Whitepaper PROVER

1. Introduction

The PROVER service is designed for objective and independent verification of video content. PROVER is an online service that allows receiving confirmation of the occurrence of events on video, to confirm the authenticity of the creation of video materials using blockchain technology. PROVER is intended for both offline uses and for embedding in third-party platforms and services for developing applications based on its functionality.

2. Industry overview

The revolution of smartphones, which started 10 years ago with the advent of the iPhone and tablet computers, had a significant impact on the ways users communicate in the network. Everyone suddenly became the creator and distributor of content, causing, as a consequence, his exponential growth. This has significantly changed and continues to change many sectors of the economy, launching the processes of digitizing the world around us. Photo and video content are actively used not only for entertainment and educational purposes but also for other needs, including economic and legal nature - financial, insurance, judicial, medical and other services. In this regard, there is a serious need for an independent, decentralized service that objectively guarantees the authenticity of the created video content and protects it from possible forgery and unfair editing.

The solution of such a problem can give a serious impact on the development of the digital economy. The technology can be used in thousands of projects from dozens of different areas and can help hundreds of millions of users around the world.

3. Problems

The authenticity of digital video recordings of events and facts of commercial and legal value is often questionable because video files can be edited for forgery purposes, by using a virtual camera (emulator). Attributes such as the date of the video could also be artificially altered.

The task of the PROVER service is to guarantee the authenticity of recording a video file with reference to a specific time of video shooting.

4. Existing solutions

4.1. Online Content Protection Solutions

The authenticity of the content and the rights to it (copyright) today are mostly being approved at the level of platforms for storing and demonstrating video materials. An example of such a solution is Google content ID, which allows only to confirm the time of downloading a video on YouTube, this is the basis for the assumption of its originality, based on the

principle of presumption of authorship (a person is considered as an author until the real author disputes this fact).

The drawback of this and similar solutions is that they do not allow to restore the time of real video recording, its originality, and integrity. In addition, these solutions work within their platforms "only on YouTube", are provided manually by service administrators upon application and always depend on the opinion of the service administrators. That is, there is always a place for the human factor - subjectivity or simply error. And with annoying truth seekers, the conversation is short - "Content owners who repeatedly filed unreasonable complaints can lose the right to use Content ID and lose their YouTube partner status."

The authors try to protect the content by watermarks and logos on the video screen, but this can only help them in the subsequent contestation of its authorship, but not to prove the fact of forgery. That means that in legal and financial matters this approach is completely inapplicable.

4.2. Solution for protecting copyright on blockchain

At the present moment, the blockchain community already has various online electronic notary services, which make possible to certify the existence of "Proof of existence" and the authorship of "Proof of ownership" of all kinds of files and documents and digital content:

- Block Notary - a service that helps to create "Proof of existence" of any content (photos, files, any media) using the TestNet3 or Bitcoin network. The frontend system is a mobile application for iOS that registers a document hash in a blockchain;
- Emercoin DPO Antifake - Technology based on the Emer platform allows to create for the item (product) a unique digital passport stored in a decentralized database - blockchain, and provides services for managing this passport. It is focused mainly on the offline segment - it helps to register individual details (VIN, IMEI numbers) in the system to the shield of real goods from fraud;
- Stampery - blockchain technology, that can verify e-mail or any files. This simplifies the process of verifying letters by simply sending them by e-mail to a specially created mailing address for each customer. Law firms use Stampery technology for a very cost-effective way of document verification;
- https://www.ascribe.io/ - Service for registration of copyright, further control and distribution of digital content. It is positioned for digital works of art. It offers to register a work, put it on sale in a secure marketplace, and then monitor its use (demonstration);
- https://letsnotar.me - an easy service that automatically stores to blockchain a hash of files that were uploaded in it. It could launch on a smartphone, allows to get access the camera, take photos and videos and hashing them. However, it can not guarantee that the video is recorded from a real, not a virtual camera - so it does not protect against forgery.

All these services unite one thing: they can assure a hash of a file that has already been written before or supposedly written down from the camera of the device, but they can not guarantee its originality, integrity, and authenticity of the video. They do not protect against forgery and unfair editing because they do not have the appropriate technology to verify the video content being created. PROVER technology allows guaranteeing that this content was

created at a specific time and place, from the camera of a particular device, to ensure that there are no signs of forgery and editing.

PROVER can become a functional addition, extending the capabilities of the services discussed above and the level of trust to them. Having the opportunity to 100% guarantee the authenticity of the created video file, these services will, for example, provide services for notarization of the authenticity of video statements.

To date, in the world, we have not been able to find information about the project involved in the verification of video at the stage of its creation. This is not surprising because to solve this problem it is necessary to have a unique set of competencies both in the field of blockchain systems and in the development of video processing modules (video analysis). Our strength in our team, because we combine all the necessary unique competencies and have extensive experience in these rarely overlapping areas.

5. What is PROVER?

The PROVER service consists of several components:

- A mobile app that installs on a smartphone and launches with the camera turned on or initiates the launch of the camera itself.
- A set of algorithms and utilities for integrating PROVER technology into third-party solutions and services.
- Smart Contract PROOF (only for implementation based on the Ethereum platform).

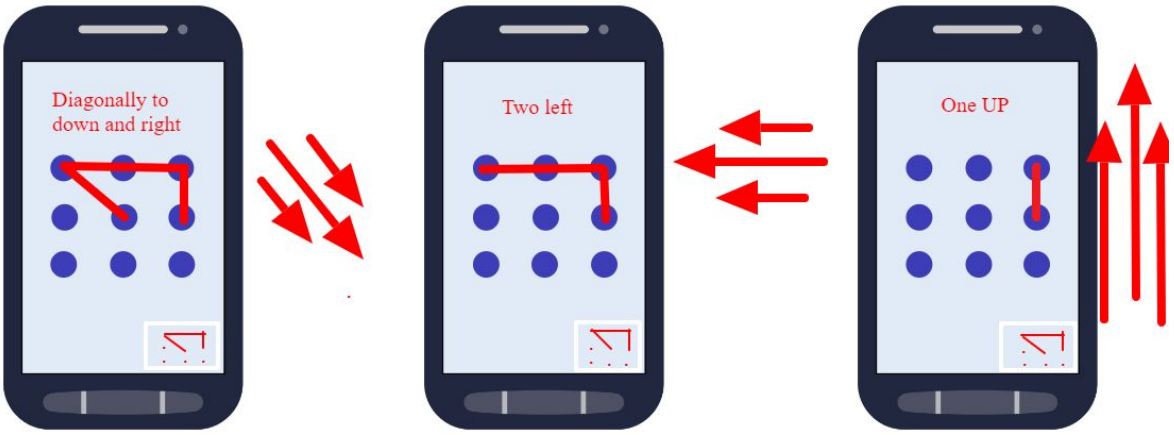
The main scientific and technical problem, the solution of which is the heart of the project, is the verification of the video footage shot by the user. During verification, the system confirms that:

- Video footage is produced by a real video camera integrated into the mobile device, and not emulated by a virtual video camera;
- The video material is complete, not edited, without gluing and insertions;
- The record was made in a certain period of time.

All source codes of the project will be placed in the public repository at https://github.com/isvirin/prover.

