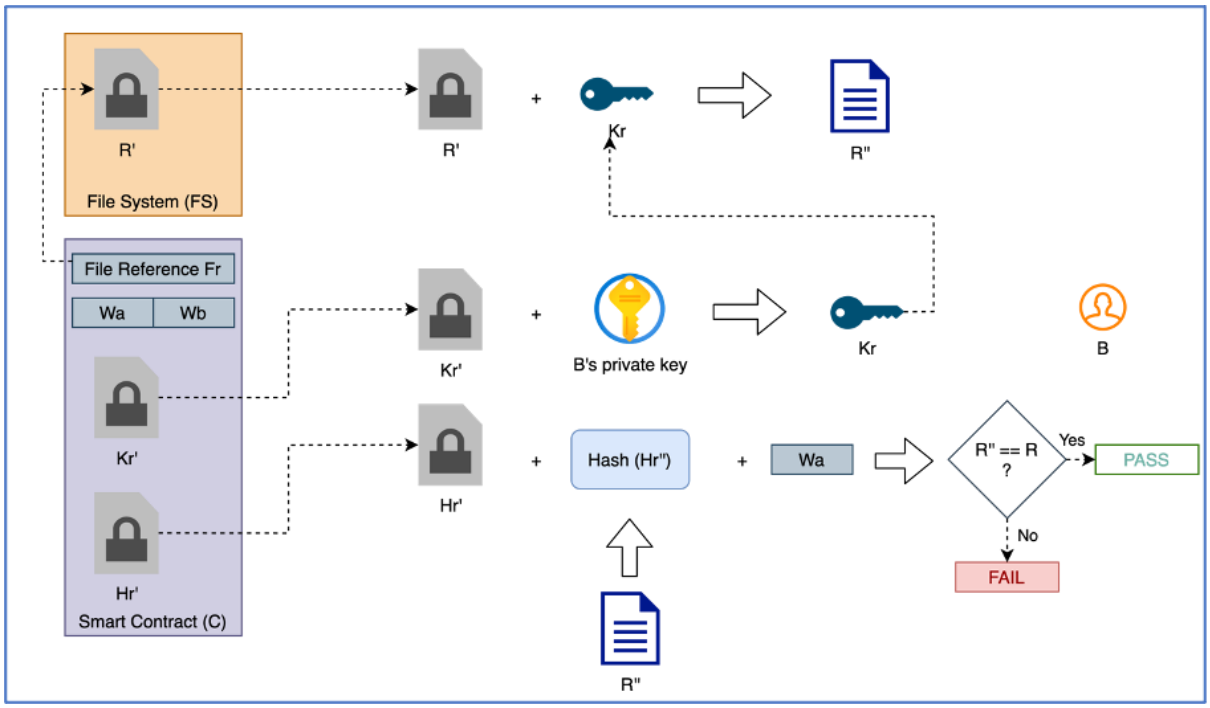
This also means that A and B each possess the corresponding private keys that can decrypt data encrypted with their public keys.

