Decentralized Inverse Transparency With Blockchain

VALENTIN ZIEGLMEIER, Technical University of Munich, Germany
GABRIEL LOYOLA DAIQUI, Technical University of Munich, Germany
ALEXANDER PRETSCHNER, Technical University of Munich, Germany

Employee data can be used to facilitate work, but their misusage may pose risks for individuals. *Inverse transparency* therefore aims to track all usages of personal data, allowing individuals to monitor them to ensure accountability for potential misusage. This necessitates a trusted log to establish an agreed-upon and non-repudiable timeline of events. The unique properties of blockchain facilitate this by providing immutability and availability. For power asymmetric environments such as the workplace, permissionless blockchain is especially beneficial as no trusted third party is required. Yet, two issues remain: (1) In a decentralized environment, no arbiter can facilitate and attest to data exchanges. Simple peer-to-peer sharing of data, conversely, lacks the required non-repudiation. (2) With data governed by privacy legislation such as the GDPR, the core advantage of immutability becomes a liability. After a rightful request, an individual's personal data need to be rectified or deleted, which is impossible in an immutable blockchain.

To solve these issues, we present Kovacs, a decentralized data exchange and usage logging system for inverse transparency built on blockchain. Its new-usage protocol ensures non-repudiation, and therefore accountability, for inverse transparency. Its one-time pseudonym generation algorithm guarantees unlinkability and enables proof of ownership, which allows data subjects to exercise their legal rights regarding their personal data. With our implementation, we show the viability of our solution. The decentralized communication impacts performance and scalability, but exchange duration and storage size are still reasonable. More importantly, the provided information security meets high requirements. We conclude that Kovacs realizes decentralized inverse transparency through secure and GDPR-compliant use of permissionless blockchain.

CCS Concepts: • **Computer systems organization → Peer-to-peer architectures**; • **Security and privacy → Distributed systems security**; *Privacy-preserving protocols*; *Cryptography*.

Additional Key Words and Phrases: Decentralization, Non-repudiation, Accountability, Privacy, Anonymity

ACM Reference Format:
Valentin Zieglmeier, Gabriel Loyola Daiqui, and Alexander Pretschner. 2023. Decentralized Inverse Transparency With Blockchain. *Distrib. Ledger Technol.* , (April 2023), 30 pages. https://doi.org/10.1145/3592624

1 INTRODUCTION

Employee data collected in the workplace can be a valuable source for analyses and predictions. So-called *people analytics* tools utilize these data to help improve collaboration and facilitate work [84]. Yet, advanced analytics also increase the risk of misinterpretations or data misusage [85]. To protect employees from malicious usage of their data, the concept of *inverse transparency* [14] has been introduced to the workplace. That entails that all usages of personal data in people analytics are tracked, stored in a tamper-proof log, and made available to the data owners [102].

Authors' addresses: Valentin Zieglmeier, valentin.zieglmeier@tum.de; Gabriel Loyola Daiqui, gabriel.loyola-daiqui@tum.de; Alexander Pretschner, alexander.pretschner@tum.de, Technical University of Munich, TUM School of Computation, Information and Technology, Chair of Software and Systems Engineering, Boltzmannstr. 3, 85748 Garching, Germany.

2769-6472/2023/4-ART $15.00
https://doi.org/10.1145/3592624

This allows individuals more oversight and control in situations of asymmetric power, such as the workplace. For such a usage log to enable accountability, one needs to establish an agreed-upon and non-repudiable timeline of events [97] and guarantee its integrity [38, 72]. More importantly, no single party can be trusted with managing this log due to the inherent power asymmetry in the workplace. Otherwise, manipulation of the logs by, e.g., removing incriminating evidence would be possible, preventing accountability. To achieve this, Schaefer and Edman recently proposed utilizing blockchain for a secure usage log [72]. Blockchain can offer advantages in contexts with untrusted participants, especially if immutability of data is required [82, 96]. Consequently, multiple other secure logs based on blockchain were developed in recent years [e.g., 28, 76]. The technology has many advantages for these applications, as it is an effective way to guarantee immutability of stored entries (integrity) and functions even in unreliable distributed networks (availability).

Yet, in the context of data sharing, permissionless blockchain has two core limitations. First, with no trusted third party, no single arbiter can attest to the successful completion of data exchanges. Both sides of an exchange have an incentive to lie; the recipient of data may claim that they never received the data, while the sender may not send it but claim that they did. To guarantee integrity of the usage log, we therefore require non-repudiable data exchanges. Second, the unique properties of blockchain mean that the confidentiality of stored data cannot be guaranteed by default. Even when storing minimal information, some form of identifier is required to denote ownership or association to entries. Without necessitating any information leak, the blockchain then allows any network participant to trace entries based on their identifier [68]. At best, users can try to hide their association to blockchain entries by keeping their identifier secret and creating new addresses. Even then, network participants can deduce information about users simply by analyzing publicly available data [6]. If the identifier is leaked or known to a third party, though, all respective entries can be retroactively associated with them. This has been identified as a problem of blockchain-based secure logs [28, 72], where confidentiality can be an important property [3]. More critically, recent privacy legislation such as the General Data Protection Regulation (GDPR) [29] of the European Union and the California Consumer Privacy Act (CCPA) [15] requires those who store personal data to protect and, on request, even delete it. As data stored in blockchain can be identifiable, even with typical protection measures, they fall under the provisions of privacy legislation. Especially the right to erasure has been identified as a core issue of blockchain in this context [60].

Intuitively, it seems as if this means a fundamental conflict between the requirements of inverse transparency and privacy legislation on the one hand, and permissionless blockchain technology on the other hand. Therefore, the only solution would be not to utilize blockchain when providing inverse transparency. We argue that this conflict can be solved differently, though. We aim to combine the strengths of blockchain (such as decentralization and immutability) with the requirements of inverse transparency (accountability) and privacy legislation (confidentiality, deletability).

