privEOS

Angelo Laub, Fabian Frank, Marcel Vaschauner, Michael Breidenbruecker,
Stefan Ionescu, Tassia Breidenbruecker

Slant Team

September 21, 2018

Abstract

We present a GDPR-compliant privacy solution for dApps on the EOS blockchain. Our infrastructure provides dApps with a simple API to store private data and delegate permission management to a smart contract.

Contents

1 Background

Distributed Applications (dApps) running on the blockchain have the potential to disrupt classical online businesses by increasing efficiency and transparency, not requiring trust in the operator, as well as reducing costs by eliminating middlemen. We are already seeing a growing number of decentralised counterparts to centralised businesses on the web. Still, a critical problem that blockchains need to solve is the lack of scalable systems to encrypt and aggregate private data without the need of a central party. In this paper we present such a system that has two desirable properties. First, each data point can be decrypted with a key that is split between a large number of parties, thus lowering the risk of collusion. Second, the system uses the safety guarantees of Intel SGX enabled machines which can decrypt data inside a secure enclave and process it without any other party being able to extract meaningful information. Due to these properties that enterprises need, we expect the number dApps and real-world adoption of dApps to increase dramatically in the near future.

2 The Problem: Storing Private Data

Since everything on a blockchain is public and readable by everyone, storing personal data poses huge challenges for dApps. Of course, data can be encrypted, but key management is a hard problem. Once data is encrypted, it cannot easily be shared with new recipients. So far, no best practices have been established to deal with these problems. Recently introduced Data Protection Regulation (GDPR)[1] in the EU have made it even more difficult to be in compliance.

3 Privacy Solution for dApps

Our solution allows dApps to store files in an encrypted way while retaining the ability to grant reading rights to interested parties. The permissioning of access

[1] https://gdpr-info.eu/

to that data can be delegated to a smart contract that adopts our proposed smart contract protocol.

3.1 Private Data Store

3.1.1 Simple API

We provide a simple API that lets dApps store data in an encrypted way. A smart contract protocol can be used to access the file. The granting of read permissions to new users can happen both explicitly through user input (interactive case), as well as non-interactively. The non-interactive use-case is particularly powerful. The conditions under which a new user gets read access to a file are freely programmable. A smart contract could for example say that only users who have paid 5 EOS may access a file.

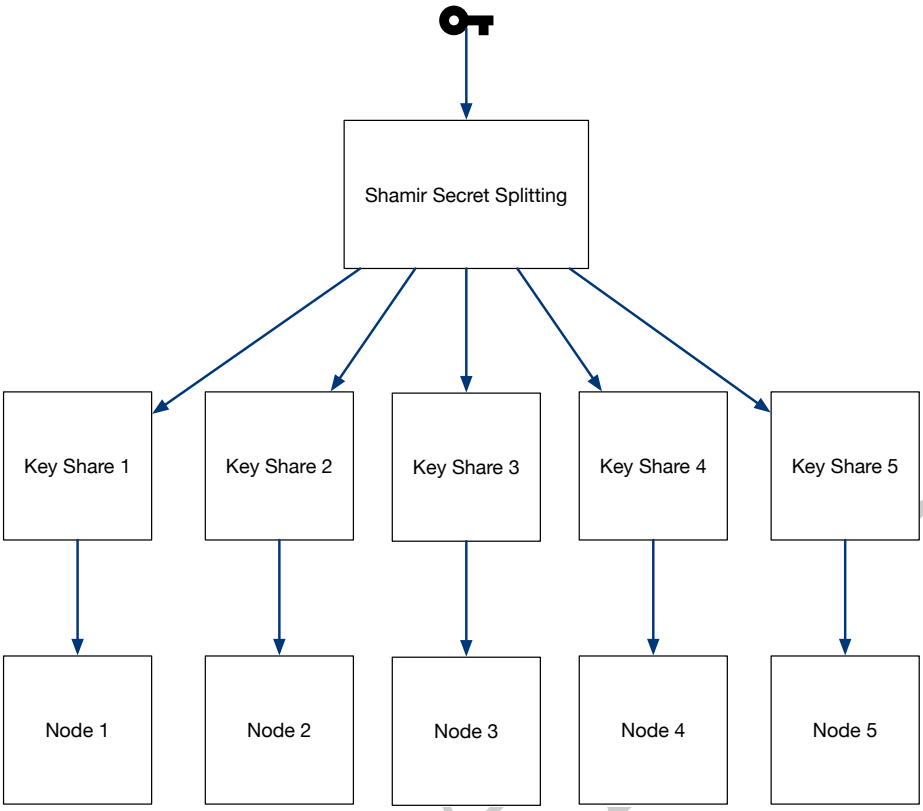
3.1.2 Encryption scheme

Our decentralised encryption scheme combines classical Symmetric Encryption, Elliptic Curve Public Key Cryptography and Shamir's Secret Sharing[2].

