ZESZYTY NAUKOWE WYŻSZEJ SZKOŁY TECHNICZNEJ W KATOWICACH
ISSN 2082-7016; eISSN 2450-5552 2022, nr 14

DOI: 10.54264/0035

MACIEJ MADEJCZYK
student, Katowice School of Technology, e-mail: *maciej.madejczyk@onet.pl*

TOMASZ TREJDEROWSKI PHD
Katowice School of Technology, e-mail: *tomasz.trejderowski@wst.pl*

BEATA TREJDEROWSKA
Ms, Katowice School of Technology, e-mail: *beata.trejderowska@wst.pl*

PROTECTION OF PERSONAL DATA USING BLOCKCHAIN

s. 89-100

ABSTRACT

The presented document is a scientific and technical proposal of the possibility of using blockchain in the process of protection of personal data on the Internet and inside IT systems. The key aspects of this proposal along with most pros and cons are discussed within this article. A brief discussion of prototype of proposed solution is also included.

KEYWORDS

PII, personal data, blockchain.

1. Definitions
1.1. Personal data

Personal data, also known as personal information or personally identifiable information (PII), is any information related to an identifiable person [1].

National Institute of Standards and Technology defines personally identifiable information as “any information about an individual maintained by an agency, including (1) any information that can be used to distinguish or trace an individual’s identity, such as name, social security number, date and place of birth, mother’s maiden name, or biometric records; and (2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information” [2].

European Union widens this definition in its GDPR directive into “any information which [is] related to an identified or identifiable natural person” [3].

1.2. Blockchain

A type of decentralized network database, a growing list of records, called blocks, which are securely linked to-gether using cryptography [4]. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. Each block contains information about the block before it, and so they form a chain, with each additional block reinforcing the ones before it. This feature causes blockchains to be resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks [5].

The generation of the mentioned cryptographic hash requires intense mathematical calculations and thus uses a significant amount of computing power. Any person or company sharing their computer resources for blockchain calcu-lations is eligible for receiving a small amount of fee (called gas fee). On the other hand, these performed calculations are necessary for proving that the corresponding blockchain transaction was correct (or not). And therefore, whether such transaction can be added to the next block (or on contrary). Any proposed solutions that are based on blockchain must either include this fee in its business model or must be based on a privately setup blockchain to perform blockchain-based calculations and register and store blockchain's transactions [6].

1.3. Smart contracts

A smart contract is a computer program or a transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement. The objective of using smart contracts is to reduce the need of trusted intermediators, transaction costs and fraud losses, as well as the reduction of malicious and accidental exceptions [7]. Due to its nature of work, blockchain is a smart contract, because it can automatically control and document legally happening transactions.

1.4. Decentralized applications (dApps)

One of the software architectures for building computer software or web applications that is based more on direct communication between parties rather than on client-server communication. Blockchains and participating applications (blockchain wallets) are considered decentralized applications because the same portion of information (chain of blocks of transactions) is shared among all the nodes of a particular blockchain. And therefore, any participating client can communicate with any node of this network.

1.5. Blockchain wallet

Blockchain wallet and a corresponding blockchain account is a computer software that allows easy interaction with various blockchain.

There are two general types of blockchain wallets: custodial and non-custodial ones. In the first type of wallet, the public and private key (a foundation of cryptographic hashes behind the blockchain; later described in this article) are kept on the server side of the wallet creator. Since the creator has access to the private key, he can manage the user's (wallet owner) funds. Non-custodial wallets work differently. Private and public keys are stored only on the client's side only. User (wallet owner) must himself make sure to remember (copy) the private key for the any event in the result of which this key could be lost (for example: deleting of the cache in the browser or switching browser or device to another). This private key, stored entirely locally on user's device, is used to interact with the blockchain. All the transactions are signed and checked with it. In the below proposed solution it acts as the decryption and encryption key [7].

2. Introduction
2.1. The definition of the problem

We live in a completely digitalized and networked world, where every action performed by us is stored on a server or inside any application. Starting from our professional lives (the work we do to support our families and ourselves), going through private aspects, leisure, and fun, and to our political objectives, religion believes and fantasies, everything has a digital mark stored somewhere, recorded either by our intentional decisions (i.e., using or not of solutions) or hap-pens completely beyond our will (city surveillance, processing of our payments and shopping etc.).