Contribution: Blockchain is uniquely positioned to solve many issues of inverse transparency in a decentralized environment. Yet, the goal of accountability for data usages requires a solution that guarantees non-repudiable data exchanges. Furthermore, secure usage logs based on blockchain are, by default, fundamentally at odds with privacy legislation such as the GDPR. The metadata recorded in blockchain are themselves personal data that need to be protected and, on request, rectified or deleted. To tackle these challenges and enable decentralized inverse transparency, we therefore contribute the data exchange and blockchain logging system Kovacs with its core components, the new-usage protocol and private pseudonym provisioning. Our contribution encompasses its concept and algorithms as well as a complete open-source implementation. The new-usage protocol enables secure and decentralized data exchange while ensuring non-repudiation, as required for accountability. The one-time pseudonym generation algorithm, meanwhile, guarantees two properties: Proof of ownership, as required for deletion requests, and unlinkability, to provide users anonymity against adversaries. Notably, Kovacs does not require any changes in the utilized blockchain software and can therefore even run on arbitrary public blockchains.

Extension: This paper is an extended version of our previously published short paper [99]. In it, we introduced the P^3 concept, encompassing a new-usage protocol and private pseudonym provisioning, and analyzed its security properties theoretically. We extend upon that in multiple, significant ways: (1) We present and implement Kovacs, a complete data exchange and usage log blockchain system that integrates P^3 to increase information security and GDPR compatibility. To that end, we add a refined use case (Sec. 2) motivating our work, expand on the requirements, adversarial model, and system concept (Sec. 3), add an implementation (Sec. 4), and expand our discussion of related works (Sec. 6) to cover non-repudiable data exchange. (2) We improve the new-usage protocol (Sec. 3.3.1) to simplify its implementation without compromising security. (3) In addition to our theoretical analyses (Secs. 5.1–5.2), we add an evaluation of the performance and scalability of the implemented Kovacs instance (Secs. 5.3–5.4).

2 BACKGROUND

We first go into more detail regarding the concept of *inverse transparency* and why we think its current realization is in need of decentralization. Then, we outline the legal difference between pseudonymity and anonymity, an important detail that we make use of later.

2.1 Inverse Transparency

Typically, when personal data are handled, their usage is covered by privacy policies or company agreements. These policies are hard to read and understand [55], calling into question whether individuals subjected to them truly understand their impact. Especially in the workplace, this can become problematic. While some usages of their data might be beneficial for employees, giving access to personal data poses the risk of profiling and misusage. The inherent power asymmetry and forced technology adoption exacerbate these risks [85, 100]. To give employees more oversight and control in this situation of asymmetric knowledge and power, the concept of inverse transparency [14] was introduced to the workplace. At its core, it is based on the principle that access to personal data should be visible (transparent) to data owners [14]. Gierlich-Joas et al. initially described how this idea can be applied abstractly as a digital leadership concept [30]. To realize inverse transparency technically, Zieglmeier and Pretschner propose to design people analytics software from the ground up to track the flow of data and create a data usage log [102]. Their *inverse transparency toolchain* is inherently centralized, though. It requires trust in multiple parties, such as the employer and the system administrators [see 98, 101].

Tracking all data usages is an important prerequisite for inverse transparency, but for true accountability we need to be able to guarantee completeness and correctness of the created usage log. Due to the inherent power asymmetry in the workplace, it is in our view not sufficient to consider the employer a trusted party that can safeguard the logs. Ideally, no trusted third party is required at all, as they might be interested to modify the log and, e.g., remove potentially incriminating evidence. Therefore, we consider it necessary to use a distributed, tamper-proof logging mechanism to guarantee accountability [see, e.g., 3, 72, 103].

In the following, we refer to the participants in a data sharing transaction as the *data owner* and the *data consumer*. The *data owner* “possesses the rights to the data” [63, p. 40]. The GDPR refers to them as the “data subject”. The *data consumer*, meanwhile, is the person or program that processes, and thereby “consumes”, personal data that identify one or more *data owners*. [63, p. 40]; [102]

2.2 Pseudonymity and Anonymity

Two concepts are important to understand when discussing the applicability and implications of the GDPR: pseudonymity and anonymity.

Pseudonymized data are personal data where identifiers (such as names) have been replaced by pseudonyms, and the association between pseudonyms and identifiers is stored separately from the data themselves. As the availability of this link allows re-identification, these data are not anonymous and fall under the provisions of privacy legislation [47]. Anonymized data on the other hand are data that have been modified as such to make it impossible to re-identify an individual from them [36, 47]. Importantly: While pseudonymous data are regarded personal data, anonymous data are not [47]. This means that to fulfill a user's legal right to erasure, we do not actually need to delete their personal data, as long as we can anonymize it by irreversibly deleting all existing links to the pseudonym [34, p. 153].

3 CONCEPT: DECENTRALIZED INVERSE TRANSPARENCY WITH BLOCKCHAIN

We present Kovacs, an inverse transparency log system that encompasses two core parts: non-repudiable data exchange and private pseudonym provisioning. Inverse transparency requires accountability for occurred data usages, which is why we develop a non-repudiable data exchange protocol. Evidences for the exchange are stored in a blockchain, ensuring log integrity. To protect confidentiality of the stored data while preserving proof of ownership, we present a pseudonym provisioning algorithm as the foundational differentiator of our concept.

First, we define our adversarial model and attacks that we consider. From our use case and the adversarial model, we derive requirements. Finally, we detail the Kovacs system, a complete solution for decentralized inverse transparency with blockchain.