5.1. Verification with swype code

At the base of video verification is the algorithm for automatically detecting of swype code in a video stream. Instead of the classical swype code, which is being entered by moving user's finger on the touch screen, forming a continuous line connecting the points, shown on the screen, in PROVER technology, the swype code is being entered by moving the smartphone with the camera in a record mode. The video analytics module, which will be developed in the first stage of the project, analyzes the video stream from the smartphone camera, will be able to detect the direction of movement of the smartphone, and therefore build a virtual line of swype code that must intersect the virtual points in the required sequence in the field of three lines by three points in each.



In other words, when the smartphone camera is turned on, the user sees the "Enter swype code" button on the screen, clicking on which, the user sees a grid of 9 points by 3 points in a row, a swype code for input generated in a blockchain, and the current point from which the input of the swype code begins. To enter the swype code, the user needs to move the phone in a space so as to the virtual swype code line travels along the required path. After the code is entered, the grid disappears and a notification appears that the code is entered correctly.

Using the automatic algorithm for recognizing the swype code will allow the user at the stage of video recording to be sure that afterward, later, if that the video will need to be checked for authenticity, this swype code will be recognized by the service.

5.2. Verification using sensors

A stream of metadata (data from all sensors available in mobile device, with the maximum frequency - an accelerometer, a gyroscope, a magnetometer, GPS coordinates, etc) will be recorded in parallel with the video file to prevent forgery. Later, there will be developed a mathematical algorithm, that could allow restoring this information in the most precise details of the movements of the mobile device in the hands of the user to compare these movements with the video.

Here we will describe the mathematical model of positioning in space based on a three-axis accelerometer and a three-axis magnetometer under conditions of displacements with insignificant accelerations.

To determine the position of the object in space, we introduce the global three-dimensional cartesian coordinate system $OXYZ$, so that the axis OZ, coincided in direction with the direction of gravity field power lines $\vec{g}$, and the axis OY, coincided with the declination of the vector of the magnetic field of the planet $\vec{h}$.

To describe the position of the sensor in the global coordinate system (GCS) we introduce a local coordinate system (LCS), whose axes will coincide with the corresponding axes of the acceleration sensors and the magnetic field. Then the position of the LCS (sensors) in the GSC can be described by four vectors:

$\vec{r_{LCS}}$ – the displacement vector of the origin of the LCS relative to the GCS origin;
$\vec{i_{LCS}}$, $\vec{j_{LCS}}$, $\vec{k_{LCS}}$ – directing vectors of the orthonormal basis of LCS expressed in terms of the directing vectors of the orthonormal GCS basis.

Such description gives complete information about the orientation of the LCS in the GCS in coordinate form. The problem of determining the angular orientation reduces to finding the coordinates of the vectors $\vec{i_{LCS}}$, $\vec{j_{LCS}}$, $\vec{k_{LCS}}$ in GCS.

Due to the fact that the gravitational field is more stable than the magnetic field, let us take as a basis the acceleration sensor. The acceleration sensor readings are the coordinates of the acceleration vector of the free fall, decomposed along the axes of the LCS:
$\vec{a_{LCS}} = \{a_x, a_y, a_z\}$

The normalized vector $\vec{a_{LCS}}$ is nothing else but a vector $\vec{k_{GCS}}$, i.e., the defining vector of the OZ GSK axis, expressed in terms of the directing vectors of the LCS:
$\vec{k_{GCS}} = \frac{\vec{a_{LCS}}}{|\vec{a_{LCS}}|} = \{k_x, k_y, k_z\}$

The readings of the magnetic field sensor are the coordinates of the magnetic field vector, decomposed along the LCS axes:
$\vec{m_{LCS}} = \{m_x, m_y, m_z\}$

Vector $\vec{m_{LCS}}$ in the general case, it may not be parallel to the vector $\vec{k_{GCS}}$, therefore, it needed to get its normal component:
$\vec{n_{LCS}} = \vec{m_{LCS}} - (\vec{m_{LCS}} \cdot \vec{k_{GCS}}) \cdot \vec{k_{GCS}}$

The normalized vector $\vec{n_{LCS}}$ is nothing else but a vector $\vec{j_{GCS}}$, i.e. The defining axis vector OY GCS, expressed through the directing vectors of the LCS:
$\vec{j_{GCS}} = \frac{\vec{n_{LCS}}}{|\vec{n_{LCS}}|} = \{j_x, j_y, j_z\}$

The defining axis vector OX GCS, expressed through the directing vectors of the LCS, is found using the vector product:
$\vec{i_{GCS}} = \vec{j_{GCS}} \times \vec{k_{GCS}} = \{i_x, i_y, i_z\}$

We compose the matrix of the row represented by the vectors $\vec{i_{GCS}}$, $\vec{j_{GCS}}$, $\vec{k_{GCS}}$ then transpose it and expand it into vectors (also in rows):
$|i_x\ i_y\ i_z\ j_x\ j_y\ j_z\ k_x\ k_y\ k_z\ |^T = |i_x\ j_x\ k_x\ i_y\ j_y\ k_y\ i_z\ j_z\ k_z\ |$

Thus, the vectors:
$\vec{i_{LCS}} = \{i_x, j_x, k_x\}$
$\vec{j_{LCS}} = \{i_y, j_y, k_y\}$
$\vec{k_{LCS}} = \{i_z, j_z, k_z\}$
determining the axis of the LCS in the GCS, in other words, determine the orientation of the LCS in the GCS.

Let's express the vector $a_{\vec{LCS}}$ in GCS:

$$\vec{a} = a_x \cdot \vec{i_{LCS}} + a_y \cdot \vec{j_{LCS}} + a_z \cdot \vec{k_{LCS}} = \{a'_x, a'_y, a'_z\}$$

Vector readings $\vec{a}$ given in the quanta of the ADC of the acceleration sensor, for further calculations we translate them into the International System of Quantities (ISQ) and write the instantaneous acceleration vector:

$a_{\vec{phys}} = \frac{range}{N} \cdot \vec{a}$,

where the “range” is the range of the analog-to-digital sensor converter, and N is the division price.

To take into account the gravitational field, it is necessary to reduce the vertical component by the value of the acceleration of gravity:

$$\vec{a_g} = a_{\vec{phys}} - \{0, 0, -g\}$$

Let's move from acceleration to speed:

$$\vec{v_g} = \int \vec{a_g} dt + \vec{v_0} \approx \sum \vec{a_g} \Delta T + \vec{v_0}$$

Let's move from speed to coordinates:

$$\vec{r_g} = \int \vec{v_g} dt + \vec{r_0} \approx \sum \vec{r_g} \Delta T + \vec{r_0}$$

Thus, the proposed mathematical model makes it possible to determine the orientation of the sensor relative to the global coordinate system associated with the physical features of the planet from sensor readings, to determine by linear integration the linear movement of the sensor in the global coordinate system, and consequently, from the recorded measurement sequence, to reconstruct the trajectory of the movement of the user's personal mobile device.

6. How it works

6.1. General algorithm

The user installs the mobile app PROVER on his smartphone, grants it access rights to the camera, which allowed the app to start camera automatically.

When the camera is turned on, the user's smartphone accesses the blockchain via the Internet to obtain a swype code as a verification task that the user must perform while recording video data. Entering the swype code is carried out by moving the smartphone with the video recording mode turned on in the space of virtual points 3 by 3 points by the randomly generated trajectory (in blockchain or on the basis of data from the blockchain).

