

Incentive (INC)

Whitepaper V1.0

December 2022

Younes Hassani

younes@inctoken.org

Tarik Hassani

tarik@inctoken.org

This document describes the definitions and theory behind the INC token, explains the different uses, and focuses on the implementation of the INC Survey protocol.

1 CONTENTS

2	INTRODUCTION	3
2.1	BASIC CONCEPTS.....	3
3	PROTOCOL ARCHITECTURE.....	4
3.1	ENGINE	5
3.2	VALIDATOR	5
3.3	STORAGE	5
3.4	FORWARDER	5

3.5	RELAYER	6
4	THE SURVEYENGINE CONTRACT	6
4.1	ADD SURVEY	6
4.2	SOLVE SURVEY.....	7
4.3	ADD PARTICIPATION	8
4.4	ADD PARTICIPATION FROM FORWARDER	9
5	THE SURVEYVALIDATOR CONTRACT	10
5.1	CHECK SURVEY	10
5.2	CHECK RESPONSE	12
5.3	CHECK AUTHORIZATION.....	12
6	THE SURVEYSTORAGE CONTRACT	13
6.1	PUBLIC FUNCTIONS	13
6.2	OWNER FUNCTIONS	15
6.3	MANAGER FUNCTIONS	16
6.4	ADMINISTRATION FUNCTIONS	16
7	THE INCFORWARDER CONTRACT	17
8	THE RELAYER SERVER	17
8.1	IMPLEMENTATION	17
8.2	LIMITATIONS AND RISKS OF CONCURRENCY	19
9	DISCLAIMER	19

2 INTRODUCTION

Incentive (INC) is an [ERC-20](#) utility and governance token that was born to support a set of projects that allow the use of incentives to promote collaboration and cooperation, some completed and others in process. INC is built on **Polygon**, which addresses Ethereum's limitations, such as transaction speed, throughput, and gas fees.

Throughout this document we will use the **INC Survey** project as a reference.

2.1 BASIC CONCEPTS

INC Survey is a survey protocol and as such, allows on the one hand that the company creates a questionnaire asking the appropriate questions to collect opinions and promote the brand and on the other, that users participate and receive an incentive in exchange for their collaboration.

Surveys are vitally important for businesses; a clear understanding of customer needs is the key to growth.

You can do a survey to measure employee engagement, or a customer satisfaction survey to assess whether you provided good service. Whatever the case, the data obtained will provide valuable insights for decision making.

There is no doubt that surveys are powerful weapons for successful businesses, and they are even more so by taking advantage of the advantages offered by the blockchain, such as decentralization, immutability and transparency.

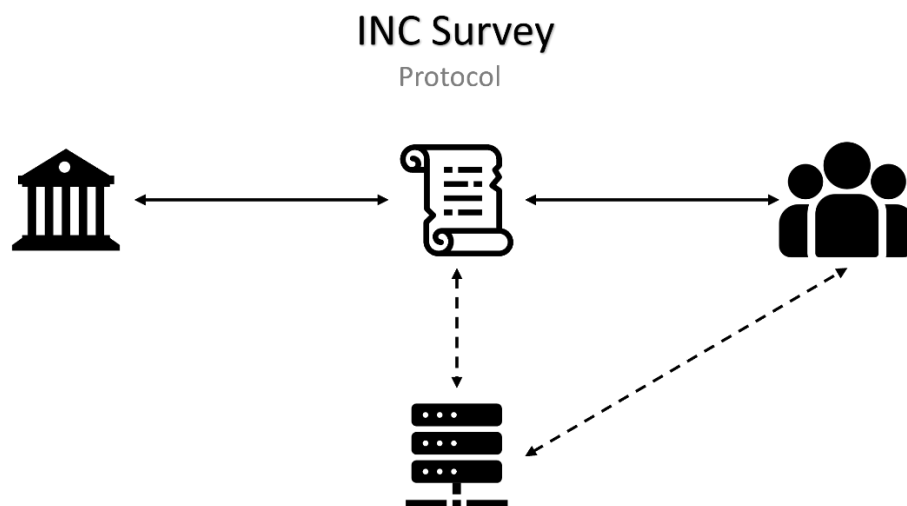


Figure 1: INC Survey Protocol

The figure above shows a simplified schema of the protocol, *INC Survey* has the following features:

- ✓ **Validate surveys:**

The parameters entered by the survey owner are checked to ensure that all requirements are met.

✓ **Validate participations:**

Each response is checked using the corresponding validator indicated by the survey owner.

- ✓ **Keep budgets and reserves:**

Once the survey is completed or resolved, the owner can recover the remaining budget and gas reserve.

- ✓ **Guarantee rewards:**

Once the participation is validated and registered, the user receives his reward immediately in his wallet.

- ✓ **Allow participation without gas:**

To participate in the surveys, it is not necessary to have MATIC in the wallet. Users can delegate transactions to the INC Relayer module that uses the gas reserve provided by the survey owner.

These features, among others, constitute (as far as we know) the first and only survey protocol on the blockchain where an ecosystem based on trust is forged, which benefits both brands and consumers.