3.1 Adversarial Model

A user u may be either in the set of *data owners* $O = \{o_1, o_2, ..., o_n\}$, in the set of *data consumers* $C = \{c_1, c_2, ..., c_m\}$, or both. The adversary α can be any user and assume any role. Whenever a consumer $c_i \in C$ accesses data of an owner $o_j \in O$, a usage $u_{ij}(c_i \longrightarrow o_j)$ is appended to the usage log U stored in the blockchain.

We assume that α has limited computational capacity, and can therefore never assume control over the blockchain network. Yet, within their means, they aim for maximum damage and therefore do not "play fair". To start with, α aims to attack the integrity of the log. As we use blockchain, preventing usages from being appended or retroactively modifying them is infeasible for α [96]. Therefore, they instead try to repudiate occurred data usages or claim fake ones. Furthermore and more critically for blockchain, though, α is motivated to attack the confidentiality of the stored data by gaining access to information that is not meant to be accessible by them.

Specifically, α tries to conduct the following attacks:

(a) *Repudiate* a data usage $u_{ij}(c_i \longrightarrow o_j)$.
(b) *Fabricate* an entry $u_{ij}(c_i \longrightarrow o_j)$ for a usage that did not occur.
(c) Derive from any entry $u_{ij}(c_i \longrightarrow o_j)$ with $\alpha \notin \{c_i, o_j\}$ the identity of c_i or o_j.
(d) Associate any two entries $u_{ij}(c_i \longrightarrow o_j), u_{ik}(c_i \longrightarrow o_k)$ with each other, thereby leaking their association with a single *data consumer* c_i.
(e) Associate any two entries $u_{ji}(c_j \longrightarrow o_i), u_{ki}(c_k \longrightarrow o_i)$ with each other, thereby leaking their association with a single *data owner* o_i.
(f) Leak the identity of c_i for a stored usage $u_{ij}(c_i \longrightarrow o_j)$ with $\alpha = o_j$, *after* c_i has legitimately exercised their right to erasure under the GDPR regarding u_{ij}.

3.2 Requirements

From our use case and adversarial model arise six main requirements: (1) No trusted third party is necessary. (2) The usage log needs to be non-repudiable and tamper-proof, preventing, e.g., data consumers from removing incriminating entries. (3) The usage log needs to be non-forgeable, preventing, e.g., data owners from creating valid, but fabricated entries. (4) *Data owners* can efficiently query for arbitrary log entries concerning their data and view their content. Importantly, they can prove the association of the data consumer to the logged usage (non-repudiation). (5) *Data consumers* can efficiently query for arbitrary log entries concerning their usages and verify their content. (6) No third party can derive identities or usage information from data in the blockchain.

In addition, the data stored in the usage log are governed by the GDPR for as long as they can be associated with the identities of users. From that follows an additional requirement, namely compliance with the GDPR rights [29, Ch. 3]. According to Pagallo et al. and Godyn et al., the main issue to solve for GDPR-compliant blockchain is the *right to erasure* [29, Art. 17]; [32, 60]. This is confirmed in the legal analysis of Tatar et al., who additionally note that a responsible controller may need to be identifiable to exercise that right [81]. We agree with their view, as we elaborate in the following. Most GDPR rights, such as the *right of access* [29, Art. 15], are not negatively affected when storing personal data in a blockchain. In fact, some may even be strengthened, as portability of and access to the data is enabled by design [37]. Technical challenges only arise from the inherent immutability of blockchain. This property implies that stored data cannot be deleted or altered, which affects the *right to erasure* and the *right to rectification* [29, Art. 16]. We can generalize both issues to a single technical requirement, as enabling the deletion of personal data indirectly enables rectification: by removing the incorrect entry and adding the rectified version. Furthermore, as we have discussed in Sec. 2.2, we can fulfill a user's legal right to erasure by anonymizing their personal data [34, p. 153]. Therefore, we arrive at requirement (7): The usage logs stored in the blockchain can be anonymized retroactively, making re-identification technically impossible.

3.3 The Kovacs System

Our concept for the Kovacs system consists of four parts: The new-usage protocol, the pseudonym generation algorithm, the block structure, and the deployment model. First, the new-usage protocol guides the decentralized, non-repudiable communication between data consumer and data owner when data are shared and the usage is logged. Second, the pseudonym generation algorithm enables the provisioning of private pseudonyms that guarantee unlinkability and proof of ownership. Third, the block structure describes how usage data are stored in the blockchain and how they are protected. Finally, the deployment model determines the required trust and computational resources.

3.3.1 New-Usage Protocol. Many properties we aim for hinge on the specific protocol that is followed when a data usage occurs. Concretely, that means a data consumer c_i is accessing a datum d of a data owner o_j and this usage being logged in the blockchain. This protocol is the first core step towards our goal. The most important challenge hereby is to guarantee non-repudiation of the occurred usage without a trusted third party. Therefore, we adapt the non-repudiation protocol designed by Markowitch and Roggeman [54] for our use case. In short: After the protocol is successfully completed, each party will possess proof of their interaction with the other party. Importantly, o_j can prove that c_i has received the datum d.

