Blockchain and Privacy: Concepts and techniques[*]

Working Group on Blockchain and Privacy
Innovation Network for Finance IT

2018/11/26 at 15:59:48

Abstract

This project investigates how blockchain technology can be made consistent with GDPR and provide a platform for auditable GDPR-compliant personal information management. The present report contains security and privacy related concepts and techniques for managing personal information in a blockchain technology context.

1 Introduction

The EU General Data Protection Regulation (GDPR)) [Par16] aims to protect the privacy of natural persons in the EU by enforcing strict regulations around the processing of personal data. As such, it defines concrete rights for individuals to control and restrict how their data is used, and sets out obligations and principles for those in charge of processing.

The GDPR provides a legal framework for treating personal data. However, in order to realize its requirements in practice, a practical analysis is necessary as well.

For one, while the GDPR establishes terminology around data processing, many of the terms are important to interpret in a modern computational context. This interpretation is necessary in order to inform the implementation of concrete, compliant solutions.

Although the GDPR lays out principles for data processing which implicitly require capabilities on the part of those handling personal data, the technical means that provide these capabilities are left to individual interpretation.

Finally, the GDPR ties some of its definitions to the state of the art in technology. For instance, this applies to defining appropriate standards for data protection.

It is therefore prudent to *(i)* establish an interpretation of the definitions provided in the GDPR from a computer science and technological state-of-the-art perspective], *(ii)* identify current and emerging technologies that deliver the required capabilities and adequately honor the data processing principles laid out in the GDPR, and *(iii)* determine to what extent emerging technologies may affect, constructively but also adversely, what is "realistic" in the context of GDPR.

In this report, we set out to address these points.

In particular, we summarize the terminology of the GDPR (§ 2), propose an initial technical interpretation (§ 3), and finally explore solutions and technical challenges pertaining to the GDPR in the context of two emerging technologies; blockchain and distributed ledger systems on one hand, and privacy-enhancing technologies on the other (§ 4).

[*]Milestone 1 report (draft); for internal distribution only.

2 Terminology

The GDPR builds terminology around the processing of personal data. It defines the rights of those contributing personal data, and sets specific obligations for those handling it.

We focus on three key actors in the context of this report.

(i) **Data Subject.** An identifiable natural person that entrusts personal data to another entity for processing.

(ii) **Data Controller.** A natural or legal person that receives the personal data. The data controller determines how and to what end personal data is processed within an organization. The data controller is responsible for ensuring compliance with the GDPR. If data is processed jointly, across multiple organizations, the task of controlling how the data is processed may be distributed across multiple "joint controllers" (Article 26). An example of a data controller is a private business providing a web-service.

(iii) **Data Processor.** A natural or legal person that performs data processing, on behalf of a data controller. The data processor, like the data controller, has obligations within the GDPR. A data processor may, for instance, be a cloud service provider utilized by a data controller such as the web-service provider in the example above.

Whenever personal data is gathered for processing, it is required that the data subject first provides consent (a few exceptions to this rule are described in Article 6). A data subject can furthermore exercise rights to control his or her personal data and gain insight into how it is being used.

(i) **Right of access.** A data subject may request information about his or her personal data, including the purposes of its processing, data provenance, and justifications for data-driven decisions made by a controller (Article 15).

(ii) **Right to withdraw consent.** A data subject may at any time withdraw consent to data processing (Article 7). Note that this also entails subsequent data erasure in most cases.

(iii) **Right to rectification and erasure.** A data subject may request the correction of erroneous data (Article 16) as well as its deletion (Article 17).

(iv) **Right to portability.** A data subject may request his or her personal data form the data controller. Also, may request the data's transfer to another data controller in a structured, commonly used and machine readable format (Article 20).

In order to ensure these rights, along with providing additional safeguards, data controllers and processors must adhere to a set of core principles (Article 5). The following are of particular interest in the context of this project:

(i) **Fairness, transparency.** These principles concern the data processing procedure. The controller must process data fairly, *e.g.*, ensure that decision making (automated or manual) does not involve undue bias. The controller must ensure processing transparency, *i.e.*, log all processing and make the audit trail accessible to data subjects and other authorities where applicable.

(ii) **Data minimization, purpose and storage limitation.** These principles limit data collection, distribution, as well as the duration for which personal data can be stored. For instance, distributing personal data for joint processing across multiple controllers must be done in a fashion that minimizes data exposure.

(iii) **Integrity, confidentiality, accuracy.** The controller must employ proper safeguards to ensure that personal data is not leaked/ accessed without authorization, as well as data integrity (protect against *e.g.*, loss, corruption, accidental deletion). Furthermore, the controller must take every reasonable step to ensure that the collected data is accurate and up to date.

(iv) **Accountability.** The controller is responsible for providing evidence of adhering to the above principles.

Meeting these principles requires specific capabilities, that can be categorized as set out below:

(i) **Consent management.** This includes obtaining proper consent from data subjects, communicating (in an accessible way) what data processing steps the consent applies to, and handling changes in consent, either on the side of the data controller (*e.g.*, re-requesting consent if data processing procedure changes), or the side of the data subject (*e.g.*, consent revocation).

(ii) **Data modification.** This includes updating/correcting records as well as *deletion*.