The information that we share through the Internet and with usage of various applications is a part of personal data, also known personally identifiable information (PII). Various legal solutions have been executed (i.e., GDPR) to protect our digital privacy in these areas [8].

Depending on the hardware and software architecture and business model used by a given application, PII can be transferred and stored in a various solution, with storing on a server and exchanging it with an acting client being one of the most popular. Our PII data is kept on a distant server in this case, encrypted with a strong and secure cryptographic algorithm in most cases.

This approach narrows down to the one of the biggest problems when it comes into securing our PII data. All that is needed to gain access to it, is for the attacker or intruder to poses of our password. This can be achieved including social engineering methods. Then, using the same service (i.e., healthcare network, bank, or social network), they have unlimited access to our most precious, intimate, and confidential data.

2.2. Solution proposal

What we want to achieve as a counter-solution is the deviation of above architecture in four fundamental areas:

1. store data on a single server, but inside a blockchain,
2. store data not as a datagram (i.e., separate data chunks, like name, birthdate, bank account, credit card num-ber), but in form of a one big, encrypted data blob and
3. store private key, which allows decryption of this data, entirely on client side (meaning in a local computer, mobile device or inside web browser's local storage),
4. limit number of connections and transactions between client and server to as minimal as possible.

In other words, PII data is encrypted both in storage and in transit, as a meaningless blob of bytes, and can be de-crypted into logical and readable form on a user computer or device only. The private key needed to decrypt this data is stored within browser's local cache memory and only there can be turned into readable data. In case of user clearing the cache and forgetting its private key, the entire PII dataset is inevitably lost, because it is never and anywhere stored in a readable form. It is always kept (both in storage and in transit) as an encrypted blob of data.

It is also not possible to gain access to that PII dataset by any non-authorized personae (i.e., attacker or intruder), because it is stored in a decentralized form, inside a blockchain. It is also not possible to alter this data in any unauthor-ized way, because each part of this data is stored within a block and each change of this data is a blockchain's transac-tion. Which is resistant for any kind of alteration or modification, because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks [5].

Our aim is to replace the traditional client-server paradigm with the client-blockchain alternative. And build a pro-totype of a web service that will allow to store PII data securely, with the assumptions and requirements explained above.

Another huge problem is data in transit. The current client-server paradigm assumes establishing connection be-tween communicating parties as often as possible. Because data is in transit all the time and travels between server and client on regular basis, it is an easy task for the attacker to gain access to it using techniques like man-in-the-middle, session hijack or similar.

Our counterproposal is to limit amount of these connections to an absolute minimum. We want to achieve this by storing data locally on user device and exchanging it with the server (actually: blockchain) only 2-3 times per session.

3. Problem analysis

When we want to limit amount of communication between the server and the client as much as possible, we en-counter following problems:

1. Data desynchronization between sessions.
2. Security of the decryption and encryption key when using the browser by different users
3. Possibility of data loss, or the inability to recover data in the event of a loss of the key
4. Problem of finding out which user was working on given blob and saving that blob to a correct record in the database.

These four problems will be now further discussed.

3.1. Data desynchronization between sessions

The problem occurs in the following scenario. The user was working on one of their devices. Let us call it Device A. The user has performed tasks or duties, updating their locally stored data blob. But has not finished their session on that device and switched to using the webservice on a different computer, device, or browser. Let us call it Device B.

Because session was not finished on Device A, the server (or actually — blockchain) has not received the most up-to-date data blob. Starting activity on Device B causes that device to receive outdated data blob from the blockchain, making user working on outdated PII data. This is not the only problem. Supposes that the user decides to continue his work, not even knowing that they are working on outdated data blog. If user ends their session on Device B and causes data blob to be sent to blockchain, that update will override or otherwise invalidate "older" blob update, still waiting in the Device A, because of unfinished session there.

One of the proposed solutions to the problem is to limit sending encrypted and locally stored data blobs to block-chain to a minimum, while keeping sending information about active session (and changed to data blob occurring local-ly) to the blockchain or supporting database as often as needed. This would allow to store information about active sessions and warn the user that starting a new session on given device, while another session is kept active on another device, will lead to a data loss or corruption.