SENDING A REQUEST



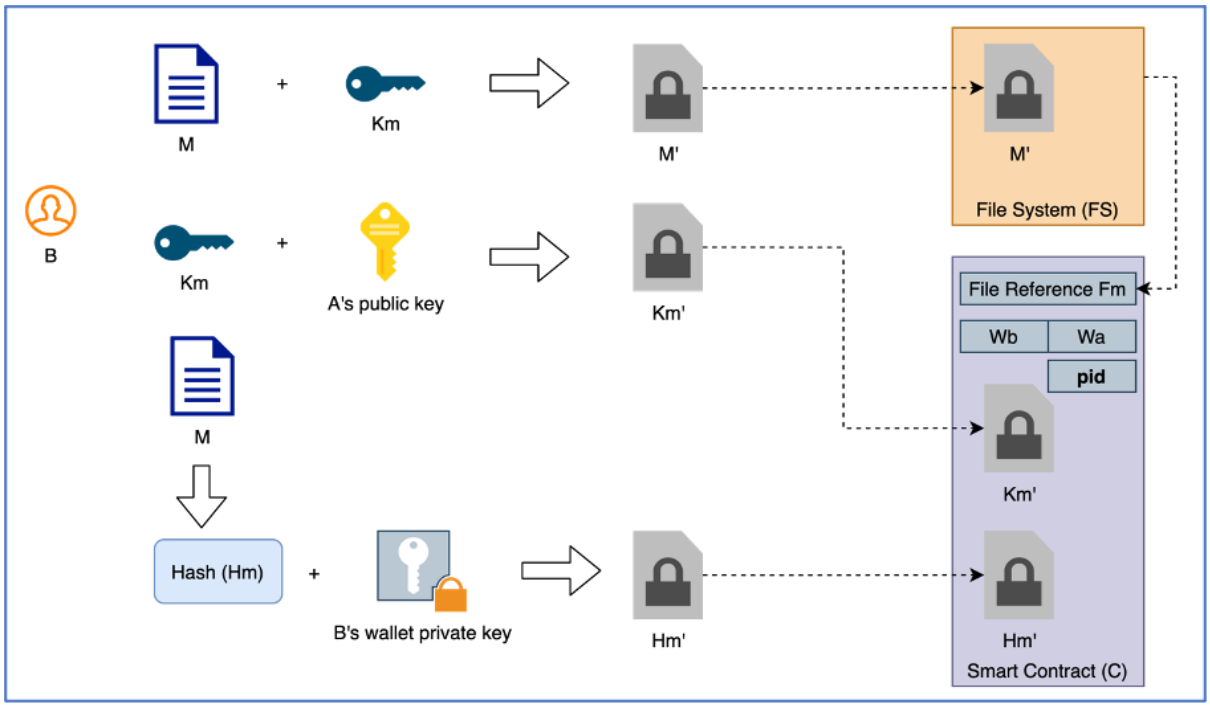
1. A generates a request message R.
2. A generates a symmetric key Kr.
3. A encrypts R with Kr obtaining R'.
4. A stores R' on FS and gets file reference Fr.
5. A encrypts Kr with B's public key obtaining Kr'.
6. A hashes R obtaining a hash Hr.
7. A signs Hr with A's wallet private key obtaining Hr'.
8. A registers request <Wa, Wb, Fr, Kr', Hr'> on smart contract C.
9. C enforces that Wa is the address of the requester A.

RECEIVING A REQUEST



1. B monitors contract C and detects that a request meant for B has been registered.
2. B reads <Wa, Wb, Fr, Kr', Hr'> from smart contract C.
3. B uses their private key to decrypt Kr' obtaining Kr.
4. B uses file reference Fr on FS to read R'.
5. B uses key Kr to decrypt R' obtaining R".
6. B hashes R" obtaining Hr".
7. B verifies Hr' against Wa and Hr"
8. B accepts R" as sent by A if and only if the verification passes, implying that R"==R.