3 PROTOCOL ARCHITECTURE

The current implementation of the protocol is as follows:

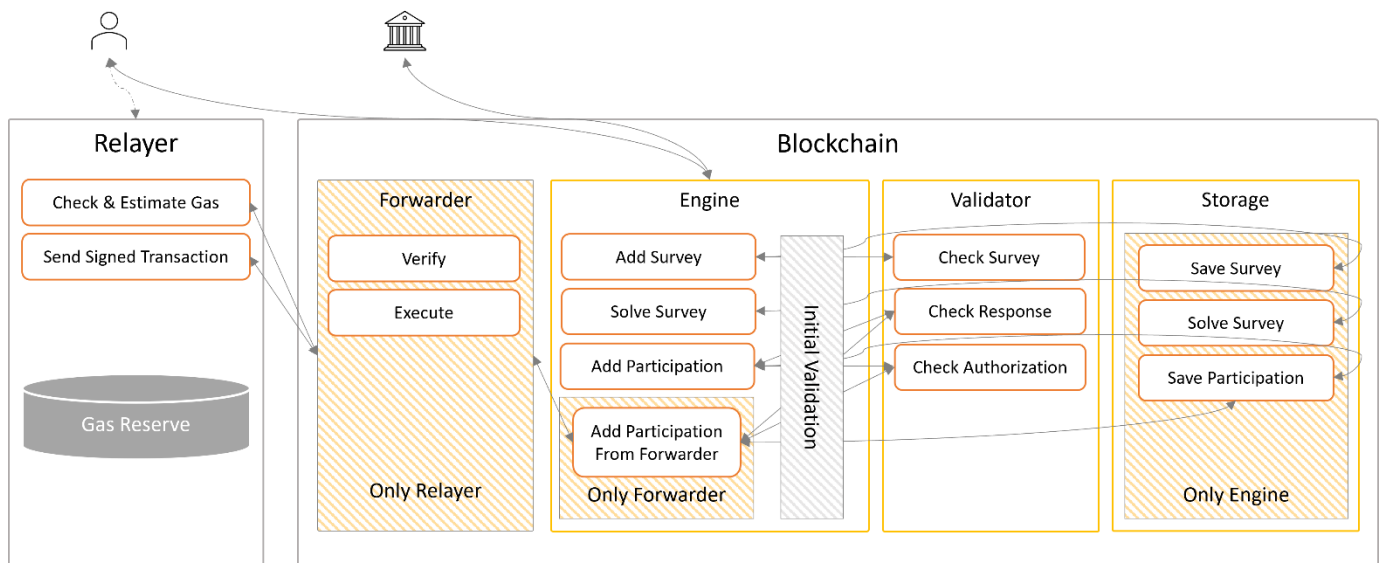


Figure 2: Protocol Architecture

To create a survey, the company must go to the Engine. However, the user can choose to delegate the operation to the Relayer and in this way, he does not need to have a balance in his wallet since he does not have to pay the gas fee.

As you can see in Figure 2, access to some features is limited:

- Only the Relayer can execute the meta-transactions in the Forwarder contract.
- Only the Forwarder can invoke the 'Add Participation From Forwarder' function in the Engine contract. This function, unlike 'Add Participation', receives the estimated gas to execute the meta-transaction and is responsible for updating the gas reserve in the survey.
- Only the Engine can invoke the registration functions in the Storage contract.

The protocol components are briefly described below before going into detail.

3.1 ENGINE

It represents the 'SurveyEngine' contract, is the main entry point to the protocol and is responsible for:

- Check and add surveys
- Resolve active surveys
- Check and add participations

3.2 VALIDATOR

Represents the 'SurveyValidator' contract, is responsible for checking the survey parameters to ensure consistency and validating the responses of participations through the validators indicated during the creation of the survey.

3.3 STORAGE

Represents the 'SurveyStorage' contract, is the warehouse of surveys and participations and provides all the necessary functions to access the stored data, such as:

- Get / Search for Surveys
- Get participations
- Get questions from a default survey
- Get answers to a default question
- Get information about the status of a survey

3.4 FORWARDER

Represents the 'INCForwarder' contract, is responsible for receiving the signed meta-transactions containing the participation transactions and executes them through a low-level call.

3.5 RELAYER

The Relayer is an external module that allows users to participate in surveys without having a balance in their wallet to pay for the gas of the transactions, since it is responsible for managing the gas reserves provided by the creators and sending the transactions to the Forwarder.

4 THE SURVEYENGINE CONTRACT

This contract is the manager of the validation and storage contracts, since only he can call the sensitive or updating functions that affect the surveys and participations in both contracts.

The functions of this agreement are described below.

Note: *The validations that are performed on the client serve to improve performance and avoid unnecessary calls to the blockchain, however, once the transaction is sent, they are checked again on the server or blockchain.*

4.1 ADD SURVEY

When a transaction is sent to create a new survey, the Engine checks certain parameters such as balance and budget and then invokes the validation contract that is responsible for validating the survey by checking all parameters, questions and answer validators. If everything is correct, it then invokes the storage contract to register the survey and finally executes the following transactions:

- . Transfer the budget tokens to this contract for safekeeping
- . Transfer the Engine fee to the beneficiary
- . Transfer the gas reserve to the Relayer to finance the participations

Before sending a survey to the blockchain, several checks are performed from the client, among which is the allowance check, since the Engine contract must be able to send the tokens on behalf of the creator to reward participants automatically.

So, if the creator hasn't approved this contract, for example, because it's their first survey. He will need to submit 2 transactions, the first to assign approval and the second to record the survey. Approval is assigned using the approval method of the ERC-20 standard.