The user performs a video recording using the camera of his smartphone and at any time of the recording performs "input" of the swype code. When entering the swype code, the user can see hints in the form of virtual dots 3 to 3 and his detected movement of the smartphone. For additional confirmation of reliability, synchronous video recording of a sequence of measurements from smartphone sensors (an accelerometer and a magnetometer).

After the video is completed, the hash of the video file is calculated. The resulting hash is stored in the blockchain.

Recorded video is stored by the user on a personal device or cloud disk and can be presented for authentication if necessary.

Thus, in blockchain in one form or another, the following information is available:

- Date and time of receipt of the individual swype code;
- The generated swype code;
- A hash of a video data file (the video data file itself is stored by the user);
- Date and time of loading the hash.

To verify the authenticity of the needed video file, performs a file hash calculation, which retrieves from the blockchain, the stored there, information, if it is not there, notifies that the hash has not been uploaded. On the basis of the received information, it is concluded that the file with this hash was created not earlier than the time of receiving the individual swype code and not later than the time of loading the hash in the blockchain.

The user's video file is visually reviewed for continuity to exclude video editing and confirm the presence on the video the swype code, issued to the user as a verification task. After completing the first stage of the PROVER project, the presence of the swype code on the video will be determined by the automatic algorithm. After the second stage of the PROVER project, the continuity of the video recording will be determined by the automatic algorithm. If existing continuous video record is containing the same code as generated from the blockchain swype code, there is a reason to made the conclusion that the record was made by the user not earlier than when the swype code was issued. Herewith the moment when the swype code is issued is fixed in the blockchain and is unavailable for modification by the user.

After completing the third stage of the PROVER project will be available an algorithm of reconstruction the path of the personal mobile device in the user's hands during video recording and compare the reconstructed trajectory with the recorded video data. If the video recording and the restored trajectory of the personal mobile device coincide, there is made the conclusion that it is unambiguously confirmed that the recording was made from a real camera embedded in a personal mobile device, and not emulated.

6.2. Implementation for Ethereum

The implementation for the Ethereum platform is carried out on the basis of the PROOF Smart Contract. The mobile application of the user accesses via the Internet to the PROOF Smart Contract to obtain the swype code. PROOF Smart Contract captures the issued one-time swype code and the time it was issued.

After the video is completed, the hash of the video file is calculated. The received hash is sent to the PROOF Smart Contract, where it is stored.

The PROOF Smart Contract saves the following information:

- Date and time of issue of the individual swype code;
- The issued swype code;
- A hash of a video data file (the video data file itself is stored by the user);
- Date and time of uploading the hash.

To verify the authenticity of user's video file calculates its hash, which is sent to the smart PROOF contract. In response, the smart contract PROOF returns the information stored for that hash or notifies the user that such hash has not been loaded. On the basis of the received information, it can be concluded that the file with this hash was created not earlier than the time of receiving the individual swype code and not later than the time of loading the hash into the smart contract.

6.3. Implementation for Emercoin

The implementation for the Emercoin platform assumes that the swype code is generated based on the hash of the last (or any other) block in the chain, the address of the client of the blockchain network and possibly other additional information (for example, the IMEI of the user's mobile device). The pseudocode for generating a swype code might look like this:

```
bytes32 mixedString = mix(userAddress, IMEI, blockHash);
bytes32 temp = sha3(mixedString);
swype = uint16(uint256(temp) % 65536);
```

The hash of the video file is calculated after the video is recorded. The received hash is stored in the blockchain of Emercoin along with the number of the block on which swype code is based, as well as the IMEI of the mobile device with which the video was captured, and the transaction must be executed from the same address that was used in calculating of swype code.

To verify the authenticity of user's video file, calculates its hash, which retrieves the stored information from the Emercoin blockchain. On the basis of the received information, it can be concluded that the file with this hash was created not earlier than the time of adding to the blockchain of the block based on the hash of which the swype code was generated and not later than the download time of the hash of the video file in the blockchain.

7. Where it can be applied

PROVER technology can be used independently, but it takes on special value because it can be used as a basis for a large number of applications and services from a wide range of areas.

7.1. Auto Insurance

Let's see more detailed at the possible creation of an application solution for car insurance, for example, for express insurance for a certain short period (one day, one trip, etc.). The client carries out the video recording of the condition of his car for the conclusion of an insurance contract with the purpose of providing this video, certified by the PROVER system, as evidence in a case of an insured event.

In a case of an insured event, the user presents the recorded video data to the insurance company to verify their authenticity.

The application solution for the insurance company will include the following components:

- ***The platform server (service provider)*** is a non-visual service that receives requests from the Mobile applications of clients, transmits them to the Server of the insurance company, and places information in the blockchain.

- ***The insurance company's server*** is a service of an insurance company that directly rates the insurance services provided to the user. In fact, the server of the insurance company believes how much money is left for the client. For data verification, the insurance company's server can access the platform server, and can directly access the blockchain.

- ***Mobile application*** is for managing the status of customers, allowing to enable or to disable the insurance, and specify the volume of provided services. The mobile application interacts only with the Platform Server, which guarantees the client to storage of the data in blockchain. The access code to the service can be the insurance policy number, which is bought by the client directly at the Insurance company. When the insurance is activated, the user must record on the video all sides of the vehicle. A user's video file is stored on his (smartphone, tablet or on a personal cloud drive such as Google Drive or Dropbox), a hash is sent to the Platform Server, which is hosted in the blockchain. In a case of an insured event, the user sends a video file to the Insurance Company, and the Insurance company verifies its authenticity by hash and the time of recording through the blockchain.

This approach allows excluding fraud with insurance when a client in collusion with an employee of an insurance company or a law enforcement officer conclude an insurance contract of a pre-damaged car or claims compensation of fake insurance event. On a global scale, it will allow insurance companies to save billions of dollars annually.

The essential point for that application is confirmation unambiguous or with a high degree of reliability, that the provided video data has been recorded by the client not earlier or not later that known points fixed in time (insured event). And the recording was made by the user real camera of a personal mobile device (smartphone, tablet, a personal computer, a laptop). This to prevent the possibility to apply the video data which recorded before the insured event happened.

The innovative product will be a server platform and a mobile application that allows to enable and disable insurance and lasting of the insurance period, enter the amount of requested services (insurance options). This will allow, for example, not to insure the car from theft when the driver is behind the wheel, or even to disable insurance when the car is parked in the garage. In this case, the installation of additional devices in the car is not required.

Competitive advantages of the created product are:

- The opportunity to reduce the price of insurance through the flexible management of insurance during the entire period of the contract, thereby attracting car owners of different levels of prosperity, incl. Those who did not use this insurance product in general because of its high cost;
- No need to install additional devices on the car;
- The opportunity for the owner of the insurance to easily and quickly prove the occurrence of the insured event by sending video materials to the Insurance Company, while confirming the authenticity of the presented video is carried out using PROVER technology;
- The possibility of providing a trusted interaction between the client and the insurance company using a distributed registry based on blockchain that stores all insurance options management transactions where the guarantor of the safety of the actions performed by the client is not the insurance company but millions of users of the blockchain network around the world!