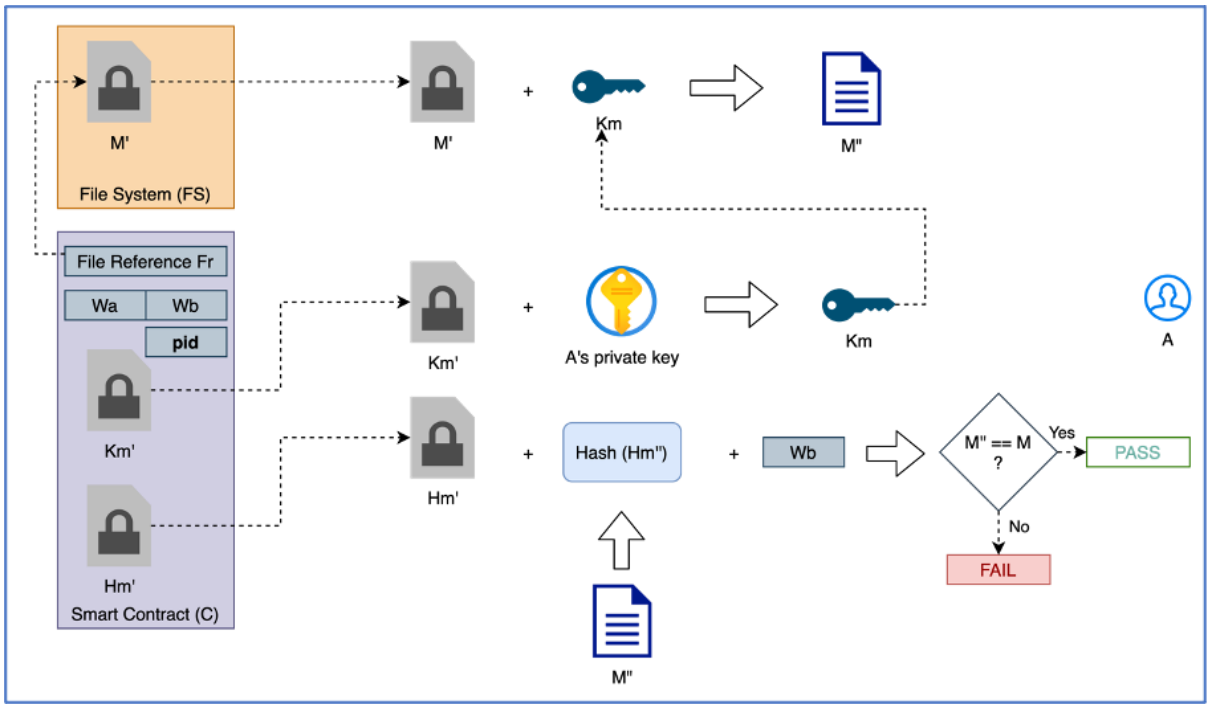
SENDING A RESPONSE



Sending a response is similar to sending a request, except that B registers an additional pid parameter, which is the message ID of the request this response is meant for.

1. B computes a response message M.
2. B generates a symmetric key Km.
3. B encrypts M with Km obtaining M'.
4. B stores M' on FS and gets file reference Fm.
5. B encrypts Km with A's public key obtaining Km'
6. B hashes M obtaining Hm.
7. B encrypts Hm with A's public key obtaining Hm'.
8. B registers response <Wb, Wa, Fm, Km', Hm', pid> on smart contract C, where pid is the unique message ID of the request message generated by the smart contract C.

RECEIVING A RESPONSE



1. A monitors contract C and detects that a request meant for A has been registered.
2. A reads <Wb, Wa, Fm, Km', Hm', pid> from smart contract C.
3. A uses their private key to decrypt Km' obtaining Km.
4. A uses file reference Fm on FS to read M'.
5. A uses key Km to decrypt M' obtaining M"
6. A hashes M" obtaining Hm".
7. A verifies Hm' against Wb and Hm".
8. A accepts M" as sent by B if and only if the verification passes, implying that M"==M.

IMPLEMENTATION

KEY COMPONENTS

- Smart Contract: Deployed on the Polygon Testnet, an Ethereum Virtual Machine (EVM)-based blockchain network, with the smart contract being written in the Solidity language.
- File System: Files are stored on the Inter-Planetary File System (IPFS), along with the Pinata Pinning Service to ensure the availability of files.
- Participants: Simulated via Python scripts running on Google Cloud Services (GCS)

SETTING UP

The smart contract assigns overall admin rights to the deployer at the point of creation. This deployer can then assign requester rights to selected wallet addresses. In other words, requests are restricted to a selected set of users. This is to prevent spam from potentially malicious actors. We also need the public key from both the organizer and each participant to be made available. Users participating in this protocol must send a transaction to upload their public key to the smart contract. The smart contract will enforce that the public key uploaded is linked only to the sender of the transaction. In other words, malicious actors cannot manipulate the public key of other users.