(iii) **Data protection.** This includes *(i)* securing data against data breaches, *e.g.*, via encryption, proper employee training, authentication mechanisms, *(ii)* collaborating with other data controllers/ processors to remediate data leaks/ personal data becoming public, *(iii)* preventing accidental data deletion and ensuring service availability.

(iv) **Logging.** All data processing must be documented, and the resulting log stored for archival purposes.

(v) **Attestation.** A data controller must be able to demonstrate (prove) facts about the data itself, as well as about data processing, and consent management. This includes demonstrating proper data subject consent, facilitating audits, justifying decision making upon request, etc.

(vi) **Notification.** A data controller must implement means to notify affected data subjects and relevant authorities of data breaches.

These capabilities come with qualifiers and conditions, *e.g.*, "reasonable effort", and "taking into account the state of the art". We will discuss interpretations of these qualifiers in Section 3 below.

Finally, personal data exists in one or more of the following *states*:

(i) **Default.** In its default state, personal data resides within a data controller's (or multiple joint controllers') jurisdiction (possibly via delegate data processors). The data is stored in its original form, *i.e.*, no processing such as pseudonymization or encryption has been applied to it.

(ii) **Pseudonymous.** Pseudonymous data can no longer be attributed to a specific data subject without the use of additional information, provided that such additional information is kept separately and is subject to technical and organizational measures to ensure that the personal data are not attributed to an identified or identifiable natural person (Article 4, 5).

(iii) **Encrypted.** Data protected via encryption; the GDPR does not explicitly state whether encrypted data is equivalent to pseudonymous data. In the context of data breaches, the GDPR makes a clear distinction between encrypted and default (unencrypted) personal data: if the leaked personal data was unencrypted, the data controller must inform all affected data subjects (Article 34, 2.), however, if the data was encrypted, this is not necessary (Article 34, 3. a.).

(iv) **Public.** This includes personal data that has been leaked or published. Personal data can be published by a data subject or a data controller (if given consent). If the data has been leaked, the data controller is responsible for remediating the issue by contacting other data controllers who have gained access to the data and request its deletion.

Furthermore, data can be **anonymous**. Data that has been anonymized does not qualify as personal data anymore. The principles of data protection should therefore not apply to anonymous data, namely data which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable. The GDPR does not explicitly define means by which data can be anonymized.

3 Interpretation

From a technical perspective, the above terminology raises a number of important questions.

Concepts such "deletion", "reasonable effort", or "pseudonymous" are by no means trivial to define from a computer science perspective.

However, such an analysis in necessary as it informs the development of concrete tools and procedures for GDPR-compliant data processing.

As a point of departure, we have select several concepts and definitions introduced in the previous section, propose preliminary interpretations from a computational standpoint, and establish additional terminology. We further expand on these in Section 4 where we discuss blockchain, distributed ledger, and privacy-enhancing technologies.

Reasonable (computational) effort. The obligations outlined in the GDPR are not absolute. For instance, to rectify leaked data, a controller must take "reasonable" steps. By default and unless stated otherwise, we interpret "reasonable" to maximally require no more than being *computationally feasible*. This mirrors the modern cryptographic approach to defining and reasoning about security. Virtually all cryptographic techniques and tools deployed in practice today assume that an attacker has bounded computational resources; they do not (necessarily) protect against attacks that are only hypothetically, but not realistically, possible by an attacker with superpolynomial or even infinite computational power. We adopt this view in this report, both in determining an upper bound to what constitutes reasonable expectations towards data controllers and processors, as well as what levels of security mechanisms are sufficient to meet the data protection capability. We further discuss security definitions in Section 4.1.

Stored data. Data is not necessarily kept within a single, centralized storage system. Rather, different approaches to distributed storage exist today, and must be taken into account, as these raise the question of how much control any single data controller has over a particular dataset. We consider two orthogonal dimensions of distributed storage: *replication* and *federation.*

In *replicated storage*, a dataset is replicated across multiple (possibly thousands or even millions) of machines. These machines may furthermore be *decentralized*, that is managed by different data controllers that have no control over each other or legally binding mutual business agreements and might even be anonymous. We note that such replicated decentralized storage of personal data, may be directly contrary to GDPR principles which say that there must be a single liable data controller. The Bitcoin network [Nak08] is an example of a replicated storage system.

Federated storage provides a way to encode a dataset into multiple data points such that a single data point reveals no information about the original data but the joint data points can be decoded back into the dataset. Encryption and secret-sharing (§ 4.1.3) are examples of such encodings. Once encoded, a dataset can be federated across network of machines which furthermore may be decentralized, that is managed by different, mutually independent data controllers. In this case, no single data controller (or delegate data processor) has access to the original dataset.

Note that replication and federation can be combined.

In order to address the fact that data storage can be distributed, we introduce the notion of data views.

Data view. A data view consists of a dataset and a set of controllers (this may be a single controller or a group of controllers). A data view defines *(i)* the state of the dataset (*e.g.*, anonymous, pseudonymous, default, etc.), given *only* the information the controllers in the group hold, and *(ii)* the control the controllers hold over the data, in particular the ability to delete or modify the dataset. This allows us to reason about distributed storage scenarios.