7.2. Confirming the authorship of the original video (Digital proof of ownership)

The user, while recording video (using a smartphone or tablet camera), can also register his authorship and be able to confirm it. This can be interesting for mobile reporters, freelance correspondents (stringers), bloggers, extreme sportsmen, travelers, musicians, composers and other people engaged in creative work, as well as ordinary users of social networks and video services.

Monetization is assumed on the basis of the freemium model. The application will provide the opportunity to register several videos for free, then the service will offer to buy a subscription for a certain period or offer to purchase packages for registration of 50, 100, 500 video files.

7.3. Remote control of patients (medication tracking)

PROVER technology can become the basis of a remote patient control system, in particular, in the field of confirmation usage of prescribed medication. For this, the patient installs the app of the clinic or pharmaceutical company, where PROVER technology is built-in. Patient has a "medications kit" and a schedule of taking medications. At the right time, a notification pops up on the phone, the user turns on the application, starts video recording, takes a pill out of the package and drinks it, authenticating the video with swype code.

For the clinic and insurance company, this can serve as confirmation that the patient has received the proper therapy in full.

For the patient - it proofs that he correctly fulfilled all the prescriptions of doctors, and the treatment will be paid by the insurance company.

For a pharmaceutical company, this can be a guarantee that the entire prescribed course of taking their medicine will be carried out by the patient in full, which will significantly increase the sales of medicines, bringing additional profit.

7.4. Online games and quests

The PROVER technology could be integrated into the mobile games applications, which allows you to register the game events, for example, detection of "treasures" with game prizes, achievement of a certain level, geographical point, etc.

7.5. Reports on the work done

With PROVER technology, it will be easy to carry out a remote inspection of the works (construction, installation, cleaning, patrolling, courier delivery, placing the goods in the store, placing outdoor advertising, etc.). Video impartially captures the process and fact the work done, and the PROVER service objectively assures the date and time of video recording.

7.6. Recording the evidence of traffic violations and public order

PROVER technology can be used in the field of public order protection. With its help, the user could impartially record the video evidence of crime or offense with local time, date and coordinates of the crime scene. And to transmit it to law enforcement agencies with objective evidence of authenticity. The data can be used as evidence in a court, technical expertise can be claimed, and the expert will be able to retrieve information about synchronization and geo-location and confirm its authenticity.

7.7. Educational projects

For educational projects, PROVER technology can perform the function of remote identification of users and verification of their actions within the educational platform. To date, in the world has gained popularity of the platforms for Mass Open Online Courses (MOOCs), for example, Coursera, Udemy, Udacity, edX and others. Vulnerable part of these services is the impossibility of 100% confirmation of the learner's personality and verification of the received knowledge. At the moment, on the Coursera platform, it is required to do selfies, but this approach is absolutely not protected from forgery. It is possible to integrate PROVER technology into the distance learning applications and create video recordings of the exams by the user, to make sure that the exam is really passed by the person on the video, and that he did not use cheat sheets, did not left the room, etc. This approach will allow to issue personal certificates with a photo and be sure that the person pictured in the photo on the certificate is the person, who passed the exams. The video of exams can also be stored for further confirmation.

7.8. Crypto protected video statements - confirmed video messages of a legal nature

PROVER technology makes possible to protect and verify video messages of a legal nature without a necessity to visit any officials. Remote video statement, used for a transaction, giving evidence, explanation, report, interview, etc., will become objective and legitimate evidence. The system confirms the date and the place on the video, the notify body compares the video with the photo or video could be available in the database and confirms that it is the same person. For example, remote reception of citizens. Municipal and corporate officials can work in this way, considering the recorded appeals and statements of citizens /clients.

8. Team

The project team has experience for more than ten years of teamwork. We implemented a number of large-scale projects in the field of intelligent video surveillance systems, hardware products, and IT services for healthcare in 64 countries. At the moment, the project team follow to the principles of openness and decentralization and invests in the development of advanced technologies associated with the blockchain. We know how to make high-quality and popular products, and we believe that we can benefit from the whole blockchain community by making the PROVER project our contribution to crypto-economics.

8.1. Core

Nadezhda Nabilskaya, CEO, co-founder

- In the field of information security and technical security means has been working since 2010.
- She is a graduate of the Russian Presidential Program for Management Training.
- Project management: software development - 3, research and development work - 5, applied research - 2.

Alexey Rytikov, CTO

- Experience in developing software in the field of technical security equipment for 10 years.
- Participated in couple of RnD projects as a key participant. Has experience in developing complex technical systems in the field of video surveillance.

Vyacheslav Voronin, scientist in the field of video analysis

- Associate Professor of the Don State Technical University (Russia).
- PhD in Technical Sciences.
- Co-author of the monograph "Method and algorithms for the separation of a useful signal against a background of noise during the processing of discrete signals".
- Reviewer of the International Journal of IEEE Transactions on Image Processing, International Conference on International Symposium on Image and Signal Processing and Analysis (ISPA), International Conference on International Symposium on Circuits and Systems (ISCAS).
- The winner of the personal award of the head of the administration (Governor) of the Rostov region (Russia).
- Laureate of the award "Polzunovskie grants".

Vitaly Suprun, mobile application developer

- Experience in developing mobile software for more than 10 years.
- The developer of mobile application ECG Dongle.

Elena Yuferova, business consultant

- Experience in consulting companies as the head of the direction of HR consulting and real sector companies as an expert in the management of organization and human resources for more than 25 years.

- Co-author of the handbook on personnel management. Co-author of the practical manual for managers "Face to face with the future employee" M, 2001.

8.2. Advisory board

Evgeny Shumilov

- The evangelist is a blockchain of technology.
- Founder and CEO of Emercoin.

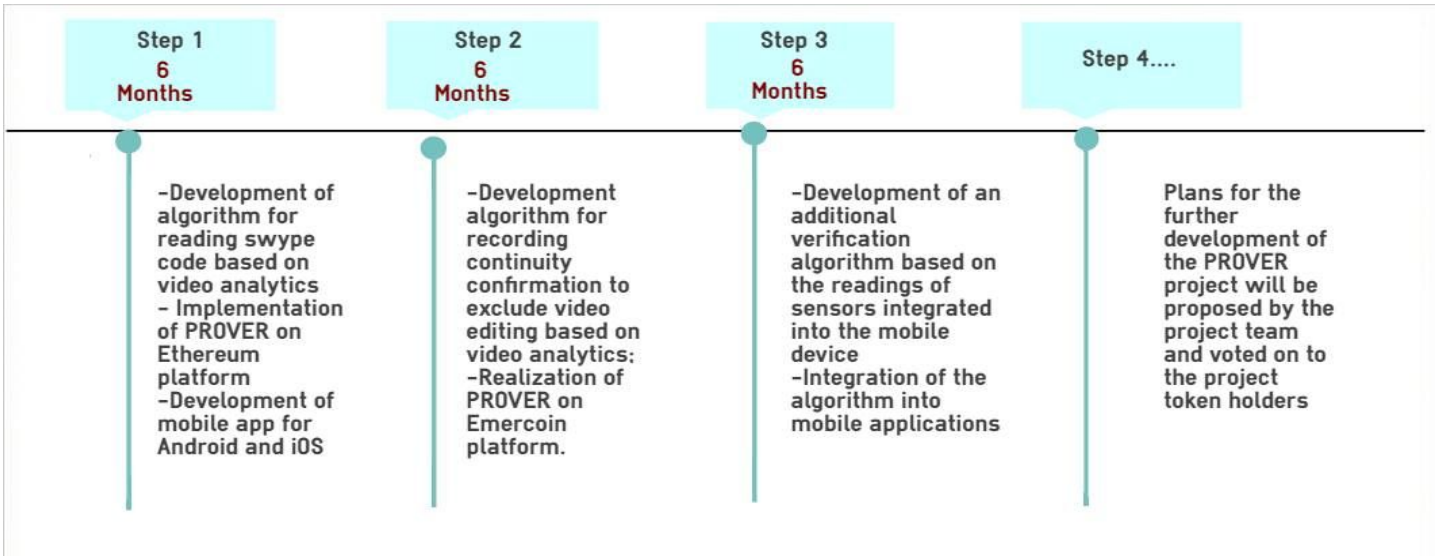
Oleg Khovayko

- Expert in cryptography and finance.
- Technical director of Emercoin.