The protocol is described in Fig. 1. For this scenario, we assume that o_j returns the requested datum d directly. Alternatively, they could also return, e.g., the decryption key for a datum stored elsewhere. To begin with, o_j requests a one-time pseudonym from c_i. This is used to compose the block when the protocol completes. As the pseudonym is only relevant for c_i to be able to

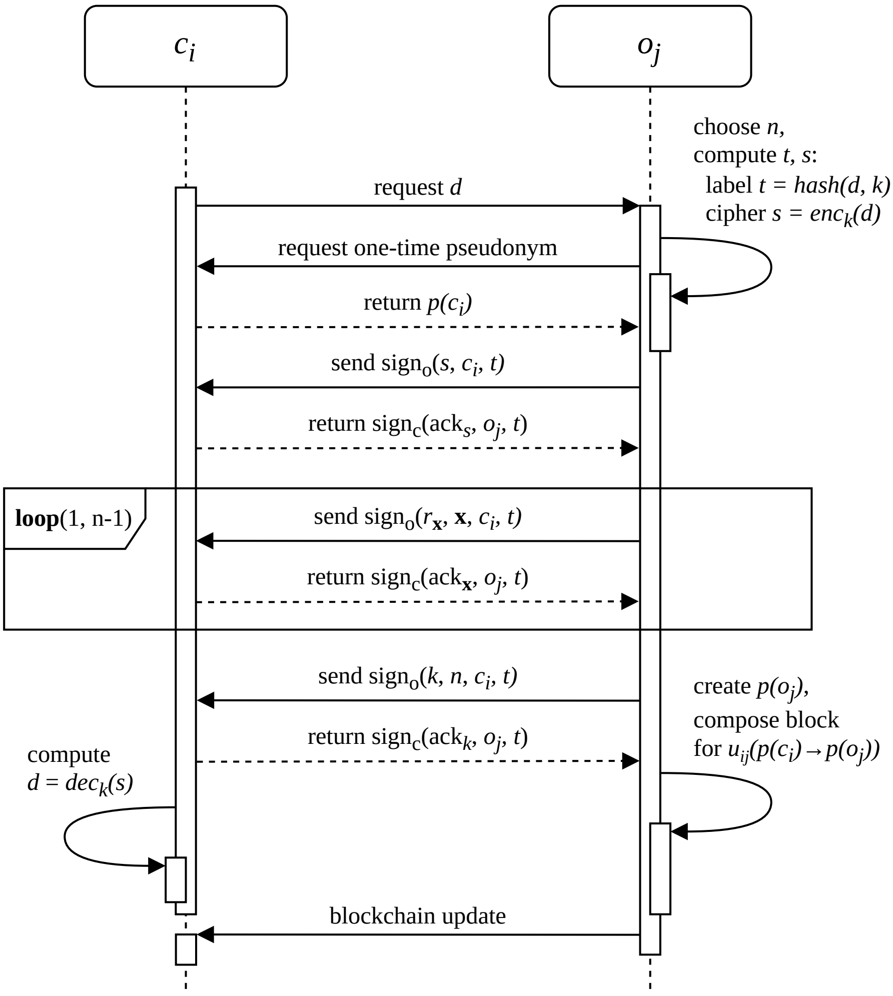

c_i
o_j
choose n,
compute t, s:
label $t = hash(d, k)$
cipher $s = enc_k(d)$
request d
request one-time pseudonym
return $p(c_i)$
send $\text{sign}_o(s, c_i, t)$
return $\text{sign}_c(\text{ack}_s, o_j, t)$
loop(1, n-1)
send $\text{sign}_o(r_\mathbf{x}, \mathbf{x}, c_i, t)$
return $\text{sign}_c(\text{ack}_\mathbf{x}, o_j, t)$
send $\text{sign}_o(k, n, c_i, t)$
return $\text{sign}_c(\text{ack}_k, o_j, t)$
create $p(o_j)$,
compose block
for $u_{ij}(p(c_i) \rightarrow p(o_j))$
compute
$d = dec_k(s)$
blockchain update

Fig. 1. The new-usage protocol, adapted from [43, 54, 75]. c_i requests access to the datum d from u_j. After receiving pseudonym $p(c_i)$, o_j sends the cipher s and the label t, which is a hash of datum and key serving as an identifier for the transaction. The receipt of s is acknowledged by c_i. Then, the iterative non-repudiation protocol is performed, with a random number of steps n unknown to c_i [see 54, pp. 5–6]. In each step x, o_j sends a random independent value r_x that must have the same size as the key k [43, p. 1610]. Only in the last message, o_j sends the actual decryption key k. Each message must be acknowledged by c_i quicker than the decryption could be performed. After receiving ack_k from c_i, o_j ends the exchange. This leads to c_i timing out, indicating that the exchange was completed. Thus, c_i can proceed to compute the datum d by decrypting s with k. Lastly, the block is composed and published. Both parties store their respective non-repudiation evidence on their own machine.

identify the block, it does not need to be verified. The number of steps n is chosen at random by o_j. For less critical data, it can be reduced [54, p. 7] to lower energy consumption or improve scalability. Then, o_j computes $n - 1$ random independent values r_x and a symmetric encryption key k. Importantly, the random values must be of equal size to the chosen key. Running $\text{enc}_k(d)$, they obtain the cipher s that they send to c_i. Now, in each step x, o_j sends a message with one of the random values r_x instead of the actual key k. Only in the n^{th} and final step, o_j sends k. [43, 54] After completion of the exchange, c_i can decrypt the cipher s to obtain d. Finally, o_j composes a block for $u_{ij}(p(c_i) \rightarrow p(o_j))$ and publishes it.

Following Markowitch and Roggeman [54], as c_i cannot predict n, and if the chosen decryption function takes long enough to compute, they will not be able to get any meaningful data when cheating [54, p. 5]. Only after having received the last message, they can decrypt the cipher [54, 75]. Now, each party holds non-repudiation evidence of the interaction. For o_j, this is { $\text{sign}_{\text{c}}(\text{ack}_s, o_j, t)$, $\text{sign}_{\text{c}}(\text{ack}_k, o_j, t)$ }, for c_i it is { $\text{sign}_{\text{o}}(s, c_i, t)$, $\text{sign}_{\text{o}}(k, c_i, t)$ } [43, p. 1610].