For instance, suppose a single controller (accidentally) leaks personal data to a replicated storage system such as the bitcoin network. The data view of that controller states that *(i)* the dataset is public (since, in principle, anyone now has access to it), and *(ii)* the controller does not have the ability to modify or delete the dataset. Note that the data view of the entire network (or a large enough fraction of it) supporting the replicated storage system *does* offer the ability to delete and modify the data.

On the other hand, suppose that a controller encrypts the dataset, and stores the data only in encrypted form. Additionally, the controller sends the ciphertext (but not the encryption key) to another controller. In this scenario, there are two relevant data views to consider, that of the sender and that of the recipient controller. The sender's data view states that the data is encrypted and can be both modified and deleted. The recipient's data view is that the data is anonymous (and therefore not personal) since the ciphertext alone does not reveal any information about the dataset. (We expand further on this discussion in our interpretation of the various data states.) Furthermore, the data view of the recipient provides no ability to modify or delete the data.

Deletion. We interpret deleted data as data that is not recoverable with reasonable computational effort. A straight-forward example is unreplicated data that has been erased from disk. However, more nuanced mechanisms for making data unrecoverable exist. For instance, encrypting a dataset and erasing the encryption key from disk is, arguably, equivalent to complete data erasure.[1].

Replicated, decentralized storage brings about another issue in regards to data deletion. Due to the lack of a central data controller, and the underlying system design (and purpose), systems such as Bitcoin (and most other blockchain systems) make it computationally and organizationally difficult to erase records. In fact, these

[1]This is a somewhat nuanced issue, since theoretically the encryption mechanism, while unbroken at the time on encryption might be broken in the future, in which case the ciphertext could be used to infer information about the underlying data.

systems are specifically designed to be immutable, *i.e.*,, for deletion or altering of records to be extremely hard. Roughly speaking, deleting a record would involve convincing every data controller in the system (possibly thousands of independent parties) to delete the record and then rebuild the system to re-record every record entered after the deleted record. Such an operation would be directly against the protocol governing the system, and it is not technically clear how it would be done without severely damaging the system. Furthermore, even if technically possible, the participants in the system would have a strong incentive not to do so as it would erode the trust in the system which relies on the systems immutability. Should personal data be stored (either maliciously or accidentally) in such a system, it is not realistic to require a single data controller to affect its erasure, since the controller's data view does not permit it. Article 17 of the GDPR, at least indirectly, accounts for this scenario (note italicized text):

> Where the controller has made the personal data public and is obliged pursuant to paragraph 1 to erase the personal data, the controller, *taking account of available technology and the cost of implementation, shall take reasonable steps*, including technical measures, to inform controllers which are processing the personal data that the data subject has requested the erasure by such controllers of any links to, or copy or replication of, those personal data.

Removing records from certain types of replicated storage goes beyond reasonable computational effort.

We further use the notion of data views to interpret the different states of data defined in the GDPR. As previously mentioned we consider the state of data as contingent upon a specific data view. Here, we focus on three data states: encrypted, anonymous, and pseudonymous.

Encrypted data. A dataset is encrypted, given the data view of a specific controller, if it has undergone encryption[2] and the controller holds both the encryption key and the ciphertext, *i.e.*, the controller is able to decrypt the data. Likewise, the (joint) data view of two controllers of a dataset, where one controller holds the encryption key and the other the ciphertext, classifies the data as encrypted (since the two controllers jointly hold all information necessary to decrypt). The data view of a controller only holding a ciphertext but not the corresponding encryption key or vice versa, defines the data as anonymous.

Anonymous data. We define anonymous data as data from which personal data cannot be derived with reasonable computational effort[3]. A trivial example is data without any linkage to personal data, *e.g.*, a public dataset that does not contain any personal information, or randomly generated data.

Deriving how much personal data, not even a single bit?

The data view of a controller holding a ciphertext, without the corresponding encryption key, is a more nuanced example. The ciphertext in isolation is anonymous data since it reveals no information (unless it is possible to break the encryption scheme) about the original dataset.

Pseudonymous data. We interpret pseudonymous data as data which only provides a marginal (and furthermore quantifiable) advantage at inferring any addi-

[2]We only consider encryption algorithms that are standardized (*e.g.*, AES [NIS01]) and/ or deemed secure by the academic community (*e.g.*, Shamir secret-sharing [Sha79]).

[3]More formally, we define anonymous data as data which does not provide an advantage at inferring any information about personal data over random guessing.

tional information about personal data. Differential privacy [DMNS06] is a strong candidate as a concrete mechanism for achieving pseudonymization.

According to a EU report it is classified as an anonymization technique. Is pseudo/anonymization a gliding scale of quantitative measures – intuitively not either-or but a question of how many bits are revealed– indexed by views and the degree of control ()collusion, collaboration) view holders have?

4 Relevant Technologies and Concepts

In this section we survey computational security and privacy concepts and technologies that pertain to the GDPR. In particular, we seek to expand on our interpretation from the previous section, and determine *(i)* how privacy-enhancing, blockchain, and distributed ledger technologies can help realize the necessary controller capabilities and ensure compliance with the GDPR's data processing principles (cf. § 2), and *(ii)* what other implications these technologies have on meeting GDPR compliance.