3.1.2.1 Threshold Encryption

Our system uses the threshold version of Shamir's Secret Sharing algorithm to split up a secret into N shares, of which $M < N$ are required to reconstruct the plain text.

[2]https://cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf



The figure above shows the splitting of a secret key into 5 pieces that will be stored on 5 different nodes. The key splitting algorithm can be configured to require any number of keys in order to be able to reconstruct the secret. For example we could configure the algorithm to require a threshold number of 3 of the 5 key shares to be able to reconstruct they secret key. This ability makes our network robust against lost keys as well as secure against single leaked keys.

3.1.2.2 Publishing a file

1. Alice generates a random key for symmetric encryption. That key is specific for one file and will only be used once.
2. Alice encrypts the file using the random key and stores the encrypted file in IPFS
3. Alice uses Shamir's Secret Sharing to split the key into N shares with a threshold of $M < N$ $(M, N \in \mathbb{N})$.
4. Alice encrypts each of those N shares with the public key of the respective node before sending it over the network.
5. The encrypted shares are distributed to the N participating nodes, one each, where they will be safely stored by our open-source key management software.

3.1.2.3 Accessing a file

1. Bob requests access to a file.

2. If Bob should be granted access according to the rules specified in the smart contract, a transaction will appear in the blockchain indicating Bob is authorised to access that file.
3. Each node encrypts their share of the symmetric key using Bob's public key
4. Each node sends their share to Bob
5. Bob decrypts each share using his private Key
6. As soon as Bob has received M shares, he can reconstruct the original symmetric key that was used by Alice to encrypt the file
7. Bob can now use this key to decrypt the file

3.1.2.4 Deleting a file

The owner of the file or whoever the smart contract authorises can issue a delete command. This instructs all nodes to delete their share of the key and no longer serve key sharing requests for that file. Additionally, the owner can choose to delete the actual encrypted file, wherever it may be stored. This ability enables applications to satisfy the right to be forgotten demanded by the GDPR.

3.1.2.5 Properties of this Encryption Scheme

The scheme uses well known and battle-tested cryptography. It can be implemented using existing and already tested software libraries.

Each node only sees their shamir share of the symmetric key, which means any individual node can never decrypt the file. In order to illicitly decrypt the file, M nodes would need to collude. Given sufficiently large numbers of M, this becomes increasingly unlikely.

Access to new Bobs can be granted automatically by a smart contract in a non-interactive way.

3.1.3 Relationship to EOS storage

Our system is designed to work as a companion to EOS Storage[3], whenever the need arises to store private data. EOS Storage provides the storage and bandwidth capacity for the actual files, we provide the decentralised key management. In practice though, any storage can be used that allows accessing the file via a URL, for example IPFS or Amazon Web Services.

3.1.4 Offline Data Access

Certain applications require offline access to data, either for security reasons or due to being deployed at a remote location with bad connectivity. Since the actual data can be stored anywhere by the application, it can also be stored on an on-site file server if desired. Alternatively, an IPFS node that pins the needed files can be deployed on-site.

[3] https://github.com/EOSIO/Documentation/raw/master/EOS.IO%20Storage.pdf

3.1.5 Key Revocation

An additional benefit of storing files in a private data store is the ability to revoke keys. If a key needs to be revoked, the file can simply be re-encrypted with a new key and deleted from the private data store.

3.1.6 Storing Multiple Files

If a number of files need to be shared, like a directory tree, the simplest option is to just publish a zip file or a tarball. Encrypted disk images, as they can even be mounted remotely, offer a very efficient way to share any number of files with the same recipient.