The following flow chart shows the survey creation:

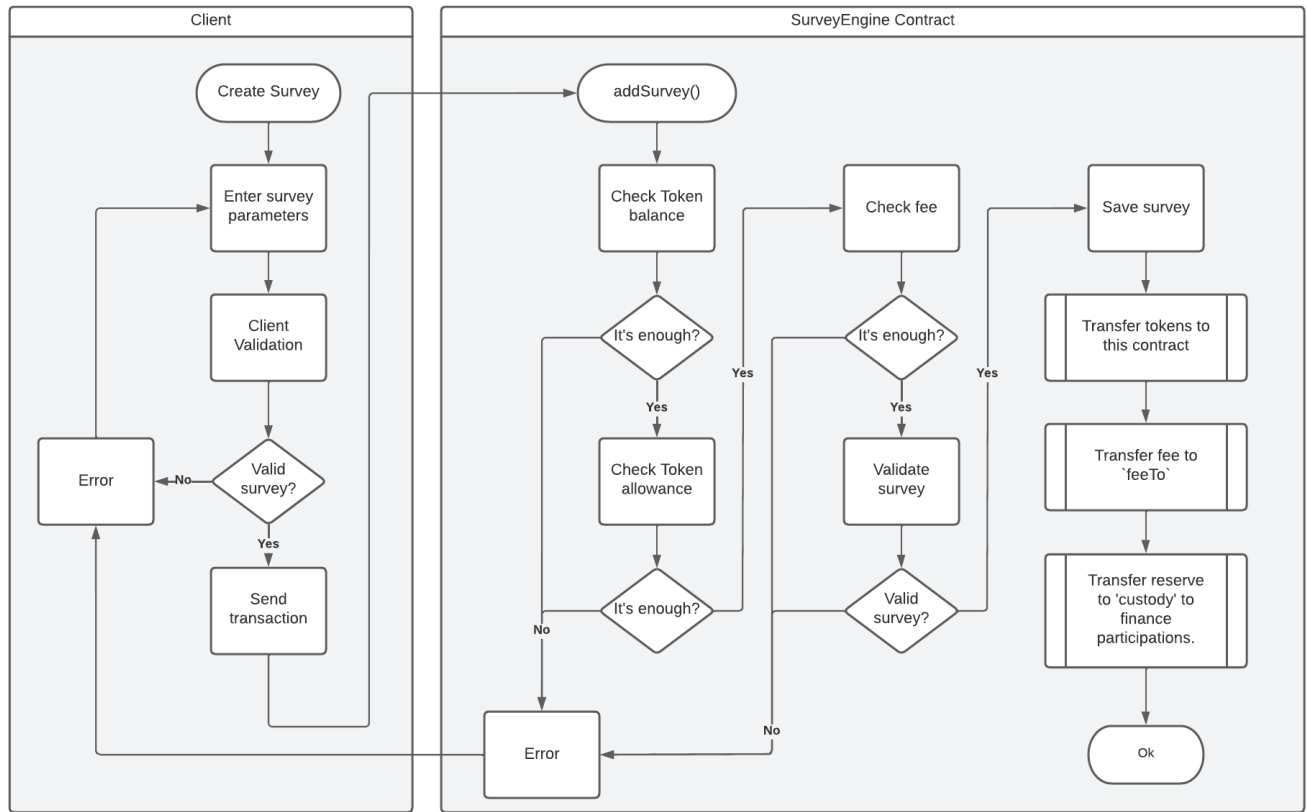


Figure 3: Add Survey

As can be seen in Figure 3, once the survey is registered, the transfers mentioned above are carried out.

It is worth mentioning that the transfer of gas reserve to the custody (Relayer) is carried out by means of the wrapped coin ([WMATIC](#)) instead of the native coin (MATIC).

The reason is that the transfer of the native currency cannot be delegated since it does not have the approval method.

Therefore, WMATIC is the envelope of MATIC and can be implemented as a smart contract to delegate transfers and facilitate returns.

4.2 SOLVE SURVEY

Resolving the survey consists of withdrawing the remaining budget and gas reserve, leaving the survey unusable. The action is defined as follows:

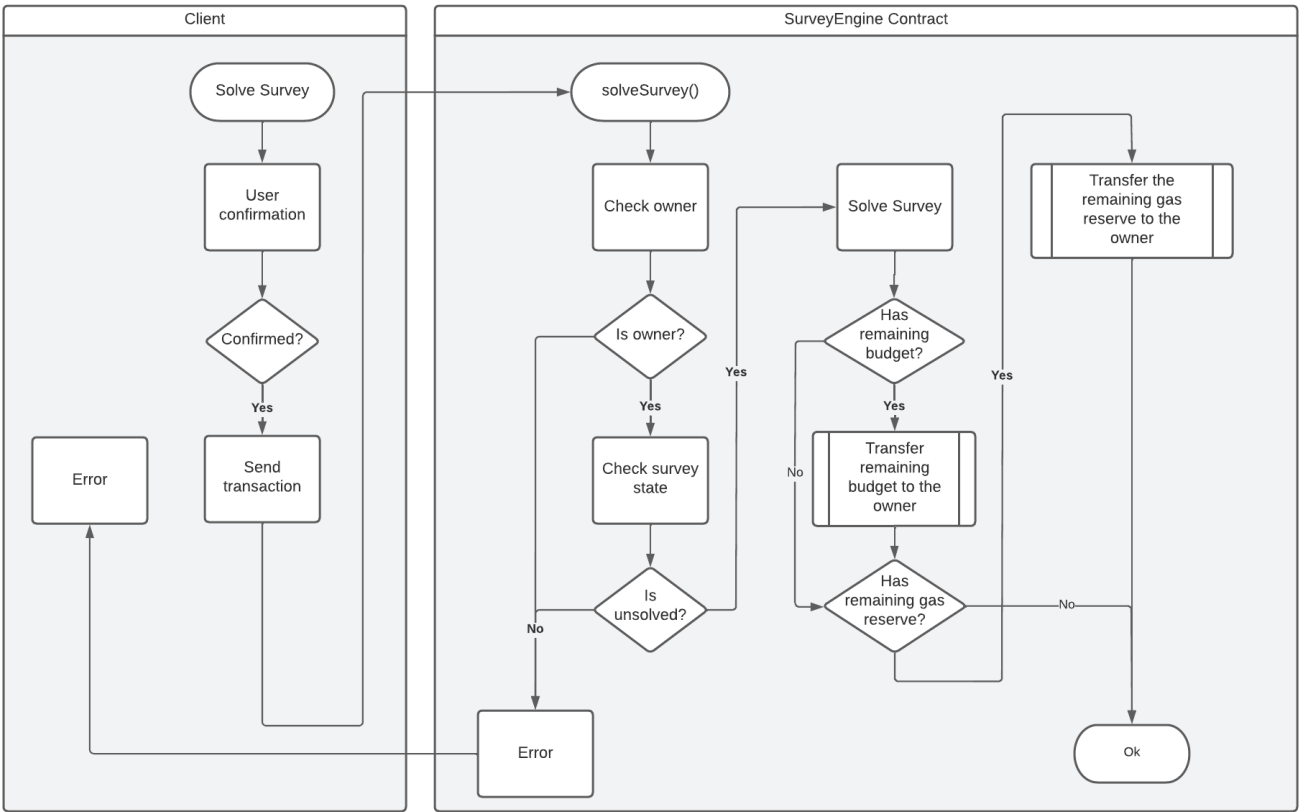


Figure 4: Solve Survey

As you can see in this diagram, the remaining budget and gas reserve are transferred to the owner of the survey.

Considering that the Engine has approval from the Relayer, to transfer the remaining gas reserve, the following steps are carried out:

- . Get the wrapped tokens (WMATIC) from the Relayer
- . Unwrap the tokens (convert them to MATIC)
- . Transfer the native currency to the owner of the survey

4.3 ADD PARTICIPATION

When a transaction is sent to add a new participation, the Engine checks that the survey is open and that there is enough budget for the reward, then it invokes the validation contract that is responsible for checking the authorization (if the survey requires a coupon) and to validate the type of answers and their content through the validators indicated by the creator. If everything is correct, it then invokes the storage contract to register the participation and finally transfers the reward tokens to the participant.

Participating by assuming the gas of the transaction, is relatively simple and is defined as follows:

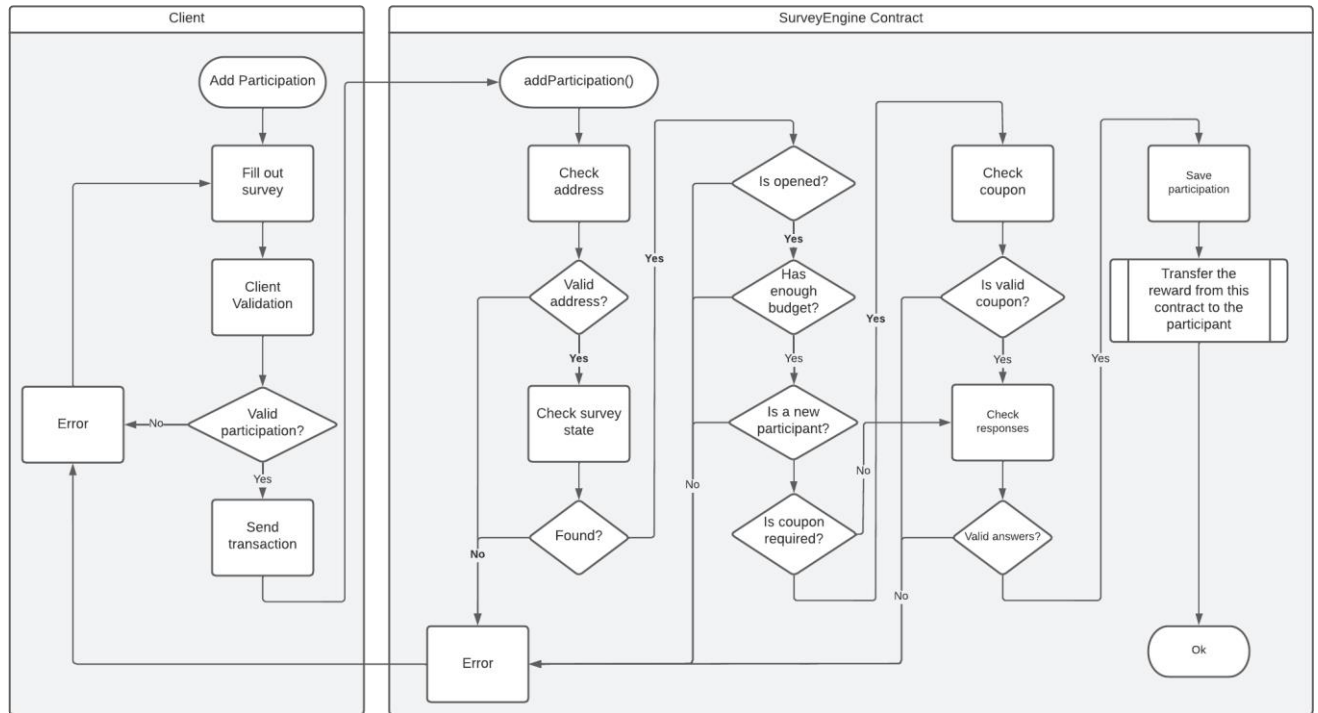


Figure 5: Add Participation

As can be seen in the diagram above, once the participation is validated and registered, the reward is immediately transferred to the user.