4.1 Cryptographic Security and Privacy

In the cryptographic literature security of a system (such as an encryption algorithm) is defined relative to the capabilities of an assumed adversary trying to break the security properties of the system. In particular, two settings distinguish themselves, namely the *information theoretic* and the *computational* setting.

In the information theoretic setting we assume an adversary that has *unbounded* computational resources, i.e., can compute infinitely fast and hold infinite amounts of data in its memory. Security in this setting can be divided in two categories, *perfect*- and *statistical* security. Perfect security, meaning that security cannot be broken, i.e., even with unlimited computational resources, an adversary cannot break security of the system in any way. Statistical security, on the other hand, means that the adversary may be able to break security of the system but only with some probability which is independent of the computational resources of the adversary. Specifically, the probability of breaking security is governed by a system parameter called the *statistical security* parameter, which can be set to make this probability arbitrarily small.

In the computational setting we assume an adversary that has bounded computational resources, *i.e.*,, systems secure in this setting are theoretically possible to break, but doing so requires some amount of computational resources. The amount of computational resources required is governed by a system parameter called the *computational security* parameter. This parameter is then tuned so that breaking security is assumed to require an enormous amount of resources, far exceeding what any real adversary can realistically be expected to possess. Note that statements about security in the computational setting are typically relative to *hardness assumptions* about certain computational problems; e.g., that factoring large numbers requires an in inordinate amount of computational resources to solve. We then show that breaking security of our systems is *at least* as hard as solving the computational problem. This means that if new techniques are developed for solving the underlying problem that disproves the hardness assumption this may effect the security of the cryptographic system. For this reason we prefer to rely on well studied hardness assumptions.

Thus, the fundamental difference between the two settings, is that in the information theoretic setting, we are essentially guaranteed that no amount of technical breakthroughs will invalidate the security of the system. Whereas, in the computational setting, breakthroughs in the methods used to solve the underlying computational problems may invalidate our hardness assumptions and consequently weaken the security of the system.

As discussed in Section 3 we generally consider protecting personal data using systems with security in the computational setting as a "reasonable" effort in the terms of GDPR. However, security in the information theoretic setting does raise some interesting questions in relation to GDPR requirements.

4.1.1 One-way functions (cryptographic hashing)

Cryptographic hash functions are a basic building block of many cryptographic systems, including most blockchain systems. A hash function is a deterministic function that takes any piece of data an produces a short representation of the data of some fixed size called a *hash*. A cryptographic hash function is a hash function where: *(i)* given a hash value it is computationally infeasible to produce some data with the same hash value faster brute force, i.e., we can not do better than try to guess the original data and test if the hashes match *(ii)* given some data and its hash value it is infeasible to produce a different piece of data resulting in the same hash value.

Property *(i)* above seems to suggest that a hash hides the original data and thus that a hash is anonymous. However, one should be careful with this view as it largely depends on whether a brute force attack against the hash is feasible. This may be the case if an attacker has additional knowledge about the hashed data. For example, if the hash represents a phone number an attacker may "simply" check the hash value against the hash value of all phone numbers in order to reverse the hash.

4.1.2 Trapdoor functions (encryption)

We have already covered encryption in relation to the GDPR in some detail above. Here we will just try to explain some of the terminology around the technology.

One typically distinguishes two distinct types of encryption, symmetric and asymmetric encrypt (also known as secret and public key encryption). A symmetric encryption algorithm is randomized algorithm that takes some message and a *key* parameter and produces a *ciphertext*. For the encryption algorithm to be secure we, roughly speaking, require that the ciphertext alone does not reveal any information about the message. This includes not being able detect whether two ciphertext produced using the same key are encryptions of the same message. Given a ciphertext and the key used to encrypt it should however be possible decrypt the ciphertext and completely recover the message.

An asymmetric encryption algorithm is similar except that encryption uses a special *encryption key* and decryption uses separate *decryption key*. The encryption key is then considered to be public while the decryption key is kept secret by a single party.

4.1.3 Secret sharing

Secret-sharing is a cryptographic technique for secure, federated data storage, formalized in seminal work by Shamir [Sha79].

Conceptually, a secret sharing algorithm encodes a secret data point v into multiple *shares* such that *(i)* given all [4] shares, it is possible to reconstruct the secret v, and *(ii)* an incomplete subset of shares reveals no information [5] about v.

Thus, if we split a secret into shares and distribute the shares across multiple servers, no single server stores the secret value, and all servers must collaborate to reconstruct it.

In relation to the GDPR, secret sharing provides *data protection* when storing personal data. Namely, secret sharing can be used to protect data within the

[4]More flexible schemes also exist which allow a secret to be reconstructed given only a subset of shares.

[5]Apart from leaking a bound on the bit length of v.

organization of a data controller, by secret-sharing the data across multiple (possibly isolated) servers. This way an attacker has to compromise all servers in order to gain access to the private data. Taking this idea further, the servers could be federated across *multiple* data controllers. Considering this setting raises interesting questions in the context of the GDPR.

In particular, in this case no single data controller (without collaborating with others) has access to the original data. In other words, the data view of any individual data controller (or an incomplete subset) holds no information about the original data, but the data view of the entire set of controllers hold full information on the shared data.