This protocol depends on the nodes being able to verify the authenticity of requests and, importantly, being protected against man-in-the-middle or eavesdropper attacks. Therefore, each request is signed by the sender. We do not aim to reinvent the wheel here, instead relying on the established HTTP over TLS standard [69]. This enables communication confidentiality and authenticity [42]. By utilizing the approach of a web of trust, as established in PGP [2], nodes are fully independent of any trusted third party to verify certificates. In that case, unknown certificates would be rejected and would need to be verified in-person. Alternatively, if sensible for the specific deployment, a certificate authority can be used to sign the individual certificates used by each node to sign and encrypt its requests. As these are often used in companies to enable the signing of internal emails or access to protected resources, no additional certification infrastructure is required in either case.

3.3.2 Pseudonym Generation Algorithm. The second core step towards our goal is the ability to generate unique one-time pseudonyms that guarantee unlinkability and enable proof of ownership without requiring a trusted third party. Unlinkability is required for the data we store to be able to qualify as anonymous data (see Sec. 2.2). Proof of ownership, on the other hand, enables the owners of the pseudonyms to exercise their rights as given by the GDPR [see 29, Art. 12.6].

Florian et al. [27] describe a pseudonym generation algorithm that serves as inspiration to our solution. Pseudonyms are guaranteed to be unlinkable to each other and to the real identity of the user. Furthermore, authenticity proofs enable proof of ownership, meaning that our requirements are met. Beyond those properties, their algorithm provides sybil-resistance, which is achieved by requiring additional computational steps for joining a network and creating new pseudonyms [27, p. 68–69]. In our case, the additional property of sybil-resistance is not required, as there is no inherent danger in a user creating multiple pseudonyms (see Sec. 3.3.1). We therefore omit these additional complexities and simplify our algorithm accordingly. By that, we reduce its computational complexity and energy consumption to a minimum.

Therefore, we define our pseudonym generation algorithm as follows: As part of the new-usage protocol (see Sec. 3.3.1), the user has created a new RSA private-public key pair with a key size of 4096 bits. As discussed above, the chosen encryption method can be updated if a higher level of security is appropriate. Now, to generate the one-time pseudonym, the collision-resistant and cryptographic hash function BLAKE2 [12], specifically BLAKE2s [12, p. 121], is applied to create a cryptographic message digest. Concretely, the user hashes the public key of their key pair, with the resulting irreversible and cryptographically safe digest representing their one-time pseudonym. BLAKE2s ensures a digest size of at most 32 bytes, which is important to minimize storage requirements.

The pseudonym generated with our algorithm then guarantees three important properties: First, the owner of the pseudonym, and only the owner, can prove the authenticity of the pseudonym (shown below). Second, the unique properties of the hash function guarantee uniqueness [8]. Third, users only need to manage a single key pair for each block, significantly reducing the complexity of the operation and increasing its speed.

To enable proof of ownership, we make use of the asymmetric nature of the RSA key pair. When a user wants to prove their ownership of a pseudonym, they sign a message with their private key and make available the corresponding public key. The signed message proves that they are in possession of the private key. The public key is then hashed by the recipient with the BLAKE2 hash function. If the result is the correct pseudonym, the ownership is proven. As BLAKE2 is collision-resistant, it is infeasible for an attacker to guess a different string that would result in the same pseudonym. Making matters even more secure, they would in addition need to crack the utilized RSA algorithm to be able to sign a message with a matching private key.

3.3.3 Block Structure. Next, we define the actual data stored in each block of the blockchain. As we have stressed above, our goal is to not require any changes to the underlying blockchain software, thereby allowing our solution to be used with existing blockchains.

As an example, consider a new entry logging the usage $u_{ij}(c_i \to o_j)$. Based on requirements (4) and (5), we need to store a one-time pseudonym for both the data consumer and data owner, to allow each party to efficiently query for entries concerning them. In addition, both parties need to be able to access the stored usage information, while preventing third parties from reading it, following (6). As shown in Fig. 2, each block therefore contains a payload with the data consumer's pseudonym $p(c_i)$, the data owner's pseudonym $p(o_j)$, and two copies of the usage request. This does not contain any identifiable information and only consists of the type of datum requested and a justification. To provide confidentiality, each copy is encrypted with an encryption function $enc()$, once with the *owner*'s and once with the *consumer*'s one-time public key. Importantly, this key is not shared publicly and instead stored securely with the private key. The chosen encryption should be regularly updated, but we require asymmetric (public-key) encryption [19]. As of this point, we recommend RSA [70] with a key size of 4096 bits [41, 46]. Notably, though, each individual can choose their preferred key size, and therefore security level, themselves.

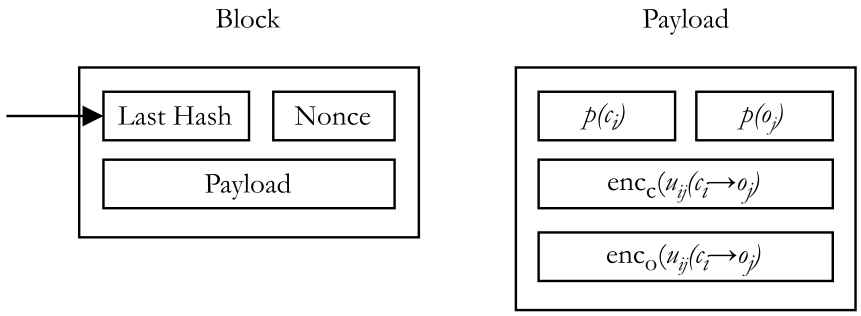

Block
Last Hash
Nonce
Payload
Payload
$p(c_i)$
$p(o_j)$
$\text{enc}_\text{C}(u_{ij}(c_i \to o_j))$
$\text{enc}_\text{O}(u_{ij}(c_i \to o_j))$

Fig. 2. The components of each block (adapted from [58]), for the exemplary usage $u_{ij}(c_i \to o_j)$. We replace the transaction history utilized in Bitcoin with a generic payload. The payload consists of the pseudonyms $p(c_i)$ and $p(o_j)$ of the data consumer and data owner, respectively. Furthermore, the logged usage is stored twice, once encrypted for the *consumer* and once for the *owner*.