4.4 ADD PARTICIPATION FROM FORWARDER

To participate through meta-transaction (without assuming the gas), the Relay and the Forwarder come into play, since this function is only accessible from the Forwarder contract which, in turn, is only accessible from the Relay address. The action is defined as follows:

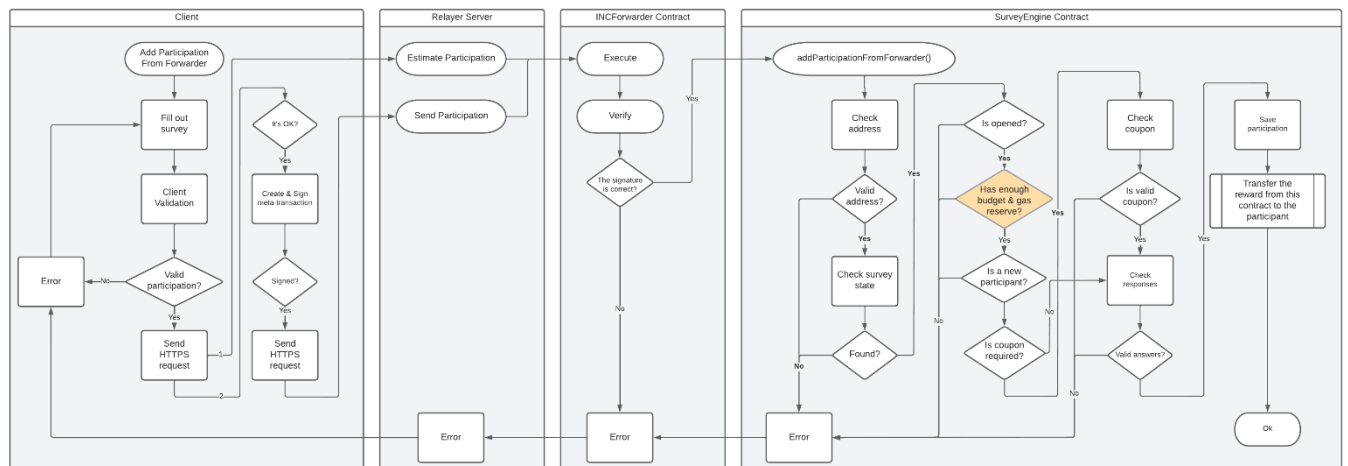


Figure 6: Add Participation From Forwarder

As you can see in the Engine contract, this time, the gas reserve is also checked.

The client must always call the first method (Estimate Participation) before executing the meta-transaction (Send Participation), since information needed by the second method to verify the integrity of the request is saved.

The first call made through the Relayer (Participation Estimation) against the blockchain is simply to obtain the gas quote that the user must sign along with the meta-transaction.

During the second call (Send Participation), the Relayer re-executes the estimate with the user data to detect errors and confirm the availability of sufficient gas reserves before executing the meta-transaction.

Once the participation is validated and registered, the reward is immediately transferred to the user.

5 THE SURVEYVALIDATOR CONTRACT

For its part, the validation contract defines a series of parameters that are used during the different validations that are managed by the developers to maintain an optimal configuration:

titleMaxLength: Maximum survey title size, default value 128

descriptionMaxLength: Maximum survey description size, default 512

urlMaxLength: Maximum survey icon URL size, default 2048

startMaxTime: Maximum time to start the survey, default 864000 seconds

rangeMinTime: Minimum survey duration, default 86400 seconds

rangeMaxTime: Maximum survey duration, default 2592000 seconds

questionMaxPerSurvey: Maximum number of questions, default value 100

questionMaxLength: Maximum content size of each question, default value 4096

validatorMaxPerQuestion: Maximum number of validators per question, default value 10

validatorValueMaxLength: Maximum value size of each validator, default value 128

hashMaxPerSurvey: Maximum number of coupons per survey, default value 1000

responseMaxLength: Maximum content size of each response, default value 2048

The most important functions of this contract are described below.

5.1 CHECK SURVEY

Using the parameters indicated above, the following validations are performed:

- **Validate title:**
The title is mandatory for what is verified that it has been indicated and that it does not exceed the maximum size titleMaxLength.
- **Validate description:**
The description is optional, if it is verified that it does not exceed the maximum size descriptionMaxLength.
- **Validate logo URL:**
The logo URL is optional, if it is checked that it does not exceed the maximum size urlMaxLength.
- **Validate date range:**
 - The start date must be greater than or equal to the current date, less than the end date and less than the maximum time to start the survey (Current time + startMaxTime).
 - The duration (End date – Start date) must be greater than or equal to rangeMinTime and less than or equal to rangeMaxTime.
- **Validate budget:**
It is verified that the budget is greater than 0.
- **Validate reward:**
It is verified that the reward is greater than 0 and less than or equal to the budget.
- **Validate participations number:**
The number of participations must be a positive integer (budget%reward == 0)
- **Validate hashes to check coupons:**
Hashes are used to check coupons during participations.
If hashes are sent, the size of each one is checked, and the total should not exceed the maximum hashMaxPerSurvey.
- **Validate number of questions:**
The number of questions must not exceed questionMaxPerSurvey.
- **Validate questions:**
The content of the questions is a json that contains information about the question and the type of component used in the view.
In addition, it is checked that the total size does not exceed the maximum defined questionMaxLength.
- **Check validators number:**
The number of validators for each question must not exceed the maximum defined validatorMaxPerQuestion.