In such a setup secret-sharing could be seen as mean of *data minimization* as no individual data controller would have any information on original data. However, one could also argue that, the data held by any single data controller could be interpreted as effectively anonymous. Note, this follows from the same logic that leads us to conclude that a ciphertext is anonymous data in the view of a data controller that does not hold the corresponding key. Does this then mean that secret-shared personal data is outside of the scope of the GDPR? And could a data controller entrusted with personal data then secret-share this data with external parties without the consent of the data subjects? We revisit this discussion in Section 4.1.4.

While a fascinating technique in its own right, secret-sharing also forms the basis for an even more powerful cryptographic primitive, namely *secure multi-party computation* (4.1.4) which enables computing on secret-shared data.

both computational and information-theoretic secret sharing schemes exist

4.1.4 Secure Multi-Party Computation (MPC)

Secure Multi-Party Computation (MPC), just as secret-sharing, has its roots in theoretical work dating back over thirty years [Yao86]. Whereas secret-sharing securely federates data storage, MPC securely federates data processing. At a high level, MPC allows multiple servers (each potentially hosted by a different data controller) to jointly perform a computation, without ever learning the data on which the computation is done.

The applications of MPC are broad, and the past decade has seen several successful deployments []. For instance, MPC was used recently to compute the gender pay gap–the difference in earnings of male versus female employees–across a large portion of privately-held businesses in the City of Boston []. The businesses provided their employees' income records as inputs to the multi-party computation, and learned the total earning figures *without* revealing individual employee records or even sub-totals of individual businesses.

While there exists a number of different MPC schemes, one of the most practical and conceptually accessible approaches is based on secret-sharing (cf. for instance []). This approach provides cryptographic protocols for computing on secret-shared data. Once values have been secret-shared across multiple servers, the servers can jointly compute a secret-shared result of *any function* on original values, for instance, some statistic on the values, as in the examples above. The servers do so by manipulating their shares of the original values and, crucially, *without* ever reconstructing any of the original data or any intermediate data resulting from the processing. Thus, throughout the entire data processing, the underlying data remains secret-shared. At the end of the computation, each participating server stores a share of the result. The servers can use these shares to reconstruct the result of the data processing.

As data processed using MPC is always secret-shared, MPC offers capabilities in terms data protection similar to those discussed for secret-sharing. Only, MPC extends the protection offered by secret-sharing, from data while stored to include data while being processes. I.e., with MPC a data controller using secret-sharing across multiple servers to protect stored personal data, would never have to reconstruct the data in a single location, not even while processing. Also, consider two or more data controllers each holding separate data sets but requiring to process on the union of their data sets (such as the businesses holding salary and gender information of their employees in the example above). This processing could be done by the data controllers first secret-sharing their individual data sets and doing the processing using MPC. In such collaborative data processing MPC represents a very strong form of data minimization, as no personal data is ever exchanged between the data controllers during processesing.

In fact taking the view of data secret-shared between data controllers as anonymous, as discussed above, this means that the data controllers would only be processing anonymous data. Thus, MPC offers the desirable capability to anonymize and process data without loosing any utility of the data.

It is important to note that MPC protects the data that is being computed on; it does *not* protect any reconstructed result of the computation. For instance, if in the aforementioned gender-pay gap study the computed result was the name and salaries of the top five earning employees across all companies, this would obviously reveal personal information. In the MPC literature this is referred to as *output leakage*. While the above example is an extreme case, one can envision far more subtle cases of a computation result inadvertently allowing infer personal data. Output leakage must therefore be taken into account when evaluating MPC and its relation to the GDPR.

4.1.5 Zero-Knowledge Proofs

Zero-knowledge proofs (ZKP) (first proposed by Goldwasser *et al.* [GMR89]) is a type of protocol that allows one party (the *Prover*) to prove a *statement* to an other party (the *Verifier*) so that: *(i)* the verifier will be convinced by the proof if the statement is true, *(ii)* the prover can not fake a proof to prove a false statement, *(iii)* by seeing the proof the verifier learns nothing but the statement.

As an illustration, consider a customer at a bar wanting to prove to the bartender that he is over the legal drinking age. He may do this by presenting identity-papers stating his birth date to the bartender (such as a passport). This proof would arguably have properties *(i)* and *(ii)* described above. However, since the bartender would learn not only that the customer is over the drinking age, but the exact age of the customer, property *(iii)* does not hold. If instead the customer used a ZKP to prove the statement "my age is larger than the drinking age" the bartender would learn no other information from the proof.

An important application of ZKP's is to prove that some value was computed following some specific rules of correctness. From a privacy perspective this is interesting in a setting where a verifier needs to learn some function on the provers private data, but does not require the private data it self. I.e., a prover may use ZKP's to prove that some value y is the result of applying a given function f to private data x.

In cryptocurrencies this idea is currently being used to provide transaction privacy. In most cryptocurrencies (e.g., bitcoin) all transaction data must be public in order for the system to validate the correctness of each transaction. This specifically means the addresses of the sender and the receiver and the amounts of all transactions are made public. However, using ZKP's, we could encrypt the transaction data, and then simply prove that the encrypted transaction is valid (i.e., that it does not spend money the sender is not entitled to spend). This idea lies behind certain privacy preserving cryptocurrencies, most notably the ZCash system.