Besides allowing both parties to read the usage log, storing it twice is also important because the block is created by o_j (see Sec. 3.3.1). c_i then needs to be able to verify the validity of the block. In case o_j manipulates the stored entry, c_i can utilize their copy of the usage and the non-repudiation evidence (see Sec. 3.3.1) to defend themselves against the faked evidence.

3.3.4 Deployment Model. Finally, as we have hinted at above, the chosen deployment highly influences the privacy and security guarantees that can be given. Our concept can be flexibly adapted and supports both centralized and peer-to-peer architectures. As we aim to not be dependent on any trusted third party, though, our deployment architecture is fully decentralized.

The central component of the deployment is the Kovacs system that handles data exchange, pseudonym provisioning, key management, and block creation. Each node in the peer-to-peer network runs its own Kovacs instance as well as a private non-repudiation store to store its RSA key pairs, pseudonyms, and non-repudiation evidences (see Secs. 3.3.1 and 3.3.2). The data exchange does not require an intermediary and utilizes the peer-to-peer network. As the usage log blockchain is permissionless, it is shared between all nodes that want to participate in the network. That means the architecture is fully decentralized, with each node communicating directly with other nodes both for data exchanges and blockchain updates (see Fig. 3).

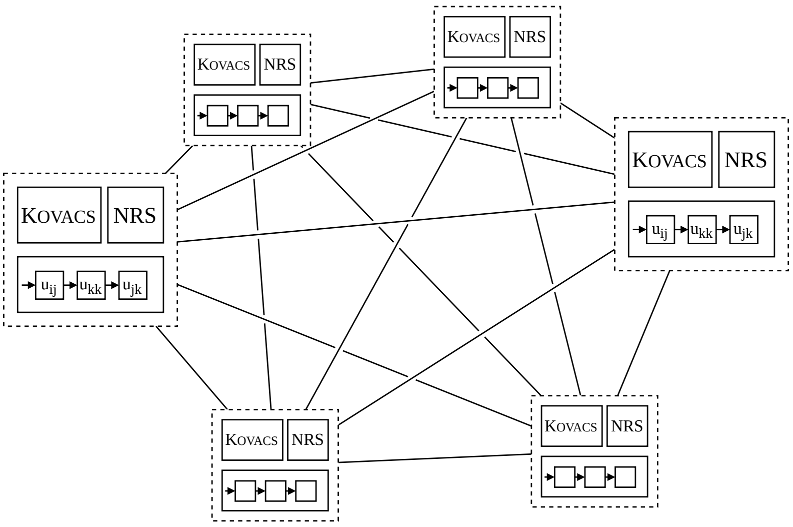


Fig. 3. Deployment in a fully decentralized peer-to-peer architecture. Each node runs its own Kovacs instance and private non-repudiation store NRS [see also 97], while the blockchain copy is shared within the network [see also 58].

The management of a large number of keys as required by our approach can itself become a privacy risk. In theory, the nodes would not necessarily need to store their key pairs and used pseudonyms at all to reduce their attack surface. Yet, these are important to enable requirements (4), (5), and (7). To be able to query for entries concerning their usages, users need to know which pseudonyms belong to them. In theory, they could iterate all blocks in the blockchain and simply try to decrypt their content, but this quickly becomes infeasible. Additionally, to exercise their GDPR-awarded rights, users need to prove their ownership of a pseudonym, requiring them to be in possession of the respective private key for the specific block (see Sec. 3.3.2). Otherwise, anyone could claim to be the owner of the encrypted data and, e.g., request its deletion. Still, each user can choose on their own how to manage their keys. If they prefer the maximum level of security, they are free to, e.g., delete new keys immediately after usage.

4 IMPLEMENTATION

Our implementation of Kovacs represents one possible manifestation of our concept. Some important facets have to be chosen for the concrete implementation and deployment scenario, namely the concrete blockchain, realizations of the underlying algorithms, as well as the integration with a complete network. Along those facets, we therefore describe how we implement Kovacs for our proof of concept and evaluation. The source code of the final tool is available on GitHub under the MIT license.[1]

4.1 Blockchain

First, the consensus mechanism and concrete blockchain for the implementation need to be chosen. We aim for blockchain-agnosticism, therefore the concrete choices are considered an implementation detail. We only outline our considerations and final choices below to ensure transparency and reproducibility regarding our evaluation.

4.1.1 Consensus Mechanism. The choice of consensus mechanism was made between the commonly known proof of work (PoW) [58], proof of stake (PoS) [40], and proof of authority (PoA) [93]. We arrive at the choice by process of elimination. We cannot use PoA, as it would violate our requirements by requiring a central authority that creates and signs blocks [7, 50]. If we choose PoS, we need to have a currency that can be staked. However, a usage log has no concept of "currency". Accordingly, we use PoW as the consensus algorithm for our implementation.

4.1.2 Blockchain Implementation. We do not depend on smart contracts, which made our selection of a concrete blockchain more flexible. Potential choices included Bitcoin [58] and Ethereum [92]. The widely used Hyperledger Fabric [5] was not considered, as it is based on PoA-based consensus. Between the two, Ethereum offers multiple advantages for our use case, namely a configurable hash difficulty and support for the creation of private chains [31]. Therefore, we use it for our implementation. As the client, we use the official implementation *Go Ethereum* (Geth).

Note that, at the time of implementation, Ethereum still used PoW consensus. Recently, Ethereum switched to PoS consensus [78]. When setting up a private network, it would therefore have to be configured to use PoW instead. Alternatively, a different PoW-based blockchain can be used.