3.2. The security of the decryption and encryption keys on shared computers

In our proposed solution everything is stored on the client side, in web browser's local storage. When the same device or web browser is used by more than one user then this information (namely encryption and decryption keys) can possible be made available to all other users.

To solve this problem, we would have to consider storing encryption and decryption keys in a location other than browser's local storage. Or to provide end user with an ability or a tool to clear the decryption key and set it back. Or to encrypt the encryption and decryption key with other hash, salt, or data. Or introduce any other solution for separating different key sets stored and used on the same device. This is one of the areas to be explored together with the planned prototype and proposed experiments.

3.3. The possibility of data loss, or the inability to recover the data in the event of a loss of the key

In our proposed solution, the entire data blob is stored locally and encrypted with encryption key know only to data owner (the user). Therefore, in the event of a decryption key lost (intentionally or not intentionally — i.e., accidentally clearing of the browser's cache), the user loses access to all the data. It is impossible to decrypt or gain access to the data blob in any other way than by providing a correct decryption key.

This is a well-known problem that can be found i.e., when encrypting disks or pendrives using BitLocker or similar software. The solution is also an established industry standard, and it narrows down into asking the user to store de-cryption key in a safe location. Print it on a piece of paper, save it to an USB drive or store it within on-line storage [9].

Similar solutions can be considered in case of our proposed solution. User would be requested to remember, write down, print out or store securely the decryption key and would be provided with the possibility of re-entering it in the event of any decryption key loss (like in mentioned situation of clearing browser's local storage).

3.4. Distinguishing of users on the blockchain side

In typical client-server paradigm, sensitive data is stored on the server and local client (namely a web browser) is working with the data stored remotely. Therefore, it must always provide the server with any kind of identification. Based on that ID, server "knows" which client is sending requests and can save data in a correct record in its database.

In a proposed solution, sensitive data is stored locally and exchanged with the blockchain only on a rare basis. There is no ID kept on server or locally and therefore the blockchain may not "know" to which user currently sent data blob belongs and thus, to for which user given blob should be saved.

One of the solutions for this problem is to use decryption key also as a unique identifier of the user and the data blob that it decrypts. Decryption key would be generated on server-side in this case. Unfortunately, this solution assumes that the decryption key will be travelling through the Internet together with the data blob that it can decrypt, which poses a security risk. To resolve this issue our proposed solution must possible include storing the decryption keys in other location than data blobs, for example inside Amazon KMS system. Or use other ways of uniquely identifying of data blobs and their corresponding users [10].

We will now discuss two different approaches into design and development of a prototype system based on above mentioned criteria, assumptions, and requirements. Both use case scenarios assume using blockchain, but first one only uses transaction signing using blockchain platform while second scenario assumes using of a crypto wallet and storing decryption key in one of them.

4. Scenario one: Locally stored decrypted data

Our first suggestion covers the use of the method of signing transactions on the blockchain platform.

When the user enters our prototype website, we read the information from the browser's cache, namely the de-cryption key and date of the last operation, if any.

If both pieces of information are missing, two buttons are displayed, to allow: (1) user to create a new account and (2) reading of the data blob. User is requested to provide (into an input field) the decryption key. It will be used both as decryption password and a unique identifier, to read a correct blob from the database and save it to the local storage.

When the decryption key is available, we read the data blob from the server. When the date and key are available, we immediately display the data and allow the user to continue his work where he has finished it.

4.1. Registration of a new decryption key

When the user creates a new account, a decryption key is generated for the user on the server side. Once generat-ed, it is saved on the server and assigned to an information blob — empty for now. Both decryption key and empty data blob is returned to the client (user's web browser) during this first query.

The key is then cached (on client-side) along with the already decrypted blob of information. In addition, the user has the option to copy their key, to save it somewhere (e.g., in KeePass or on a piece of paper).

The web service interface also provides two more buttons: (1) for clearing the cache (this operation deletes data blob and the decryption key, both stored locally), and (2) for end work (this operation encrypts the locally stored data blob and sends it in an encrypted form to the server along with the decryption key). The second operation (end of a work) also removes the blob from the cache, but it leaves the key. So, the next time users start using our prototype webservice, the data will be received from the blockchain and made available to the user automatically and instantly.

This operation (registration of a new decryption key) in client-blockchain paradigm corresponds to a typical oper-ation of registering new account in client-server communication. The only difference is that entire dataset for the whole user account and all operations available on it is stored in the blockchain, not in the server, and is stored in an encrypted form. It is available in unencrypted form for the local device only.