3.1.7 Mutable Files

Since files are represented by URLs, it is even possible to share files with dynamic content. This is of course not possible when the file is store in IPFS, it would have to be stored on a normal web server. The only requirement is that the recipient of the data has access to the URL where the file is stored.

3.1.8 Digital Rights Management

Digital rights management is a hard problem and a controversial issue. Cryptographic systems can prevent unauthorised access to data. But when an authorised person has read access to the data, they can copy it and potentially share it with others. Cryptographic techniques alone cannot prevent this issue.

> None of this technology that you're talking about's gonna work. We have Ph.D.'s here, that know the stuff cold, and we don't believe it's possible to protect digital content.
>
> – Steve Jobs on DRM[4]

The realm of digital rights management is therefore outside the scope of our solution.

3.1.9 Objectionable data

In general, file hosting services face the issue of potentially hosting objectionable or illegal data. In our system, the nodes don't store the actual files, but only shares of encryption keys. Nodes have no knowledge of the nature of files that might or might not have been encrypted with the key they possess a share of. They have also no way of decrypting said file as M shamir shares are required to decrypt it, yet they posses only one share. Since no actual files are hosted, objectionable data will most likely not pose any problems in our system.

[4] https://www.rollingstone.com/culture/culture-news/steve-jobs-rolling-stones-2003-interview-243284/

3.1.10 GDPR and Node Liability

The european General Data Protection Regulation (GDPR) imposes fines on applications which are non-compliant. It is up to the developer of the application to design their application in a way that makes it compliant with the GDPR. Since the nodes in our network just provide applications with key management functionality, a node would not be liable for the non-compliance of an application using its services in the same way a file hosting provider would not be liable for failures of its customers to comply with the GDPR. Furthermore, as described in the previous section, nodes don't store the actual files and have no knowledge of the nature of the files that might or might not have been encrypted with the key they possess a share of.

3.2 Privacy Preserving Computation

Certain applications require the ability to perform computation on a private data set. In order for the data to remain private, it must not be shared with anyone in the decrypted state. Homomorphic encryption has been proposed as a solution for this problem. However, current Fully Homomorphic Encryption implementations are far too slow to be of practical use. Until Fully Homomorphic Encryption becomes practically usable, we will be using Intel Software Guard Extensions (SGX) which keeps the data in a secure enclave where computation can be performed.

3.2.1 Intel Software Guard Extensions (SGX)

Intel SGX[5] provides applications with a secure environment that lets you perform computation privately and without needing to trust the operating system it is running on. That means an encrypted data set could safely be decrypted inside an SGX enclave and computation performed on the plaintext. As long as the Intel SGX security model is not broken, no trust would be required in the node operator.

3.2.2 Private Computation Scheme

1. Application submits computation task, indicating the data set to work on and the webassembly bytecode.
2. One of the nodes picks up the task, launches an SGX instance
3. Once integrity of that SGX instance has been verified, the code running inside the SGX instance may request access to the shamir shares
4. The code inside the SGX decrypts the shamir shares, reassembles the private key and can now decrypt the data set
5. The webassembly bytecode provided by the application is executed inside the SGX enclave, performing the desired computation on the data set.
6. The result of the computation will be encrypted with the intended recipient's public key and published

[5] https://software.intel.com/en-us/sgx

3.2.3 Providing Computational Resources

Performing these computations on public nodes presents a number of challenges. The resources of the nodes would have to be shared with all apps requiring computation. A relatively complicated pooling and reputation system would be required. Data sets could differ widely in size. Some, being only a couple of kilobytes, and some being in the terabytes. A mechanism similar to Gas would be needed to objectively quantify resource usage of computation tasks. More complex tasks based on big data sets would likely be very expensive. The Golem Project[6] has built such a node network based on Intel SGX technology. However, it does not yet allow general computation by everyone.

Since Intel SGX provides secure and verifiable computation on a single host, applications could run their own private computing nodes. The application developer can adapt the resources of their server to their specific computing needs. This would give developers the maximum flexibility while still maintaining the privacy preserving properties of the system.

4 Node Network

Our solution requires a network of nodes running our open-source key management software. The ideal network has a large number of independent nodes and is resistant to sybil attacks. Nodes should also be economically incentivised to behave properly.

4.1 EOS Block Producers

Building such a network is expensive and difficult. That's why we propose to use EOS Block Producers (BPs) as node operators, since they already possess all of the properties we desire.