MESSAGES

Message ID

- The message ID is an incrementing counter beginning from 1.
- Every message, whether request or response, is given a unique message ID that is 1 more than the last recorded message.
- Every message is either a request or a response.

Message Fields

Each message contains 8 fields.

Field Name	Description
mid	The message ID of this message.
sender	Wallet address of the message sender.
receiver	Wallet address of the intended receiver of the message.
file reference	Identifier from which the encrypted message file can be retrieved from the file system.
encrypted symmetric key	Symmetric key encrypted with the receiver's public key. This symmetric key is the one used for encrypting the message file, which can also be used to decrypt the message file.
signed hash	Hash of the unecnrypted message file signed using the sender's wallet private key.
pid	The message ID of the request message being responded to. "0" if this message is a request.
decrypted symmetric key	Unencrypted form of the symmetric key in the "encryptedSymmetricKey" field. This is meant to be empty when the message is first created. Later, the "sender" may record the decrypted symmetric key here so that anyone who wants to check and verify the contents of the encrypted file (accessed via the file reference) may do so.

MAKING A REQUEST

Requests are made to the smart contract via the *request* method.

The *request* method takes 4 inputs:

1. The wallet addresses of the intended responder.
2. The file reference of the encrypted request file. This is an ID to be used for retrieving the encrypted request file from the file system.
3. The encrypted symmetric key. This is the symmetric key used to encrypt the request file, which is then encrypted with the public key of the intended responder.
4. The signed hash of the request file. This is the hash of the unencrypted request file, signed with the wallet private key of the requester.

The message ID of this message is added to the set of pending IDs for the receiver of this message.

MAKING A RESPONSE

Responses are made to the smart contract via the *response* method.

The *response* method takes 5 inputs:

1. The wallet addresses of the intended recipient. This should be the "sender" of the request message that this response is meant for. (ie. the requester)
2. The file reference of the encrypted response file. This is an ID to be used for retrieving the encrypted response file from the file system.
3. The encrypted symmetric key. This is the symmetric key used to encrypt the response file, which is then encrypted with the public key of the requester.
4. The signed hash of the response file. This is the hash of the unencrypted response file, signed with the wallet private key of the responder.
5. The message ID of the request message that this response is meant for.

The smart contract enforces that:

- The sender must be the receiver of the message referenced by the "pid", i.e., the responder must be the intended responder specified in the original request message.
- The receiver must be the sender of the message referenced by the "pid", i.e., The receiver of this response must be the requester of the original request message.
- The "pid" must exist in the sender's set of pending IDs.

PYTHON SCRIPT USAGE

MESSAGE USER

Each *MessageUser* instance consists of an instance of the *Sender* class as well as an instance of the *Receiver* class. This encapsulates what each user needs to do to interact with the protocol. The **organizer** will need to **send** requests and **receive** responses, while each **participant** will need to **receive** requests and **send** responses.

SENDING A MESSAGE

The script will help to prepare and send a message according to the messaging protocol described above.

1. A symmetric key is generated. This key will be used to both encrypt and decrypt the message file.
2. The message file is encrypted using the symmetric key.
3. This encrypted file is uploaded and pinned to IPFS via the Pinata service. A file reference, a base58-encoded string that can be used to download the encrypted file from the public IPFS network, is returned.
4. The symmetric key is encrypted using the receiver's public key.
5. A hash is obtained from the unencrypted message file.
6. This hash is signed using the wallet private key of the sender, producing a signature in the ERC-191 format [4]. Signing is done using the ECDSA algorithm.
7. The message is registered on the smart contract according to the fields specified in the previous "Message" section. The "pid" must be specified if this message is a response. The "decrypted symmetric key" does not have to be specified at this time.