4.2 Algorithms

With the blockchain in place, we turn to the implementation choices for the core algorithms of the Kovacs system. Broadly, the algorithms are, of course, just implementations of our concept. Yet, certain aspects may be realized in different ways depending on the application, which may impact, e.g., security or performance. Therefore, in the following, we deliberate the concrete implementation choices for the identity verification, time-asymmetric encryption, block composition, and fake chatter algorithms.

4.2.1 Identity Verification. In order to enable the traceability of data accesses, which is fundamental for inverse transparency to enable accountability [102], the data owner must know the data consumer's real identity. This enables attribution of a data usage to the responsible party. Thus, nodes need to be able to request and verify each other's identities.

We have noted in Sec. 3.3.1 that fully decentralized identity verification is possible by utilizing, e.g., a web of trust [2]. Yet, in our work with industry partners, we found that most companies rely on institutional identity providers (IdP) to realize single sign-on (SSO) for company-internal identity verification. As the use case of inverse transparency is specifically tailored to the company-internal context, we therefore show how to integrate an institutional IdP for identity verification. To minimize the risk of confidentiality attacks, we apply the concept of *self-sovereign identity* [see, e.g., 49, 64]. That means that each party is issued a *verifiable credential* [79] from the IdP, which it can present to exchange participants directly.

[1]https://github.com/tum-i4/kovacs

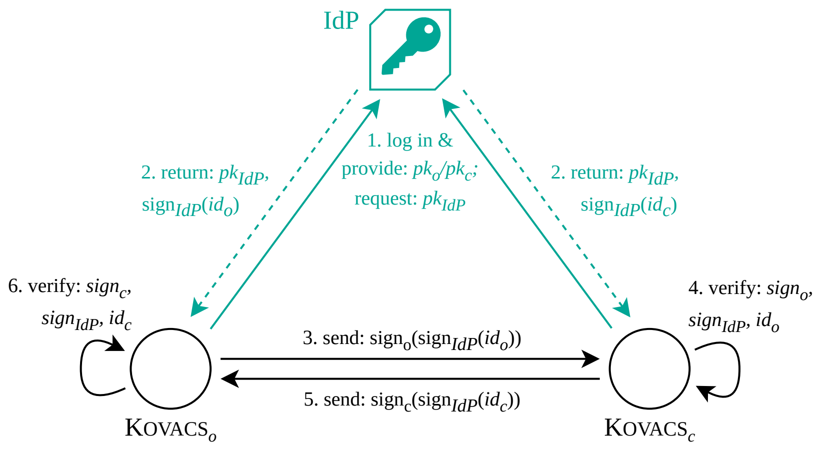

IdP
1. log in & provide: pk_o/pk_c; request: pk_{IdP}
2. return: pk_{IdP}, sign$_{IdP}$(id_o)
2. return: pk_{IdP}, sign$_{IdP}$(id_c)
3. send: sign$_o$(sign$_{IdP}$(id_o))
5. send: sign$_c$(sign$_{IdP}$(id_c))
4. verify: $sign_o$, $sign_{IdP}$, id_o
6. verify: $sign_c$, $sign_{IdP}$, id_c
KOVACS$_o$
KOVACS$_c$

Fig. 4. Identity verification steps for a data owner o and a data consumer c, utilizing an external IdP to issue verifiable credentials [following 57]; [see also 64, Chap. 3]. The KOVACS nodes of o and c log into the IdP once (1.) and send their public key to receive the corresponding verifiable credentials id_o and id_c as well as its public key pk_{IdP} (2.). During future data exchanges, o first provides their identity information to c (3.), which c verifies locally (4.). If the identity is confirmed, c also provides their identity information (5.), which is verified as well by o (6.).

Enabling self-sovereign identity requires only small adaptations to the IdP and can otherwise utilize existing authentication infrastructure. Following [57], each party's KOVACS node is issued a unique verifiable credential by the IdP on request. To trigger this, it logs in and sends a public key to be associated with their identity. The IdP then creates the verifiable credential containing of the user's IdP ID and their public key (the *claims*), and its own signature verifying the authenticity of the credential (the *proof*), and returns it to the requester [see also 79, Sec. 3.2]. Second, to prove their identity during the data exchange, the parties share their verifiable credential and sign them with the corresponding private key. Hereby, the data owner has to prove their identity first, which protects the data consumer's privacy during peer search. Each party can then compare the sender's signature to the public key on the credential and verify that the signature of the IdP on the credential is valid. If both signatures are valid, that confirms that the identity used by the other party in the data exchange is authentic. Following the principle of self-sovereign identity, this happens locally on each node, meaning the IdP cannot gain information on the data exchanges.

4.2.2 Time-Asymmetric Encryption. After the identity verification, the actual data exchange occurs. This exchange follows our new-usage protocol. Critically, though, the fairness of the protocol depends on the decryption time of the transferred datum being longer than the chosen timeout duration (see Sec. 3.3.1). We refer to encryption algorithms with this property as *time-asymmetric*. Such an algorithm would allow the data owner to simply encrypt the requested datum after they receive a data request. Due to the longer decryption time, the timeout duration for data exchanges could be set just above the expected encryption time. However, we were unable to find an encryption algorithm with this property in our research. Thus, we have to assume that the encryption time is equal to the decryption time.