1. BPs are well known individuals or companies, which means the network is resistant against sybil attacks.
2. BPs are elected by the EOS community, so they are trusted community members.
3. BPs are well paid for their services by the blockchain. Any malicious behaviour can be grounds for removal of that BP which means they will loose their source of revenue. This means that BPS are highly incentivised to behave properly.
4. BPs are in friendly competition among each other and they are encouraged to provide additional tools and infrastructure that benefits the chain and applications running on top of it. BPs are constantly trying to outperform each other on the hunt for more votes. That's why it's likely that our open source software, if it provides a tangible benefit, will be adopted by a majority of BPs.
5. They are already running a highly dependable server infrastructure and have proven expertise.

[6] https://golem.network/

We are cordially inviting the top 100 BP candidates on EOS to become nodes.

4.2 Communication protocol

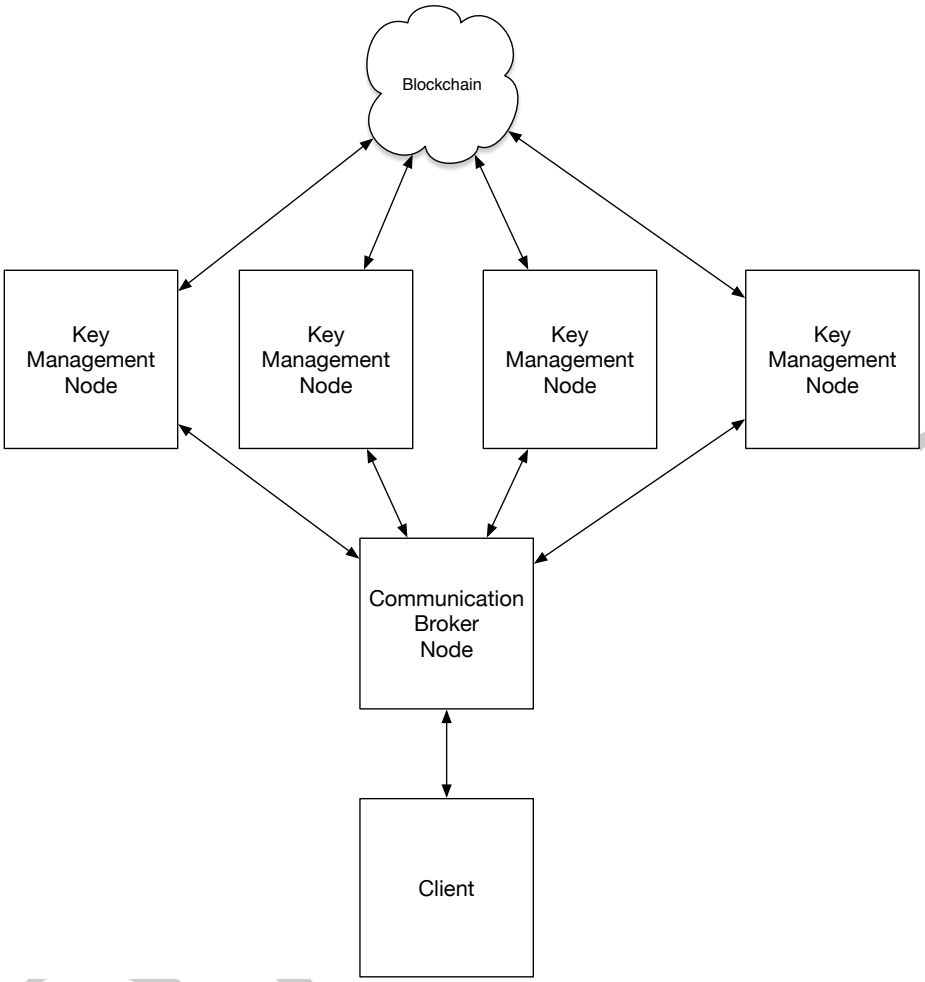
4.2.1 Open Source Key Management Software

Node operators will be paid by the privEOS contract to run our open source key management software. EOS Block Producers would want to run it on a separate server, physically separated from the block production. When a user stores a file, this will be documented in the blockchain by a transaction. Nodes only accept file storage requests legitimised by a transaction in the blockchain. When a user requests access to file, a transaction will be written to the blockchain documenting that the user has been granted reading rights to a certain file. Only in the presence of such a transaction will the nodes grant user requests to access certain key shares.