- **Check validators:**

Validators, as their name indicates, are used to validate the associated question.

Each validator contains the following parameters:

- **questionIndex:** Index of the question
- **operator:** Validation operator, used to concatenate with the following validator (And, Or)
- **expression:** Validator type, Equals, Contains, Greater, ...
- **value:** Validation value, Ex. a validator of type 'NotContains' can define the 'Hitler' value to prevent the response from containing this value.

Validator checking consists of checking the type (expression) and size of the value so that it does not exceed the maximum defined validatorValueMaxLength.

5.2 CHECK RESPONSE

The first thing that is checked is the size of the response and this should not exceed the maximum defined responseMaxLength.

Next, the type of answer is checked, as it must match the type required by the question.

Finally, the validators indicated during the creation of the survey are used, each question can have one or more validators. The validators associated with each response are traversed and the defined rules are verified.

5.3 CHECK AUTHORIZATION

Surveys can be open (for the public) or closed (require coupon).

That is, to participate in a closed survey a valid coupon provided by the creator is required.

During the participation validation, it is checked if the survey requires a coupon and if so, then it is verified that the user has indicated a valid coupon.

To do this, you get the hash corresponding to the coupon and check that the survey contains this hash.

To improve performance, the survey only saves the first and last 4 characters of each hash.

Ex. the client does not need to send 100 hashes of this size to the blockchain.

```
0x8e6eeb2804809b68e5df8ee409768b6adcb309364cb4a422dbb0d4a540f84f84
```

```
..
```

Instead, you should only send the first and last 4 characters of each hash by discarding '0x'.

```
8e6e4f84
```

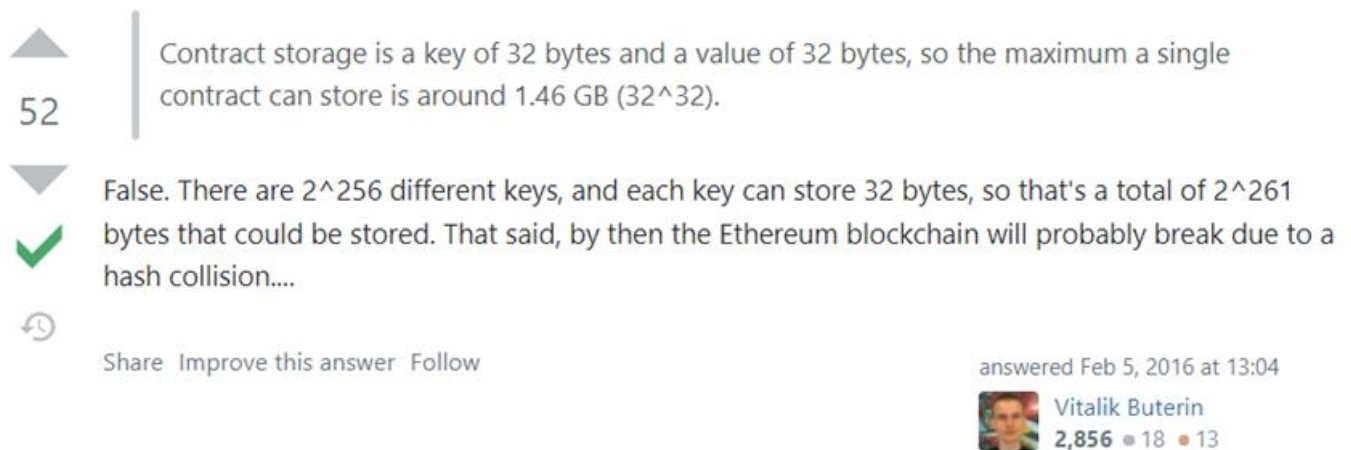
6 THE SURVEYSTORAGE CONTRACT

As mentioned above, this contract is the warehouse of surveys and participations. However, this contract only stores the addresses of surveys and participants.

The storage contract has a factory for survey contracts. **For each survey, a unique contract is created** that stores the general data, the questions, the validators, and the answers.

The maximum that a single smart contract can store is around 2^{261} bytes, so there is no need to worry about storage.

The response of Mr. Vitalik Buterin in [StackExchange](#), confirms this information.



The screenshot shows a StackExchange post with a score of 52. The question is: "Contract storage is a key of 32 bytes and a value of 32 bytes, so the maximum a single contract can store is around 1.46 GB (32^{32})." The answer, marked with a green checkmark, is: "False. There are 2^{256} different keys, and each key can store 32 bytes, so that's a total of 2^{261} bytes that could be stored. That said, by then the Ethereum blockchain will probably break due to a hash collision...." The answer is by Vitalik Buterin, dated Feb 5, 2016 at 13:04, with 2,856 upvotes, 18 downvotes, and 13 comments.

However, it is very important to consider performance, which is why the number of items per request is limited to all functions that return a set of items.