RECEIVING A MESSAGE

The script will help to poll, receive, and decipher a message according to the messaging protocol described above.

1. The message fields are read from the smart contract.
2. The file reference is used to download the encrypted message file from IPFS.
3. The encrypted symmetric key is decrypted using the receiver's private key.
4. This symmetric key is used to decrypt the encrypted message file.
5. A hash is obtained from the decrypted message file.
6. We verify this hash against the signature that was sent along in the message, by checking that the recovered wallet address is indeed the wallet address of the sender.

RESULTS

LATENCY ANALYSIS

Cryptography

Tests performed on an Apple M1 Machine with 16 GB Ram.

	Median Delay (ns)	Mean Delay (ns)
Generate symmetric key	1,000	1,096
Encrypt csv file (1MB)	14,887,000	14,533,350
Encrypt symmetric key	605,000	609,638
Hash csv file (1MB)	568,000	949,180
Sign Hash	4,762,000	4,850,152
Decrypt symmetric key	3,590,000	3,609,043
Decrypt csv file (1MB)	10,771,000	14,508,400
Verify Hash	7,379,000	7,521,713

Network

Network times should be treated as a rough guide as these can vary with factors such as the user machine's network setup, location, internet connection speed as well as overall network conditions.

	Median Delay (ns)	Mean Delay (ns)
Pin File to IPFS (1MB)	3,480,857,000	3,653,640,000
Retrieve File from IPFS (1MB)	3,790,503,000	4,600,626,333
Read one message from blockchain	722,070,000	702,579,000
Send one message to the blockchain	12,000,000,000*	12,000,000,000*

** This is based on the minimum network propagation delay of 12 seconds for a blockchain network to be fully decentralized [5]. Certain blockchain networks may offer delays with a centralization tradeoff. These blockchain networks may be utilized if their level of centralization is within acceptable bounds.*

TURNAROUND TIME

Send Message (16 seconds)

Action	Median Delay (s)
Generate symmetric key	1,000
Encrypt csv file (1MB)	14,887,000
Encrypt symmetric key	605,000
Hash csv file (1MB)	568,000
Pin File to IPFS (1MB)	3,480,857,000
Sign Hash	4,762,000
Send one message to the blockchain	12,000,000,000
TOTAL	15,501,680,000

Receive Message (5 seconds)

Action	Median Delay (ns)
Read one message from blockchain	722,070,000
Decrypt symmetric key	3,590,000
Retrieve File from IPFS (1MB)	3,790,503,000
Decrypt csv file (1MB)	10,771,000
Hash csv file (1MB)	568,000
Verify Hash	7,379,000
TOTAL	4,543,881,000

For a request and response, there are a total of 2 messages sent and 2 messages received, giving us a total of 42 seconds. Allowing for 10 minutes for the ML model to run and generate a response gives us roughly 15 minutes per round-trip. Communication and computation between the organizer and each participant can take place in parallel.

We can practically operate at a minimum interval of 30 minutes. In other words, we can operate strategies that expect to execute trades at 30-minute intervals.

PROJECT LINKS

The implementation code can be found in this Github repository: https://github.com/rocketcapital-ai/ip_preserving_ml_predictions

A detailed description of the implementation code can be found here: https://github.com/rocketcapital-ai/ip_preserving_ml_predictions/blob/main/docs/Implementation.pdf