4.2. Use of existing decryption key

User with an existing account provides a decryption key to an input field. The key is sent to a server (a block-chain) and is used as an identifier. Remote party (blockchain or server) sends back an encrypted blob of data that corre-sponds to the received decryption key. Local application decrypts received blob of data and allows user to use the appli-cation and interact with their PII data with extremely limited or no interaction with the server.

4.3. End of work

Because user is working with its data and the whole web application entirely locally, there is a need to introduce a new operation that will force-synchronize data with the remote part (backend, server, or blockchain). We call it initially "end of work" stage. It is comparable to forced logoff. In our client-blockchain paradigm user must logoff (meaning that they must intentionally end their current work) to force data synchronization with the remote part of the application.

If this part is skipped, then we are running into a problem described in point 3.1 above.

4.4. Change of an existing decryption key

It is possible to generate a new encryption key. A query to the backend (blockchain or server) with the old decryp-tion key will cause it to generate a new one and return it to the user. During the next "end of work" stage locally stored PII data (data blob) will be encrypted with the new encryption key. During the generation of the key on the server side, we need to decode the blob, using old decryption key and encrypt it again with the new encryption key.

Due to the problem described in point 3.1 above it might not be possible to change the decryption key in device X if there is an active session maintained on another device. Doing so will cause data corruption, because data blob kept in blockchain would already be encrypted with the new encryption key while other device would still try to decrypt it with the old key, kept in that device's local storage.

4.5. Other scenarios

If decryption key is found in local storage and it is verified with the remote (blockchain or server) that it is valid, but the data blob is missing in local cache then this scenario continues as described in point 4.2 with only exception that the application does not display any input field and does not ask user for the description key, but rather uses the one from the local storage. It sends it to the backend and receives encrypted data blob from where, which it then decrypts locally.

If both valid decryption key and locally stored data blob is found and if date, timestamp, or other identifier used confirms that this data blob is the most recent one then we use it for a normal local web application operation and we do not send any query for data blob to the remote to not risk data corruption.

4.6. Summary of scenario one

In this use case scenario (Locally stored decrypted data) we assume that the communication between client and the remote backend (blockchain or server) is limited to only three operations: (1) registering of a new account / creating data blob, (2) receiving encrypted data blob after providing valid decryption key each time and (3) sending encrypted data blob after “end of work” operation to synchronize local data changes with the remote backend.

5. Scenario two: Ethereum-type blockchain

The second scenario is not an alternative to the first one, but more like an extension to it. It extends the first sce-nario by the means of reusing blockchain-available solutions everywhere it is possible [11]. This includes: (1) replacing server with Ethereum-type blockchain and thus creating a decentralized application (dApp [12]), (2) replacing typical server-side logic with blockchain-based smart contract and (3) replacing local application with blockchain wallet.

In this scenario we would have to create our own prototype of blockchain, create a smart contract and reuse existing blockchain wallet software (i.e., MetaMask [13] [14], Coinbase [15]) or develop own one. Our smart contract, stored within blockchain, would have minimal logic, reduced to storing, reading, and saving of data blobs. All the remaining logic, including data blob decryption and encryption, would be still purely client-side, but based on blockchain wallet this time.

We would use smart contract as a backend in this scenario and we would limit communication between it and the blockchain wallet again to the absolute minimum — reading of encrypted data blob in the beginning of each session and saving encrypted data blob, containing all the data modified during current session, at the end of it. And the entire solution would reuse the decentralized apps (dApps) paradigm [12].

With this approach we are reusing the most important (from security standpoint) feature of dApps — the decen-tralization of data. In this paradigm there is no one single server, but a blockchain instead. To alter data stored in this way and attacker or intruder would have to perform a successful attack on all the blockchain’s nodes together and in the same moment of time. The other alternative is to attack user’s computer directly, because all the logic and application’s functionality is performed locally on user device. Data exchange is limited to a minimum, so chances for implementation of a man-in-the-middle, session-hijack or similar attack vectors are reduced to also a minimum [12].