This type of function requires a cursor and an element number that does not exceed the established limit. The storage contract defines a series of parameters to validate requests:

surveyMaxPerRequest: Maximum number of surveys per request, default value 100

participantMaxPerRequest: Maximum number of addresses per request, default value 100

participationMaxPerRequest: Maximum number of participations per request, default value 100

questionMaxPerRequest: Maximum number of questions per request, default value 100

responseMaxPerRequest: Maximum number of responses per request, default value 100

The most important functions of this contract are described below.

6.1 PUBLIC FUNCTIONS

Public functions are those that anyone can consult, these are the most outstanding:

- **Current Cursor**

Since the surveys have a maximum duration, to facilitate the search, this contract returns the current cursor that covers only active surveys (those that enter the period).

The period is from (CurrentTime - MaxDuration) to CurrentTime.

- **Remaining Budget Of**

Returns the remaining tokens budget in the survey.

- **Gas Reserve Of**

Returns the remaining gas reserve in the survey.

- **Key Required Of**

Indicates whether participation in the survey requires a key (coupon provided by the creator).

- **Owner Of**

Returns the address of the survey owner.

- **Get Surveys**

Start from the indicated cursor and return the required number of surveys at most. The number of surveys must not exceed the maximum per request defined by surveyMaxPerRequest.

- **Find Surveys**

Performs the survey search from the indicated cursor and returns a maximum of the required number of surveys. This number must not exceed the maximum per request defined by surveyMaxPerRequest.

To do this, the SurveyFilter object that contains the following parameters is used:

- search: Text to search for in the title or description
- onlyPublic: Only public surveys (which do not require a coupon)
- withRmngBudget: With a budget greater than or equal to the reward
- minStartTime: Minimum startup time
- maxStartTime: Maximum start time
- minEndTime: Minimum end time
- maxEndTime: Maximum end time
- minibudget: Minimum budget
- minReward: Minimum reward
- minGasReserve: Minimum gas reserve

- **Get Participants**

Start from the indicated cursor and return at most the required number of addresses that have participated in the survey. The number of addresses must not exceed the maximum per request defined participantMaxPerRequest.

- **Get Participations**

Start from the indicated cursor and return the required number of participations at most. This number must not exceed the maximum per request defined participationMaxPerRequest.

- **Find Participation**

Look for a participation in the survey with the address indicated.

- **Get Questions**

Start from the indicated cursor and return the required number of questions at most. This number must not exceed the maximum per request defined questionMaxPerRequest.

- **Get Responses**

Start from the indicated cursor and return at most the required number of answers that correspond to the question. This number must not exceed the maximum per request defined responseMaxPerRequest.

- **Get Validators**

Returns the validators of the indicated survey and question.

6.2 OWNER FUNCTIONS

These functions are also public; however, they only return information associated with the address making the call. The highlights are described below:

- **Get Own Surveys**

It starts from the indicated cursor and returns at most the required number of surveys that belong to the address of making the call. The number of surveys must not exceed the maximum per request defined by surveyMaxPerRequest.

- **Find Own Surveys**

Performs the search for surveys that belong to the address of the call, from the indicated cursor and returns a maximum of the required number of surveys. This number must not exceed the maximum per request defined by surveyMaxPerRequest.

This uses the SurveyFilter object in the same way as the 'Find Surveys' function.

- **Get Own Participations**

It starts from the indicated cursor and returns at most the required number of participations that belong to the address of making the call. The number of items must not exceed the maximum per defined request participationMaxPerRequest.

- **Find Own Participation**

Look for a participation in the survey with the address making the call.

6.3 MANAGER FUNCTIONS

These functions are only accessible from the Manager (Engine contract). The highlights are described below:

- **Save Survey**

Save the survey data and returns a unique identifier.

- **Save Participation**

Save the participation data.

- **Solve Survey**

Solve the survey, zeroing the budget and the gas reserve.

6.4 ADMINISTRATION FUNCTIONS

The management functions are only accessible by the developers and in this case, they serve to limit the number of items per request.

- **Set Survey Max Per Request**

Assign the maximum number of surveys per request.

- **Set Participant Max Per Request**

Assign the maximum number of participants per request.

- **Set Participation Max Per Request**

Assign the maximum number of participations per request.

- **Set Question Max Per Request**

Assign the maximum number of questions per request.

- **Set Response Max Per Request**

Assign the maximum number of responses per request.

7 THE INCFORWARDER CONTRACT

The Forwarder contract is the meta-transaction receiver, developed under the [EIP-712](#) standard and extended to limit the execution of transactions, since only the assigned Relayer can invoke the 'Execute' method.

This contract is the only one that can execute the 'Add Participation From Forwarder' method in the Engine contract to record non-gas participations, receives signed meta-transactions containing the participation transactions, and executes them via a low-level call.

8 THE RELAYER SERVER

The Relayer is an external module that manages all the participation requests without gas, since it is the only one that can execute meta-transactions in the Forwarder contract.

For security, the CORS configuration rejects requests of unknown origin, currently, only the protocol client 'https://survey.inctoken.org' can send requests to this server.

In addition, the reCAPTCHA security mechanism is used to detect traffic from automated programs or bots.