In context of the GDPR, ZKP's could also be used generally as a data disclosure minimization technique. Namely, in cases where a data controller is required to validate some claim about the personal data of a data subject, it may be sufficient for the data subject prove the claim using a ZKP rather than to present the personal data to the data controller.

4.2 Language- and systems-based security and privacy techniques

4.2.1 Data provenance tracking

4.2.2 Information flow analysis

4.2.3 Trusted execution environments

4.2.4 Differential privacy

5 Blockchain and distributed ledger technology

The term *blockchain* has its origin in the data structure used to store a log of transactions whose state is replicated in an open network of nodes employing a peer-to-peer gossip protocol. As such, it is a specific data structure for achieving a particular purpose, the atomic transfer of Bitcoin between anonymous parties without an appointed trusted organization controlling the process.

In this section we step back and propose general functional characteristics of *blockchain and distributed ledger systems*, including systems not built yet. It is analogous to describing the notion of automobile from a functional perspective rather than a particular engine technology. If Bitcoin is the Ford Model T of blockchain/DL systems, then the description should include it as an instance, of course, but also diesel-driven vans and self-driving electric-motor driven cars should be captured; they share "architectural" aspects, but are technologically very different.[6]

5.1 A blockchain/distributed ledger ontology

A blockchain system is a distributed system for managing digital (representations of) resources with certain characteristics and goals. In this paper we attempt to deconstruct blockchain systems into conceptual components so as to derive a canonical ontology and architectural components for discussing, designing and analyzing blockchain systems. We begin with a basic ontology. It is meant to be sufficiently stringent and connotative to facilitate a basic discourse without being too restrictive or formalized to discourage meaningful discussion.

5.1.1 Distributed systems

A *distributed system* is a *network* of *(computer) nodes*, each running some *program* that can receive and send *messages* to/from other nodes and from *client (computer)s* via *network connections*. Collectively, a distributed system offers a designated *service* to its clients via an *interface*, the kinds of messages it receives and sends.

A node can be thought of as a stateful *object*: it has an internal *state* that may change and cause messages to be sent as a consequence of receiving messages and of internal activities. The state is not shared with any other node; sharing needs to be explicitly modeled as a separate node or as a distributed (sub)system in its own right. A node is *reactive (event-driven)* if its state only changes as a consequence

[6]E.e., proof of work is a specific Bitcoin aspect, but does not occur in other blockchain/DL systems, just as spark plugs are not parts of electric motors.

of messages received; that is, it has no active threads of computation that change its state without being initiated by receiving a message. A node is *controlled* by an agent (see below), its *node operator.* A distributed system is *(organizationally) centralized*, if all its nodes are controlled by a single agent. It is *decentralized* if its nodes are controlled by a dynamic group of agents that are not, themselves, controlled by a single agent or colluding to act like a single agent. A node may suffer a *crash fault* (become non-responsive) or even a *Byzantine fault* (does not follow the agreed upon protocol). A distributed system may or may not tolerate crash faults and Byzantine fault, that is still provide its service to a high degree in the presence of crash or even Byzantine faults. A decentralized system may furthermore be exposed to *Sybil attacks*, where a group of agents takes over or adds a large number of network nodes to comprise the system's service by large-scale Byzantine faults.

Messages are usually classified into *queries*, *responses* and *commands.* A query results in a response that is sent by the query receiver to the query sender. Queries furthermore do not change the state of a node: the same query received multiple times yields the same response if no command is received in between. Commands generally update the state. The separation into query/response pairs and commands constitute the basic building blocks of CRUD-based and RESTful programming.

5.1.2 Resources, agents, contracts and events

Blockchain systems deal with (digital representations of) economic events and attendant information exchange. We find it useful to employ the Resources-Events-Agents (REA) accounting model, its basic terminology and its subsequent refinements, extensions (with information, location, independent perspective, structured contracts and additions in this paper) and formalizations in our discourse, which we collectively call the Resources-Agents-Contracts-Events (RACE) model.[7]

A *resource* is something physical or ideal that is economically scarce such as money/currencies, assets, physical resources (such as trucks, houses, dog food), property ownership, usage licenses, etc. A resource may be *unique* (the Guernica painting) or *fungible* (14 liters of milk, a dozen bagels). The characteristic property of a real-world resource is that it is *hard* to copy cheaply by nature (a truck) or by design (money).[8]

A piece of *information* is something physical or digital that is economically plentiful such as a message on a bulletin board, an invoice, a picture, etc. The characteristic property of information is that it is *easy* to copy cheaply.

An *agent* is a natural person or organization such as a company or division of such.

An *event* is a significant, atomic change of the *state of the world* occurring at some point in time; it includes occurrences of the following *actions* performed by agents:

- a *transfer* of some resource from one agent to another, e.g. Alice giving Bob 50 BTC;
- a *transformation* of some resource into another resource by an agent, e.g. Charlie producing a bicycle from all its parts or the National Bank of Denmark transforming part of its exclusive license to issue money into physical cash;
- a *communication* of some information from one agent to another, e.g. Bob sending Alice an invoice;

[7]The abbreviation of the alternative Resources-Events-Agents-Contracts is somewhat unfortunate.