5. BENEFITS

1. The solution allows us to organize competitions with higher frequency (up to twice per hour), and on demand. In our experience this generates interest, as new participants do not have to wait 1 week to start taking part. Instead, they can start participating almost immediately as well, whenever they are ready with their models. Data scientists are in general also keen to have their models utilized more frequently than once per week. These can have a knock-on effect that leads to an expansion of our crowd of data scientists, which is beneficial for the overall trading system.
2. The on-demand part allows for competitions where the organizer determines market opportunities in real time and asks participants to confirm or withdraw the opportunity based on the available data on market conditions. This is a real-time version of the Jane Street competition and could also apply the concepts like the ones exploited by the PredictNow company of Ernest Chan to a decentralized and certified setting.
3. The solution is similar to the Numerai process for model submission but is completely decentralized and gives much stronger guarantees by using smart contracts and persistent decentralized storage. Participants do not lose their intellectual property and we benefit from the intelligence of the crowds at a higher frequency than before. This helps us to compete, not just with other crowd-intelligence-based systems, but also with traditional asset managers that are operating at mid-frequency.
4. The solution could be generalized to different scenarios and applications apart from trading, where multiple actors must interact by exchanging messages with significant payloads with strong correctness requirements. Confidentiality can be maintained at the time while allowing for verification at a later time. By recording publicly at the time of communication and revealing it later, we are guaranteed that the original message has not been tampered with and is indeed what was passed between the original sender and recipient/s.

6. ADDITIONAL CONSIDERATIONS

MODEL SIGNATURES

It would be beneficial if a signature could be sent along with the output of a model to identify, and if possible, guarantee which model the output was generated from. A general scheme is presented here. This scheme is meant to be part of the protocol library distributed to participants.

REGISTRATION

1. The participant serializes their model into a byte string.
2. Compute the hash, H, of this byte string.

3. Register this hash with their wallet address on the smart contract.

PARTICIPATION

1. Read the request from the organizer using the above protocol.
2. Generate the response using the participant's model.
3. Serialize and compute the hash of this model, H'.
4. Send the hash along with the response back to the organizer.

The organizer should expect H to be the same as H' and reject the response otherwise. H equalling H' indicates that the model used to generate the response is the same as that registered by this participant. In subsequent responses, H equalling H' also indicates that the responses have been generated by the same model.

PUBLIC-PRIVATE KEY ROTATION

It is good practice to change cryptographic keys regularly. The smart contract can be re-organised such that it can accommodate the update of public keys associated with a wallet address, along with the ability for message receivers to identify which of their public keys has been used in the preparation of the message (and thus which of their private keys to use to retrieve the original message).

COST AND LATENCY REDUCTIONS

EVENT-TRIGGERED FUNCTIONS

With the current solution participants must use a server or a container always on to poll the blockchain and handle possible messages. A more efficient solution would be to use a Google cloud function (or AWS lambda), which is executed on demand. In this case the cost of the solution decreases considerably because the CPU is only used when the model is executed. However, the cloud function must be externally activated, periodically or on-demand, so the organizer must actively trigger the model execution after the request has been written on the blockchain; alternatively, cloud providers provide services to periodically trigger cloud functions. The triggered model first checks the requests assigned to it on the blockchain and does not proceed if it does not find any such requests. In this way the organizer is still unable to use model outputs outside of the competition.

COMPRESSION

In this implementation, we have not considered compression. Compression may be applied at various steps to lighten the payload and decrease the delay overhead. For example, the original message file might be compressed before encryption, or the encrypted file might be compressed before uploading it to the file system. More analysis is required to further understand the benefits of compression to this system.

BIBLIOGRAPHY

1. Rocket Capital Investment. "Scoring and Reward Policy.", https://docs.rocketcapital.ai/rci-competition/scoring-and-reward-policy.
2. Capoti, Davide, et al. "The Yiedl Project." Rocket Capital Investment, 13 April 2023, https://docs.rocketcapital.ai/rci-competition/.
3. Numerai Fund. "Model Uploads." Numerai Tournament Documentation. https://docs.numer.ai/numerai-tournament/submissions/model-uploads.
4. Swende, Martin Holst (holiman), Nick Johnson (arachnid@notdot.net). "ERC-191: Signed Data Standard." Ethereum Improvement Proposals, no. 191, January 2016.
5. Buterin, Vitalik. "Toward a 12-Second Block Time." Ethereum Blog, 11 July 2014, https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time.