As mentioned in the introduction to this article, any write-data operations performed on blockchain are subjected to a small fee (gas fee) paid to the person or company sharing their hardware for processing transactions inside block-chain and signing transactions stored in blockchain. This fee must be included in any discussions about the proposed solution. There are two options to manage it. One is to run proposed prototype based on own blockchain. Second is to include fee in solution’s business model. The first solution also includes monthly costs, for running servers needed to power own blockchain. So, this again narrows down to including costs in business model.

6. Pros and cons

6.1. Proposed solution in general

We have identified the following pros of the proposed solution in general:

1. Undisputable and definitive security of the data
2. Could be interesting to the people or parties interested in data security
3. Could be especially useful in scientific experiments and business applications where user is storing their most sensitive, private, and intimate data, like Facebook alternatives

So far, we manage to identify only one con of the proposed solution — that is already mentioned user experience downside. Where user is forced to perform additional operations ("end of work") to maintain data synchronization be-tween sessions. They might not be used to this since other applications (in classic client-server paradigm) does not require users to perform such.

6.2. Pros and cons for scenario one

Pros:

1. Total control over data flow and application flow
2. Fast flow of information in case of creating an API
3. Key and data security (stored on the client's side)
4. Free for users (no gas fee needed in business model)

Cons:

1. Problems for the user in case of loss of a decryption key, or the danger of storing it in the database
2. Possibility of hacking the server and data by the attacker (low chance, but still)
3. Session security (leaving the browser, not logging out)

6.3. Pros and cons for scenario two

Pros:

1. Data safety (stored locally)
2. Total control over the application flow
3. Additionally increased data security when using public blockchain with a high number of nodes
4. Decryption key safety (located on the client side in an encrypted wallet)
5. Comprehensive account recovery system already in place (due to using wallet's software solutions)

Cons:

1. Slower data flow due to the fact of using a public blockchain
2. Gas fee when using a public blockchain with a high number of nodes
3. Sever costs and at least theoretical chance of intruder injecting corrupted data into blockchain when us-ing private blockchain with reduced number of nodes

Both solutions are also expensive in terms of time and workforce for the design, development, implementation, and management. This means that a significant amount of funds must be secured for even development of the prototype.

7. Conclusions

Today implemented IT applications and systems uses the classic client-server paradigm [16]. It assumes a con-stant data exchange between client and server and storing data in a classic server. The proposed solution suggests considering a change in this, to client-blockchain paradigm with the limitation of interaction between client and backend (blockchain) to an absolute minimum.

The proposed solution lacks certain aspects from user experience area and this must be further investigated. For example, the fact that user must intentionally perform "end of work" and logoff from one device (to force data blog synchronization) to be able to work on another device is a problem that might not be widely accepted. Most today users are expecting that their application or a web service will be working in classic client-server paradigm, in which data is exchanged with the server on regular basis and do not require users to act specially in certain scenarios.

On the other hand, this proposed solution minimizes number of interactions and data exchanges between local client and the remote backend to an absolute minimum. Thanks to this and to other factors, it increases data security to an important level, by the cost of UX. It may therefore turn out that the proposed solution will be usable only in a specific, scientific conditions like for example certain IT data experiments. And that it might not be usable widely in commercial application or solutions.

Affiliation

Tomasz Trejderowski is an author of the proposed idea of a client-server solution that stores entire dataset on the server in a completely encrypted form and defined the main problem, which could be solved with the proposed proto-type. He also putted all the final conclusions together.

Maciej Madejczyk was responsible for technical details of the proposed solution.

Beata Trejderowska was acting as a business analyst and was responsible for defining all the functional and non-functional requirements needed for solving the defined problem and building the proposed prototype.