[8]We may use the term *linear* resource for emphasis to minimize the confusion with uses of "resource" in IT such as in RESTful programming, where it corresponds to stored information.

- a *conclusion* by some agent of output information from some input information, e.g. Charlie concluding that he has to pay 100 BTC from two open invoices of 50 BTC each.
- an *observation* by some agent of some information; e.g. Bloomberg observing that the price of IBM stock hit USD 146.1 June 8th, 2018.

We can describe the state of the world in terms of statements of *ownership* and *knowledge* and characterize the *effect* actions have on the state of the world. If Alice owns 80 BTC and Bob 20 BTC and Alice transfers 50 BTC to Bob, Alice owns 30 BTC and Bob 70 BTC afterwards. Notice that the sum of what they own hasn't changed: the resources in the world before and after the event are the same. In particular, after sending the 50 BTC she doesn't own them any more. This is in contrast to communication: If Alice knows that IBM stock hit USD 146.1 June 8th, 2018 and communicates this information to Bob, *both* Alice and Bob know it afterwards. The information has been duplicated.

Not all actions are *valid* in every state of the world. For example, if Alice's *credit limit* is 0—she cannot borrow anything—she cannot transfer 50 BTC to Charlie after transferring them to Bob above; in particular, she cannot *double-spend* the 50 BTC. Similarly, if she doesn't know anything about IBM's stock price, she cannot inform Bob of IBM's stock price as if it were a fact. Though, if Bloomberg first observes IBM's stock price and then informs Alice of it, she can subsequently inform Bob of it.

A *contract* is a specification of which actions a group of agents is permitted, obliged and prohibited to perform at which time, in which order und under which circumstances. At its core, a contract is an agent-independent classifier of *collections* of events: Given a (possibly hypothetical) collection of events, it classifies it as either constituting a correct and complete execution or not. Being agent-independent means that the classification should be objective: it should not depend on a particular agent performing it.

Using the term contract is motivated by the conventional notion of (paper) contract, but its interpretation as an event collection classifier goes beyond this. Depending on the setting and compositional structure we may call a specification of such a classifier also a *business rule*, *policy*, *mechanism* or *protocol*.

For anonymization, authentication, authorization and other reasons we may use globally unique *identifiers* where *entities* such as resources, agents, events, contracts, nodes, are required etc. Their mapping to actual entities is dynamically managed by an *identity management system*, which may be distributed itself. A *blockchain/distributed ledger system* can be characterized by the following properties:

5.2 Blockchain/DL characteristics

We pose that blockchain/DL systems are characterized by the following properties.

Organizational and technical decentralization: It is a distributed system whose nodes are controlled by a dynamic group of independent principals (organizations), each of which ideally has the same access to and control of the blockchain system.

Tamper-proof shared storage: It maintains a ground truth of shared facts (consistency), which are furthermore immutable (tamper-proof).

Forge-proof digital resource management: It guarantees that (digital representations of) assets (resources) can be stored and transferred, but neither duplicated nor lost.

Specifically, organizational and technical decentralization involves an open ("permissionless") or closed ("permissioned") group of agents (parties, companies) that have equal data access, update and administrative control rights to a peer-to-peer distributed system; in particular, there is no privileged information aggregator and process owner (in particular no cloud hosting provider with privileged insight into the information streams of its users). Tamper-proof recording of events is a technical guarantee against tampering with recorded information, including deleting it. Recorded events establish a joint ground truth across all agents. It thus includes comprehensive reconciliation of inter-organizational information and establishes dependable provenance of information and resources across arbitrarily long supply chains. Forge-proof digital resource management means that resources such as fiat money, cryptocurrencies, property rights, licenses, (proxies for) arbitrary physical resources (trucks, components, plants, cement bags, cancer medicine,...) can be reliably stored and transferred digitally. The system has built-in guarantees against duplication (forging) of resources. This provides the equivalent of having a purely digital "original" certificate of anything of value and establishing who, uniquely, owns it at any given point in time.

Figuratively, a blockchain/DL system is the equivalent of a direct democracy involving many individuals spread over a large geographic area with no leaders or hierarchy, yet such that it collectively behaves reliably like a single organization that stores resources (who owns what), gives consistent (the same) answers no matter who amongst its (honest) members is asked, and works effectively even when there are failing, cheating or even actively attacking individuals forming gangs (and inventing identities of individuals).

There are various inherent distributed systems trade-offs between consistency, responsiveness, tolerance of network (communication) failures, degrees of resilience to failing, cheating members and to ballot-stuffing inside the organization, between privacy and performance. These preclude a single design of a blockchain/DL system being best at everything. There are many possible different designs; any one of them is technically complicated, and they all are different under the hood from centralized/cloud-hosted database systems, which most software engineers and developers are trained to deal with.

5.3 A brief review of blockchain and distributed ledger systems

We briefly describe the essential parts of some popular blockchain systems.