4.2.2 Open Source Communication Broker

Since requesting read access requires the client to communicate with N nodes, performance can be improved by introducing a communication broker. Users with a slow internet connection, especially mobile users, will profit most from this optimisation. Any public communication broker can be used by an application or the application developer can decide to host their own. It is a requirement for nodes participating in our network to provide a public communication broker. Since all messages exchanged between client and server are encrypted and signed using public key cryptography, those messages can be exchanged over an insecure channel, including adversarial communication brokers.

4.2.3 Architecture overview



4.3 Casper-inspired Staking

As an additional layer of incentivisation, we require the nodes to stake 10,000 SHAMIR tokens in the beginning. Additionally, a rolling reserve of 10% of the ongoing earnings need to be staked for a period of one year.

Nodes must agree to an SLA of 99.9% uptime. That is equivalent to a downtime of 86 seconds per day. Since we are in a threshold system with an inherent redundancy, the service continues to function even if several nodes are offline at the same time. Thus, the actual service level experienced by apps will be higher.

Downtimes exceeding the agreed upon SLA will result in partial slashing of the stake, similar to Casper[7].

[7] https://arxiv.org/pdf/1710.09437

Nodes agree that in case an EOS arbitrator finds them to be in violation of the node ricardian contract, their entire stake might be slashed. As EOS BPs need to be in good standing with the community, in addition to that stake, depending on the severity of the misconduct, they might lose their position as EOS BP as well.

4.4 Cheating detection

When Bob accesses the file, the individual shamir shares received from the nodes can be checked against their hashes that were recorded in the blockchain. If a node were trying to slip in invalid data, it would immediately be caught.

4.5 Proof of storage

Samples of shares stored on nodes can be requested anytime and checked against their respective hashes on the blockchain. These samples would be encrypted with the public key of the node and could safely be shared with the validator.

4.6 Shutdown of Nodes

If a node wants to exit the network, a shutdown process needs to be followed before they will get their stake back. The node wanting to leave needs to hand over the shamir shares to new nodes that have joined the network in the meantime. This is to ensure that there will always be the threshold number M of shamir shares available from active nodes. Once handover has completed successfully, the nodes are required to delete all data securely and make sure all backups are destroyed. To confirm that they have fulfilled their legal obligation to delete all data, the node operator will sign a deletion-completed transaction on the chain. After 3 months, if there is no active arbitration case against the node, the node can reclaim their stake.

5 Possible Applications

5.1 GDPR Compliance

Decentralised Applications that need to store personal data can use our system to store files securely. Data ownership can remain with the customer. That means the customer can define the conditions under which data may be shared with others, if at all. Smart contracts can be used to enforce these access rules. This allows you to build decentralised applications compliant with the european Data Protection Regulation (GDPR)[8].

[8] https://gdpr-info.eu/

5.2 Zero Knowledge Services

With our solution, it is possible to buidl applications where not even the operator of the service can access the data. For example a pizza ordering marketplace could be built where only the pizzeria receiving the order would be able to see the order. All personal data of the customer could remain private even to the operator of the marketplace. For customer care or arbitration purposes, the user could grant read access to the order data to a specific customer care agent on a case by case basis, should the need arise. There would also be no central database that could be hacked, putting the users' personal data at risk.

5.3 Selling Data

A social media website could be built where users can retain full data ownership. Users could define the conditions under which data may be shared with third-parties and at what price.

5.4 Data Marketplaces

Data marketplaces could be built that don't require the seller to interact every time someone buys a data set. The price and other conditions could be defined by the seller in the beginning and automatically be enforced by a smart contract.

5.5 CIA Agent Life Insurances

An agent whose life is under threat could program a secret dossier to be published in the case of his death or becoming otherwise unable to regularly sign transactions on the blockchain.

5.6 Last Wills

The private keys of a crypto fortune could be disclosed to the heirs after a certain period of not having signed an "I'm alive" transaction.

5.7 Auditable Access Logging

Every time read access to a file is granted to someone, an unforgeable record is made in the blockchain.

5.8 Uncensorable File Sharing

A decentralised file sharing service could be built where uploaders of files could set prices for downloads. It would be difficult to censor or shut down.