Ilya Svirin, co-founder

- PhD in Technical Sciences.
- Technological entrepreneur.
- The founder of the group of companies "Nordavind".
- The developer of technologies in the field of digital video surveillance systems, personal equipment and services for health (including the world-famous ECG Dongle and the CardioCloud service).
- He is the author of numerous scientific publications on information security issues and theoretical principles of programming.

9. Project roadmap



10. Guide to investment

To fund the development and ensure the functioning of the PROVER system, a fund-raising phase, known as crowdsale, will be conducted. The crowdsale will be conducted in the ecosystem of the Etherium. During the crowdsale, people could purchase tokens PROOF by a fixed rate, which:

- Are means of payment when paying for the PROVER service. At the technological layer application services (for example, auto insurance services) pay to PROVER by PROOF tokens, and mutual settlements with their users are conducted in any currency, for example, in their tokens or fiat money;
- Gives the right to place a request for the project to finance. Projects can be aimed at developing the PROVER infrastructure directly (developing new algorithms, creating implementations for other platforms), marketing and promoting the PROVER project, creating application solutions aimed at developing the PROVER ecosystem (implementing application services using PROVER as a means of authenticating video content) . The volume of the request can not exceed the number of PROOF tokens available to the applicant, multiplied by 1000, which allows to take into account the degree of involvement of the applicant in the PROVER community, and therefore his personal interest in the development of the project;
- Gives the right to vote on projects, submitted for financing. The strength of the participant's voice is proportional to the number of PROOF tokens belongs to him.

To ensure the closure of the PROVER economic system, the PROOF tokens, collected by the PROOF smart contract, must be returned to circulation:

- 10% of the collected PROOF tokens, but not less than 1 token per each confirmed video, are sent to the project team to support the functioning of the decentralized system;
- 90% of the collected PROOF tokens remain under the control of the PROOF smart contract and are distributed following the voting results of the PROOF token holders for projects initiated by both the project team and external contractors.

10.1. Pre-ICO

The PROOF Smart Contract issues the emission of tokens PROOF, the number of which during the Pre-ICO the maximum collected amount is limited to 300'000$, upon reaching which the release of tokens ceases. Tokens are sold at a price fixed in US dollars, 125 PROOF = 1$, i.e. the investor in the Pre-ICO stage has a 25% bonus, compared to the subsequent crowdsale. The purchase is carried out by transferring the ether to the address of the smart contract, and the sender of the transaction automatically becomes the owner of the purchased tokens. **Be careful and remember that you should not pay from incompatible with ERC20 contracts wallets or from an account on a crypto exchange - this can lead to loss of control over the tokens you have purchased.** The rate of the ether to the US dollar is fixed at the time of the launch of the Pre-ICO and remains unchanged during the whole period of its holding. The duration of the Pre-ICO is 14 days from launch. During Pre-ICO, all issued tokens must be sold, otherwise all collected funds are returned to investors, minus commissions for transactions and gas prices.

All funds collected for Pre-ICO are transferred to the PROVER team and should be spent on the following works:

- Development of a prototype of a mobile application for Android, which provides interaction with the PROOF Smart Contract;
- Marketing and project promotion, preparation for crowdsale;
- Translation white paper into Chinese, German, French, Italian, Spanish and Ukrainian.

10.2. Crowdsale

The PROOF Smart Contract carries out the emission of PROOF tokens, the amount of which during crowdsale is limited to a maximum collected amount of 5'200'000 $, at which point the release of tokens ceases. Tokens are sold at a price fixed in US dollars, 100 PROOF = 1$. The purchase is carried out by transferring the ether to the address of the smart contract, and the sender of the transaction becomes the owner of the purchased tokens. **Be careful and remember that you should not pay from an online wallet or from an account on a crypto exchange, which can lead to loss of control over the tokens you have purchased.** The rate of ether to the US dollar is fixed at the time the crowdsale is launched and remains unchanged during the entire time of its holding. After the end of crowdsale, a one-time additional emission is carried out, during which 28% of the total issue of tokens are issued, 19% of which remain with the project team, 1% are for project advisors, and 8% are transferred to the IcoLab Foundation. The duration of the crowdsale is 30 days from launch.

For early birds there is a system of discounts:

- 115 PROOF = 1$ during of the first day of crowdsale;
- 110 PROOF = 1$ during of the first week of crowdsale.

The funds received during crowdsale uses to create the PROVER infrastructure and to ensure its functioning. After that, there is no emissions of tokens. The condition for the success of crowdsale is to collect a minimum of 3'600'000 $, not counting the funds collected during the Pre-ICO. Otherwise all collected funds will be returned to investors, minus commissions for transactions and gas costs.

The number of stages of the project depends on the amount of funds collected during the crowdsale and is determined from the calculation per stage. The result of each stage is creation a functionally complete product, the execution of each next stage adds its additional consumer properties.

10.3. Spendings of investment capital

Immediately after the successful crowdsale, the funds, in the amount of 1’500’000 $, are transferred by PROOF Smart Contract to the team, to perform the first stage of the project. The rest of the funds remain under the control of the PROOF Smart Contract.

At the end of each phase of the project, the team posts the results of the work in the public domain on the project website http://prover.io and at https://github.com/isvirin/prover, and also publishes the work plan for the next stage and the required funding. All this to be put to the vote, which lasts 7 days from the moment of its initiation by the development team.

10.4. Synergy of PROOF and HMQ tokens

Humaniq and PROVER projects expand the ecosystems of each other's. So for PROVER, the Humaniq project is an application solution. The mutual settlements between Humaniq and PROVER are carried out in PROOF tokens, while Humaniq works with its clients in its own HMQ tokens.

Thus, the increase in the cost of PROOF tokens is provided due to the fact that the Humaniq project extends the PROVER ecosystem, creating an additional demand for PROOF tokens on the market with one time fixed emission.

The increase in the cost of HMQ tokens is ensured by the fact that the PROVER project extends the functionality of the Humaniq system, providing additional value to Humaniq for customers, which is additionally provided in the framework of the permanent emission of HMQ tokens.

11. User guide. PROOF Smart Contract API

The PROOF Smart Contract is developed using the Ethereum ecosystem and carries out the release of PROOF tokens using the Token ERC20 standard. The ERC20 specification has been expanded to ensure the possibility of voting and implementing the general business logic of the PROVER project.