Bibliography

[1] Stevens, G. (10 April 2012). "Data Security Breach Notification Laws". Congressional Research Service 7-5700, R42475. Page 10. Available at: *www.sgp.fas.org/crs/misc/R42475.pdf*. Retrieved 8 June 2022.
[2] McCallister E., Grance T., Scarfone K.: "Guide to Protecting the Confidentiality of Personally Identifiable Infor-mation (PII)". National Institute of Standards and Technology, NIST Special Publication 800-122. Page B-1. Avail-able at: *www.nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf*. Retrieved 8 June 2022.
[3] "Personal Data". General Data Protection Regulation (GDPR). Available at *www.gdpr-info.eu/issues/personal-data/*. Retrieved 21 May 2022.
[4] Morris, David Z. (15 May 2016). "Leaderless, Blockchain-Based Venture Capital Fund Raises $100 Million, And Counting". Fortune. Available at *www.fortune.com/2016/05/15/leaderless-blockchain-vc-fund/*. Retrieved 23 May 2022.
[5] Wikipedia contributors. (7 June 2022). "Blockchain". In Wikipedia, The Free Encyclopedia. Available at *www.en.wikipedia.org/w/index.php?title=Blockchain&oldid=1091997775*. Retrieved 12 June 2022.
[6] Elrom, Elad (2019). The Blockchain Developer., Apress, *www.doi.org/10.1007/978-1-4842-4847-8*
[7] Fries, Martin; P. Paal, Boris (2019). Smart Contracts (in German). Mohr Siebeck. ISBN 978-3-16-156911-1
[8] Kuner, Christopher, Bygrave, Lee A., & Docksey, Christopher (2020). Background and Evolution of the EU General Data Protection Regulation (GDPR). The EU General Data Protection Regulation (GDPR), Oxford University Press, <*www.doi.org/10.1093/oso/9780198826491.003.0001*>
[9] Kornblum, Jesse D. (2009). Implementing BitLocker Drive Encryption for forensic analysis. Digital Investigation, 5(3), 75-84, ISSN 1742-2876, Elsevier BV, <*www.doi.org/10.1016/j.diin.2009.01.001*>
[10] Martin, Keith M. (2017). Public-Key Management. Oxford Scholarship Online, Oxford University Press, <*www.doi.org/10.1093/oso/9780198788003.003.0011*>
[11] Besancon, Leo, Silva, Catarina Ferreira Da, Ghodous, Parisa, & Gelas, Jean-Patrick (2022). A Blockchain Ontology for DApps Development. IEEE Access, 10, 49905-49933, ISSN 2169-3536, Institute of Electrical and Electronics Engineers (IEEE), <*www.doi.org/10.1109/access.2022.3173313*>
[12] Metcalfe, William (2020). Ethereum, Smart Contracts, DApps. Economics, Law, and Institutions in Asia Pacific, 77-93, ISSN 2199-8620, Springer Singapore, <*www.doi.org/10.1007/978-981-15-3376-1_5*>
[13] Gallersdörfer, Ulrich, Ebel, Jonas, & Matthes, Florian (2022). Augmenting MetaMask to Support TLS-endorsed Smart Contracts. Lecture Notes in Computer Science, 227-244, ISSN 0302-9743, Springer International Publishing, <*www.doi.org/10.1007/978-3-030-93944-1_15*>
[14] Lee, Wei-Meng (2019). Using the MetaMask Chrome Extension. Beginning Ethereum Smart Contracts Program-ming, 93-126, Apress, <*www.doi.org/10.1007/978-1-4842-5086-0_5*>
[15] Kazerani, Ali, Rosati, Domenic, & Lesser, Brian (2017). Determining the usability of bitcoin for beginners using change tip and coinbase. Proceedings of the 35th ACM International Conference on the Design of Communication, ACM, <*www.doi.org/10.1145/3121113.3121125*>
[16] Abts, Dietmar (2022). RESTful Web Services. Masterkurs Client/Server-Programmierung mit Java, 331-440, Springer Fachmedien Wiesbaden, <*www.doi.org/10.1007/978-3-658-37200-2_12*>
[17] Pimparkhede, Kunal (2021). Client side and Server Side Load Balancing. International Journal for Research in Ap-plied Science and Engineering Technology, 9(11), 30-31, ISSN 2321-9653, International Journal for Research in Applied Science and Engineering Technology (IJRASET), <*www.doi.org/10.22214/ijraset.2021.38748*>

OCHRONA DANYCH WRAŻLIWYCH Z WYKORZYSTANIEM BLOCKCHAIN

STRSZCZENIE

Prezentowany artykuł podejmuje dyskusję nad naukowymi i technicznymi możliwościami wykorzystania sieci blockchain w procesie ochrony danych osobowych. W artykule przedstawiono kluczowe aspekty proponowanego pomysłu, jak również większość zalet i wad. Zawarto również krótkie omówienie możliwego prototypu proponowanego rozwiązania.

SŁOWA KLUCZOWE

dane wrażliwe, ochrona danych, sieci blockchain