8.1 IMPLEMENTATION

This is the current implementation:

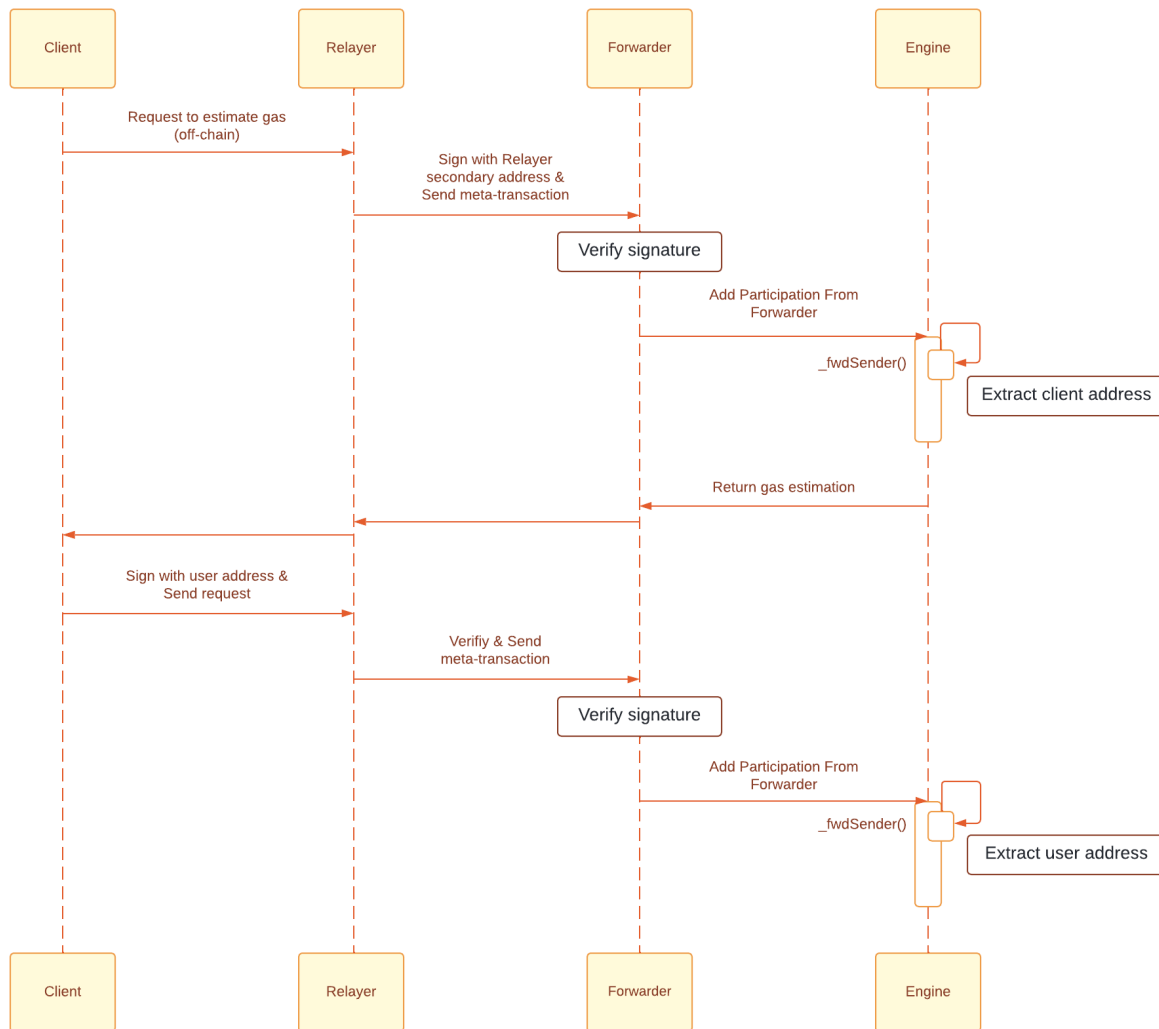


Figure 7: Add Survey

As you can see, during the first request, the client is responsible for obtaining a gas estimate before sending the meta-transaction signed by the user.

This estimate has a margin of 20% to guarantee execution and is the one that is sent in the meta-transaction so that it is finally subtracted from the associated reserve.

In general, the second request will take a while to execute, in addition, the user will take a while to sign the meta-transaction, therefore, the Relay always re-estimates the gas to check the feasibility and confirm the reserve availability before executing the meta-transaction.

This second estimate is also used to check the difference in the gas fee, if it varies by more than 10%, the meta-transaction will not be executed.

For security, the repeater saves a one-time hash between the first and second calls, so the meta-transaction cannot be executed without estimating first. This hash allows you to confirm that the participation has not been modified.

The user's session lasts for 30 minutes, if it is lost, it must start the operation again.

For its part, the Forwarder contract is responsible for verifying the identity of the sender before executing the participation transaction.

8.2 LIMITATIONS AND RISKS OF CONCURRENCY

The current technique may cause gas debris to remain in the Relayer due to the difference between the estimate and the actual gas used, but on the other hand, the Relayer must assume the expense of failed transactions due to concurrency in the cluster.

Ex.

Suppose a survey has sufficient gas reserve for a single participation.

If 2 users simultaneously submit the request to participate for the same survey, both may pass the Relayer checks, so 2 transactions will be sent to the blockchain.

In this case, only one transaction will execute and the other will fail when the reservation is verified in the Engine contract.

9 DISCLAIMER

This document is for general information purposes only, it does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation to make an investment decision.

He should not be relied upon for accounting, legal or tax advice or for investment recommendations.

This document reflects the current opinions of the authors and is not made on behalf of Incentive Token LLC or its affiliates and does not necessarily reflect the opinions of Incentive Token LLC, its affiliates or persons associated with them.

The opinions reflected herein are subject to change without being updated.