11.1. Voting for the results of the work of the project team

For PROOF tokens owners, there is a possibility to vote for the acceptance of completed work by the project team, as well as approval of the plan for further work and allocation of funding for the next stage. Voting is started by the project team by calling the ***startVoting(uint weiReqFund)*** function of the PROOF Smart Contract and lasting 7 days. As a parameter, the function receives the amount of required funding for the next stage of the project. At the beginning of voting, the event ***VotingStarted(uint weiReqFund)*** is issued, which contains information about the funding requested for the next stage.

To obtain information about the current voting, the ***votingInfo()*** function can be called, which returns information about the funding requested for the next stage and the time before the voting ends. If the vote is not completed or completed, the function returns zero values.

The ***vote(bool inSupport)*** voting function is available only to PROOF tokens holders and can be called by each holder only once during a single vote. Counting of votes is carried out only at the moment of completion of voting, in order to exclude multiple accounting of the same tokens during voting, if during the voting their transfer was made. With each successful function call, the event ***Voted(address indexed voter, bool in Support)*** is emitted, which contains the address of the voted PROOF tokens holder and its voice.

Voting is terminated by the project team by calling the ***finishVoting()*** function after 7 days of voting. The function calculates votes, taking into account the actual number of tokens owned by all holders who called the vote (bool inSupport) function during the voting. The voice weight of each PROOF tokens holder is determined by the number of PROOF tokens available in it in total emissions. Voting is carried out by a simple majority. When the voting is completed, the event ***VotingFinished(bool inSupport)*** is emitted, which contains information about the outcome of the vote. With a positive outcome of the vote, the requested funds in ether are transferred from the account of PROOF Smart Contract to the address of the project team.

11.3. Placement of evidence

Placement of evidence in the PROOF Smart Contract is implemented in two stages. At the first stage, the user calls the function ***swypeCode()***, which randomly generates the swype code and returns it to the user. In addition, the function stores the generated swype code, as well as the time of its issuance.

After the user has recorded the video (taking into account the issued swype code), he calls the ***setHash(uint16 swype, bytes32 hash)*** function, which takes as parameters the previously issued swype code and hash of the recorded video file. The function stores the transmitted hash in an associative array, where for each element the swype code, the swype code output time, the hash loading time and the owner of the video material are stored. It is the call of this function that requires payment for the implementation of the service in accordance with the tariff, depending on the number of PROOF tokens the user has.

11.4. Migrating of tokens

In the case of revealing critical errors in the business logic of PROOF Smart Contract or developing new technical solutions that significantly improve business logic, the PROVER project team reserves the right to replace the smart contract with a new one. At the same time, in order to meet the interests of the holders of the tokens, a mechanism is provided for secure migration of tokens.

11.5. Funding of projects initiated by the community

Any PROOF token holder, who is interested in the development or promotion of the PROVER project has the rights to place a request for funding. To place the request, the PROOF token holder calls the function ***deployProject (uint proofReqFund, string urlInfo)***, which receives as parameters the requested amount of funding in PROOF tokens, as well as the URL where the public description of the project is posted. It should be taken into account that the volume of the request can not exceed the number of PROOF tokens available to the applicant, multiplied by 1000. If the request for funding has already been placed by the PROOF token holder and it is on the ballot, placing one more request is not allowed. A new request can be placed immediately after the voting is completed on the already posted request, regardless of the voting results.

If the request is placed successfully, the event ***Deployed (address projectOwner, uint proofReqFund, string urlInfo)*** is emitted and the voting procedure begins, which lasts 7 days.

To receive information about the placed query for voting, the constant function ***projectInfo(address projectOwner)*** can be called up, which returns information about the requested funding, the URL for the public description of the project and the time before voting ends. If the vote is not completed or completed, the function returns zero values.

The voting function ***vote(address projectOwner, bool inSupport)*** is available only to PROOF token holders and can be called by each holder only once during the voting for each request. Counting of votes is carried out only at the moment of completion of voting, in order

to exclude multiple accounting of the same tokens during voting, if during the voting their transfer was made. Each time the function is successfully called, is emitted event ***Voted(address projectOwner, address voter, bool inSupport)***.

Voting is completed by calling ***finishVoting(address projectOwner)*** after 7 days of voting. The function calculates votes, taking into account the actual number of tokens owned by all holders who called the **vote(address projectOwner, bool in support)** during the voting. The voice weight of each PROOF token holder is determined by the number of PROOF tokens he has. Voting is carried out by a simple majority. When voting is completed, the event ***VotingFinished(address projectOwner, bool inSupport)*** is emitted, which contains information about the outcome of the vote. With a positive outcome of voting, the requested funds in PROOF tokens are transferred from the balance of the PROOF Smart Contract to the balance of the applicant.

12. Conclusion

The first phones with video cameras appeared around 2002, and the era of smartphones officially started with the advent of the iPhone came only 10 years ago. Until 2005, there was no YouTube service in the world! The first version of code Bitcoin originated in 2008, and the first wallet of Satoshi Nakamoto was created in early 2009. Can you imagine your life today without them?

The number of smartphones users in the world is about 4 billion people, and by 2020 there will be more than 6 billion. And all these people will instantly communicate in messengers, post content, including video content in various of social networks.

All of them will be involved in new economy, they will acquire goods and services online, using their smart devices, using crypto currency. Most of the today's offline services will be transmitted online and scaled to billions of users around the world. This process is seriously hampered now only by the fear of fraud and security requirements.

Our technology can put a reliable barrier in the way of scammers and open widely the doors to online and crypto-economy for whole branches of banking and insurance, legal and other conservative spheres.

It should also note the global benefits of our project for the development of the blockchain community. Our system could be a driver for the popularization of blockchain technology and crypto-currency among the population of our planet. It will bring to blockchain economy millions of new users from all over the world, increasing the overall volume and popularity of the blockchain economy!

13. Links

1. https://en.wikipedia.org/wiki/Swype
2. https://en.wikipedia.org/wiki/Insurance_fraud
3. By the numbers: insurance fraud statistics
4. Google content ID
5. Ethereum homepage

6. Ethereum Request for Comments (ERC) 20
7. Solidity homepage
8. How To Learn Solidity: The Ultimate Ethereum Coding Guide
9. BlockChain Technology Beyond Bitcoin