5.3.1 Bitcoin

The Ford Model T of blockchain systems is Bitcoin. It is an open network of self-authenticating replicated state machines collectively maintaining a list of transactions. Each transaction consists of a number of inputs and outputs, the latter with associated nonnegative amounts of its cryptocurrency, Bitcoin; a transaction is *valid* if each of its inputs refers to an output of a previous transaction that has not been spent (used as input by another transaction) yet such that an efficiently checkable predicate holds and the sum of Bitcoin at its inputs is equal to the sum of Bitcoin amounts the transaction associates with its outputs. The predicate in question is typically proof of knowledge of the private key corresponding to a public key serving as an anonymous identity ("address") that a Bitcoin transfer is made to. Bitcoin employs a decentralized lottery where a node needs to compute a nonce (a winning ticket) to prove itself as legitimate validator of a new block of approximately 1,000 transactions that it then sends to other nodes. Nodes are incentivized economically to extend the longest sequence of already validated blocks by receiving Bitcoin for block validation so that the nodes converge on a single chain of blocks.

5.3.2 Ethereum

Ethereum employs Bitcoin-like blockchain storage. Whereas in Bitcoin the behavior of an agent controlling an address is opaque, in Ethereum it is possible to tie an address to a reactive object with explicit, immutable code ("smart contract") that receives and sends messages and transfers of its own cryptocurrency, Ether, according to its semantics.

Both systems are intended to work in an open, anarchistic setting: any node and any address can participate; network nodes and users controlling an address authenticate themselves. No permission from any authority is required. Byzantine and Sybil attacks are countermanded by making it computationally extremely expensive to construct *any* valid chain of blocks and rewarding convergence on a single chain of blocks, which is then considered the "real" one.

5.3.3 Corda, Fabric and other distributed ledger systems

Blockchain subsequently received much attention as a decentralized platform for establishing (normative) consensus on a sequence of events amongst *authenticated* nodes and agents, especially in the financial sector where resources (money, bonds, etc) are essentially purely digital. Since payment for operating a node and recourse in case of cheating can be handled outside the blockchain system, such distributed ledger systems are typically designed employing classical distributed systems techniques, without cryptocurrencies for incentivizing correct behavior and without paying for computation. For example, Corda employs an architecture where authenticated nodes send information to each other privately (point-to-point) and only transfers of resources are validated by employing a small set of trusted validator nodes to establish a global total order on all requested resource transfers. The private messages are, a priori, only available to the sender and receiver; Corda has functionality for cooperatively collecting private messages and validated resource transfers to prove to each other or a third party whether a sequence of events abides by a designated protocol ("flow") such as a financial contract. If a node fails or refuses to participate in this phase, this may fail. Similar to Corda, Hyperledger Fabric employs an ordering service to globally and totally order functional (deterministic) update requests to the shared state of the world, which are then propagated and applied in that order by all peers in the system to update the state. In contrast to Corda, but similar to Bitcoin and Ethereum, Fabric nodes are, a priori, replicated state machines that can see all messages of amongst any nodes. (Partial privacy is regained by organizing the network hierarchically: a channel is a Fabric network in its own right, and each such channel participates as a single node in a higher-level network such that all intra-channel messages are kept secret from other channels.)

6 Open questions and problems

Here we identify key challenges related to the GDPR.

1. Illegal (private) content revocation/ removal in the context of public blockchains

7 Appendix

7.1 Technologies Supplementary Material

As a concrete illustration, consider the following scheme, referred to as *additive secret-sharing*. Suppose we want to split a value v into three shares. We generate two random values (of the same bit length as v) s_1, and s_2. We then compute a third value $s_3 = v \oplus s_1 \oplus s_2$. The three values, s_1, s_2, and s_3 represent our shares. Given all three shares, it is possible to reconstruct v, since $v = s_1 \oplus s_2 \oplus s_3$. Any subset smaller than three, on the other hand, does not reveal anything about v (TODO: elaborate, effectively one time pad).

A note on the relation between encryption and secret-sharing. Symmetric key encryption is a special case of secret-sharing: the encryption key represents one share, and the ciphertext the other. The key alone does not reveal anything about a cleartext value, and neither does the ciphertext. However, given both the key and the ciphertext, it is possible to decrypt and obtain the original value.

Secret-sharing generalizes this notion. Instead of two shares (key and ciphertext) there can be arbitrarily many shares.

7.2 Important GDPR Articles

Article 17, 2. Where the controller has made the personal data public and is obliged pursuant to paragraph 1 to erase the personal data, the controller, taking account of available technology and the cost of implementation, shall take reasonable steps, including technical measures, to inform controllers which are processing the personal data that the data subject has requested the erasure by such controllers of any links to, or copy or replication of, those personal data.

Article 5, 1. b. [...] collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes; **further processing for archiving purposes in the public interest**, scientific or historical research purposes or statistical purposes shall, in accordance with Article 89(1), not be considered to be incompatible with the initial purposes ('purpose limitation');

7.3 Misc

Concordium [] aims to integrate solutions for revoking illegal content into its blockchain system.

References

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[NIS01] Fips pub 197, advanced encryption standard (aes), 2001. U.S.Department of Commerce/National Institute of Standards and Technology.

[Par16] Parliament and Council of European Union. Regulation (eu) 2016/679 of the european parliament and of the council, 2016. `http://data.europa.eu/eli/reg/2016/679/oj`.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.