14. PROOF smart contract

```
/*
This file is part of the PROOF Contract.

The PROOF Contract is free software: you can redistribute it and/or
modify it under the terms of the GNU lesser General Public License as published
by the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

The PROOF Contract is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU lesser General Public License for more details.

You should have received a copy of the GNU lesser General Public License
along with the PROOF Contract. If not, see <http://www.gnu.org/licenses/>.
*/

pragma solidity ^0.4.0;

contract owned {

    address public owner;

    function owned() payable {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(owner == msg.sender);
        _;
    }

    function changeOwner(address _owner) onlyOwner public {
        require(_owner != 0);
        owner = _owner;
    }
}

contract Crowdsale is owned {

    uint256 public totalSupply;
```

```
mapping (address => uint256) public balanceOf;

uint    public etherPrice;
address public crowdsaleOwner;
uint    public totalLimitUSD;
uint    public minimalSuccessUSD;
uint    public collectedUSD;

enum State { Disabled, PreICO, CompletePreICO, Crowdsale, Enabled, Migration }
event NewState(State state);
State   public state = State.Disabled;
uint    public crowdsaleStartTime;
uint    public crowdsaleFinishTime;

modifier enabledState {
    require(state == State.Enabled);
    _;
}

struct Investor {
    uint256 amountTokens;
    uint    amountETH;
}
mapping (address => Investor) public investors;
mapping (uint => address)     public investorsIter;
uint                          public numberOfInvestors;

function () payable {
    require(state == State.PreICO || state == State.Crowdsale);
    uint256 tokensPerUSD = 0;
    if (state == State.PreICO) {
        tokensPerUSD = 125;
    } else if (state == State.Crowdsale) {
        if (now < crowdsaleStartTime + 1 days) {
            tokensPerUSD = 115;
        } else if (now < crowdsaleStartTime + 1 weeks) {
            tokensPerUSD = 110;
        } else {
            tokensPerUSD = 100;
        }
    }
    if (tokensPerUSD > 0) {
        uint valueETH = msg.value;
        uint valueUSD = valueETH * etherPrice / 1000000000000000000;
        if (collectedUSD + valueUSD > totalLimitUSD) { // don't need so much ether
            valueUSD = totalLimitUSD - collectedUSD;
            valueETH = valueUSD * 1000000000000000000 / etherPrice;
```

```
            msg.sender.transfer(msg.value - valueETH);
            collectedUSD = totalLimitUSD; // to be sure!
        }
        uint256 tokens = tokensPerUSD * valueUSD;
        require(balanceOf[msg.sender] + tokens > balanceOf[msg.sender]); // overflow
        require(tokens > 0);

        Investor memory inv = investors[msg.sender];
        if (inv.amountETH == 0) { // new investor
            investorsIter[numberOfInvestors++] = msg.sender;
        }
        inv.amountTokens += tokens;
        inv.amountETH += valueETH;
        investors[msg.sender] = inv;
        totalSupply += tokens;
        collectedUSD += valueUSD;
    }
}

function startTokensSale(address _crowdsaleOwner, uint _etherPrice) public onlyOwner {
    require(state == State.Disabled || state == State.CompletePreICO);
    crowdsaleStartTime = now;
    crowdsaleOwner = _crowdsaleOwner;
    etherPrice = _etherPrice;
    delete numberOfInvestors;
    delete collectedUSD;
    if (state == State.Disabled) {
        crowdsaleFinishTime = now + 14 days;
        state = State.PreICO;
        totalLimitUSD = 300000;
        minimalSuccessUSD = 300000;
    } else {
        crowdsaleFinishTime = now + 30 days;
        state = State.Crowdsale;
        totalLimitUSD = 5200000;
        minimalSuccessUSD = 3600000;
    }
    NewState(state);
}

function timeToFinishTokensSale() public constant returns(uint t) {
    require(state == State.PreICO || state == State.Crowdsale);
    if (now > crowdsaleFinishTime) {
        t = 0;
    } else {
        t = crowdsaleFinishTime - now;
    }
```

```
}

function finishTokensSale(uint _investorsToProcess) public onlyOwner {
    require(state == State.PreICO || state == State.Crowdsale);
    require(now >= crowdsaleFinishTime || collectedUSD >= minimalSuccessUSD);
    if (collectedUSD < minimalSuccessUSD) {
        // Investors can get their ether calling withdrawBack() function
        while (_investorsToProcess > 0 && numberOfInvestors > 0) {
            address addr = investorsIter[--numberOfInvestors];
            Investor memory inv = investors[addr];
            balanceOf[addr] -= inv.amountTokens;
            totalSupply -= inv.amountTokens;
            --_investorsToProcess;
            delete investorsIter[numberOfInvestors];
        }
        if (numberOfInvestors > 0) {
            return;
        }
        if (state == State.PreICO) {
            state = State.Disabled;
        } else {
            state = State.CompletePreICO;
        }
    } else {
        while (_investorsToProcess > 0 && numberOfInvestors > 0) {
            --numberOfInvestors;
            --_investorsToProcess;
            delete investors[investorsIter[numberOfInvestors]];
            delete investorsIter[numberOfInvestors];
        }
        if (numberOfInvestors > 0) {
            return;
        }
        if (state == State.PreICO) {
            if (!crowdsaleOwner.send(this.balance)) throw;
            state = State.CompletePreICO;
        } else {
            if (!crowdsaleOwner.send(1500000 * 1000000000000000000 / etherPrice)) throw;
            // Create additional tokens for owner (28% of complete totalSupply)
            balanceOf[msg.sender] = totalSupply * 28 / 72;
            totalSupply += totalSupply * 28 / 72;
            state = State.Enabled;
        }
    }
    NewState(state);
}
```

```
    // This function must be called by token holder in case of crowdsale failed
    function withdrawBack() public {
        require(state == State.Disabled || state == State.CompletePreICO);
        uint value = investors[msg.sender].amountETH;
        if (value > 0) {
            delete investors[msg.sender];
            msg.sender.transfer(value);
        }
    }
}

contract Token is Crowdsale {

    string  public standard    = 'Token 0.1';
    string  public name        = 'PROOF';
    string  public symbol      = "PF";
    uint8   public decimals    = 0;

    modifier onlyTokenHolders {
        require(balanceOf[msg.sender] != 0);
        _;
    }

    mapping (address => mapping (address => uint256)) public allowed;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    function Token() payable Crowdsale() {}

    function transfer(address _to, uint256 _value) public enabledState {
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]); // overflow
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
        Transfer(msg.sender, _to, _value);
    }

    function transferFrom(address _from, address _to, uint256 _value) public {
        require(balanceOf[_from] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]); // overflow
        require(allowed[_from][msg.sender] >= _value);
        balanceOf[_from] -= _value;
        balanceOf[_to] += _value;
        allowed[_from][msg.sender] -= _value;
        Transfer(_from, _to, _value);
    }
```

```
    function approve(address _spender, uint256 _value) public enabledState {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }

    function allowance(address _owner, address _spender) public constant enabledState
        returns (uint256 remaining) {
        return allowed[_owner][_spender];
    }
}

contract MigrationAgent {
    function migrateFrom(address _from, uint256 _value);
}

contract TokenMigration is Token {

    address public migrationAgent;
    uint256 public totalMigrated;

    event Migrate(address indexed from, address indexed to, uint256 value);

    function TokenMigration() payable Token() {}

    // Migrate _value of tokens to the new token contract
    function migrate(uint256 _value) external {
        require(state == State.Migration);
        require(migrationAgent != 0);
        require(_value != 0);
        require(_value <= balanceOf[msg.sender]);
        balanceOf[msg.sender] -= _value;
        totalSupply -= _value;
        totalMigrated += _value;
        MigrationAgent(migrationAgent).migrateFrom(msg.sender, _value);
        Migrate(msg.sender, migrationAgent, _value);
    }

    function setMigrationAgent(address _agent) external onlyOwner {
        require(migrationAgent == 0);
        migrationAgent = _agent;
        state = State.Migration;
    }
}

contract ProofTeamVote is TokenMigration {
```

```
function ProofTeamVote() payable TokenMigration() {}

event VotingStarted(uint weiReqFund);
event Voted(address indexed voter, bool inSupport);
event VotingFinished(bool inSupport);

struct Vote {
    bool inSupport;
    bool voted;
}

uint public weiReqFund;
uint public votingDeadline;
uint public numberOfVotes;
uint public yea;
uint public nay;
mapping (address => Vote) public votes;
mapping (uint => address) public votesIter;

function startVoting(uint _weiReqFund) public enabledState onlyOwner {
    require(weiReqFund == 0 && _weiReqFund > 0 && _weiReqFund <= this.balance);
    weiReqFund = _weiReqFund;
    votingDeadline = now + 7 days;
    VotingStarted(_weiReqFund);
}

function votingInfo() public constant enabledState
    returns(uint _weiReqFund, uint _timeToFinish) {
    _weiReqFund = weiReqFund;
    if (votingDeadline <= now) {
        _timeToFinish = 0;
    } else {
        _timeToFinish = votingDeadline - now;
    }
}

function vote(bool _inSupport) public onlyTokenHolders enabledState
    returns (uint voteId) {
    require(votes[msg.sender].voted != true);
    require(votingDeadline > now);
    voteId = numberOfVotes++;
    votesIter[voteId] = msg.sender;
    votes[msg.sender] = Vote({inSupport: _inSupport, voted: true});
    Voted(msg.sender, _inSupport);
    return voteId;
}
```

```
    function finishVoting(uint _votesToProcess) public enabledState onlyOwner
        returns (bool _inSupport) {
        require(now >= votingDeadline && weiReqFund <= this.balance);

        while (_votesToProcess > 0 && numberOfVotes > 0) {
            address voter = votesIter[--numberOfVotes];
            Vote memory v = votes[voter];
            uint voteWeight = balanceOf[voter];
            if (v.inSupport) {
                yea += voteWeight;
            } else {
                nay += voteWeight;
            }
            delete votes[voter];
            delete votesIter[numberOfVotes];
            --_votesToProcess;
        }
        if (numberOfVotes > 0) {
            _inSupport = false;
            return;
        }

        _inSupport = (yea > nay);

        if (_inSupport) {
            if (migrationAgent == 0) {
                if (!owner.send(weiReqFund)) throw;
            } else {
                if (!migrationAgent.send(weiReqFund)) throw;
            }
        }

        VotingFinished(_inSupport);
        delete weiReqFund;
        delete votingDeadline;
        delete numberOfVotes;
        delete yea;
        delete nay;
    }
}

contract ProofPublicVote is ProofTeamVote {

    function ProofPublicVote() payable ProofTeamVote() {}

    event Deployed(address indexed projectOwner, uint proofReqFund, string urlInfo);
    event Voted(address indexed projectOwner, address indexed voter, bool inSupport);
```

```
    event VotingFinished(address indexed projectOwner, bool inSupport);

    struct Project {
        uint   proofReqFund;
        string urlInfo;
        uint   votingDeadline;
        uint   numberOfVotes;
        uint   yea;
        uint   nay;
        mapping (address => Vote) votes;
        mapping (uint => address) votesIter;
    }
    mapping (address => Project) public projects;

    function deployProject(uint _proofReqFund, string _urlInfo) public onlyTokenHolders
enabledState {
        require(_proofReqFund > 0 && _proofReqFund <= balanceOf[this]);
        require(_proofReqFund <= balanceOf[msg.sender] * 1000);
        require(projects[msg.sender].proofReqFund == 0);
        projects[msg.sender].proofReqFund = _proofReqFund;
        projects[msg.sender].urlInfo = _urlInfo;
        projects[msg.sender].votingDeadline = now + 7 days;
        Deployed(msg.sender, _proofReqFund, _urlInfo);
    }

    function projectInfo(address _projectOwner) enabledState public
        returns(uint _proofReqFund, string _urlInfo, uint _timeToFinish) {
        _proofReqFund = projects[_projectOwner].proofReqFund;
        _urlInfo = projects[_projectOwner].urlInfo;
        if (projects[_projectOwner].votingDeadline <= now) {
            _timeToFinish = 0;
        } else {
            _timeToFinish = projects[_projectOwner].votingDeadline - now;
        }
    }

    function vote(address _projectOwner, bool _inSupport) public onlyTokenHolders
enabledState
        returns (uint voteId) {
        Project storage p = projects[_projectOwner];
        require(p.votes[msg.sender].voted != true);
        require(p.votingDeadline > now);
        voteId = p.numberOfVotes++;
        p.votesIter[voteId] = msg.sender;
        p.votes[msg.sender] = Vote({inSupport: _inSupport, voted: true});
        projects[_projectOwner] = p;
        Voted(_projectOwner, msg.sender, _inSupport);
```

```
        return voteId;
    }

    function finishVoting(address _projectOwner, uint _votesToProcess) public enabledState
        returns (bool _inSupport) {
        Project storage p = projects[_projectOwner];
        require(now >= p.votingDeadline && p.proofReqFund <= balanceOf[this]);

        while (_votesToProcess > 0 && p.numberOfVotes > 0) {
            address voter = p.votesIter[--p.numberOfVotes];
            Vote memory v = p.votes[voter];
            uint voteWeight = balanceOf[voter];
            if (v.inSupport) {
                p.yea += voteWeight;
            } else {
                p.nay += voteWeight;
            }
            delete p.votesIter[p.numberOfVotes];
            delete p.votes[voter];
            --_votesToProcess;
        }
        if (p.numberOfVotes > 0) {
            projects[_projectOwner] = p;
            _inSupport = false;
            return;
        }

        _inSupport = (p.yea > p.nay);

        if (_inSupport) {
            transfer(_projectOwner, p.proofReqFund);
        }

        VotingFinished(_projectOwner, _inSupport);
        delete projects[_projectOwner];
    }
}

contract Proof is ProofPublicVote {

    struct Swype {
        uint16  swype;
        uint    timestampSwype;
    }

    struct Video {
        uint16  swype;
```

```
        uint    timestampSwype;
        uint    timestampHash;
        address owner;
    }

    mapping (address => Swype) public swypes;
    mapping (bytes32 => Video) public videos;

    uint priceInTokens;
    uint teamFee;

    function Proof() payable ProofPublicVote() {}

    function setPrice(uint _priceInTokens) public onlyOwner {
        require(_priceInTokens >= 2);
        teamFee = _priceInTokens / 10;
        if (teamFee == 0) {
            teamFee = 1;
        }
        priceInTokens = _priceInTokens - teamFee;
    }

    function swypeCode() public enabledState returns (uint16 _swype) {
        bytes32 blockHash = block.blockhash(block.number - 1);
        bytes32 shaTemp = sha3(msg.sender, blockHash);
        _swype = uint16(uint256(shaTemp) % 65536);
        swypes[msg.sender] = Swype({swype: _swype, timestampSwype: now});
    }

    function setHash(uint16 _swype, bytes32 _hash) public enabledState {
        require(balanceOf[msg.sender] >= priceInTokens);
        require(swypes[msg.sender].timestampSwype != 0);
        require(swypes[msg.sender].swype == _swype);
        transfer(owner, teamFee);
        transfer(this, priceInTokens);
        videos[_hash] = Video({swype: _swype,
timestampSwype:swypes[msg.sender].timestampSwype,
            timestampHash: now, owner: msg.sender});
        delete swypes[msg.sender];
    }
}
```