To our knowledge, there are two alternative options how a longer decryption time can still be realized, which have implications on *when* the datum is encrypted and which *timeout duration* should be chosen for the data exchange. First, we can increase the timeout duration for the data exchange relative to the other steps of the protocol. This solves the problem of symmetric encryption and decryption time, but results in a longer exchange duration. The second variant is to (partially or completely) encrypt data *before* the exchange begins, which tackles both problems. Yet, assuming a reasonable number of options for which datum is actually requested, pre-encrypting all available data before the exchange is unrealistic. Exacerbating this issue, the encryption key needs to be different for every transaction, which requires all data to be re-encrypted after each request. Thus, a *full* pre-encryption is infeasible. Alternatively, we can move only *parts of* the encryption routine before the start of an exchange. These pre-computations must be independent of the specific requested datum to remove the need for re-encryption. As this is the optimal solution for our scenario, we use this approach and implement a two-step encryption process, outlined below.

The encryption is split up into a time-consuming cipher key generation procedure and a fast en- and decryption. Thus, the cipher keys can be precomputed, and only the encryption of the concrete datum has to be done at request time. To realize a time-consuming key generation, we create a random string and hash it with a random salt using a password hashing algorithm. These algorithms include key stretching functionality, which increases the time needed to calculate the hash [39]. The hash resulting from this operation is the cipher key. Importantly, the data owner only sends the random string and salt, requiring the data consumer to repeat the time-consuming key generation before being able to decrypt the datum. For the implementation of our proposed encryption algorithm, we use bcrypt [65] as the password hashing algorithm and AES-256 GCM [59] for the symmetric encryption. bcrypt and AES are widely adopted and their security guarantees have been verified on multiple occasions [see, e.g., 13, 77]. Our choice of AES GCM specifically is based on its guarantees regarding the integrity and confidentiality of data [22, p. 1].

4.2.3 Block Composition. The final step of the new-usage protocol is the block composition (see Sec. 3.3.1). As described in Sec. 3.3.3, we store the usage logs in transactions that are added in blocks to the blockchain. For simplicity, we do not rearchitect the blocks and just use regular exchange transactions to store the usage logs. This allows our system to even be used with an arbitrary public blockchain as a storage backend for increased security. Our choice requires us to mine two blocks for each logged usage: one to earn "currency" and a second to publish the usage log transaction. We require currency to be able to conduct a transaction that contains the usage log. Due to our unlinkability requirement, we create a new account for this purpose for each new usage log. Concretely, that means the following steps are conducted when a block is composed.

First, a temporary account is created and registered to receive mining rewards. Then, a block is mined to earn "currency". Now, a transaction is added that sends the generated reward from the temporary account to a hardcoded address. This transaction contains the newly created usage log. Therefore, it is not important who the receiver of the transaction is, since we only consider the metadata. The usage log transaction is still pending, meaning it is not yet stored in the blockchain. Thus, a second block is mined which contains the transaction. Finally, the temporary account is deleted.

4.2.4 Fake Chatter. Even though all communications are encrypted, an eavesdropping attacker could relatively easily trace newly published usage logs on the blockchain to participating nodes if only few exchanges take place. This attribution is possible because if exactly one block is published just after an exchange has ended, this block refers almost certainly to said exchange. Such an attribution would weaken the unlinkability of logs, though.

To solve this issue, we implement the optional *fake chatter* protocol. That means that nodes can complete fabricated "exchanges" that generate traffic, but do not add new blocks to the blockchain. Specifically, the data consumer creates a peer that completes an exchange as usual and additional peers that imitate data exchanges with random data owners. During such an exchange, the data consumer informs the data owner that this is not a real data exchange. Accordingly, no actual data are shared and no usage log is created. That also means that the blockchain is not written to, ensuring that it is not congested. Since the communication between the peers is encrypted, an attacker cannot distinguish between fake chatter and real exchanges. Hence, they would be unable to attribute a usage log to an exchange. If a sufficient number of concurrent exchanges take place, the protocol can be deactivated. Consequently, the peers' privacy is protected in both cases.

4.3 Integration

As the final aspect of our implementation, we need to consider the integration of the Kovacs system with a complete network. In the following, we therefore describe how the peer-to-peer network of nodes is created without a trusted third party and explain how we adapted the open-source *Revolori* SSO server to enable our identity verification algorithm.

4.3.1 Peer Discovery. We use *libp2p* as our peer-to-peer library since it offers encrypted communication and peer discovery for both structured and unstructured networks. For a structured network, libp2p requires a bootstrap node [74]. Thus, we do not use this approach since it would violate our goal of decentralization. As an alternative, libp2p also offers peer discovery in an unstructured network implemented with a flooding algorithm, which does not rely on any centralized service [73]. While flooding suffers from longer search times and likely worse scalability, we prioritize decentralization over performance. Accordingly, we implement an unstructured network.

4.3.2 Identity Verification Server. Finally, we adapt the existing *inverse transparency toolchain* [98, 101] to integrate the Kovacs system with a realistic identity verification server. In our realization of inverse transparency, we completely redesign the original fully centralized architecture to be as decentralized as reasonable. As the only centralized component, we use the SSO server *Revolori* as a stand-in for a company-internal IdP. We adapted it for our use case, adding functionality for the creation of verifiable credentials. Furthermore, we added an API endpoint that allows access to its public key, enabling local identity verification (see Sec. 4.2.1). These changes are non-invasive to the functionality of *Revolori*. Thereby, we show how Kovacs can be integrated with the existing inverse transparency toolchain. Our adaptations to *Revolori* are available on GitHub.[2]

4.4 System Model

Finally, we connect these components into a complete system. Fig. 5 is a model of the Kovacs system, outlining its components and their interactions. As the main component, the *Kovacs core* handles interaction within the node and with the peer-to-peer network (see Sec. 3.3.4). For data exchanges it first requests input data from the *non-repudiation input generation* component, which generates the parameters for the new-usage protocol (see Sec. 3.3.1). Furthermore, it requests one-time pseudonyms from the *P^3 pseudonym provisioning* component (see Sec. 3.3.2). The *query program*, finally, is utilized by the user to access the stored usage log. To do so, it loads all used pseudonyms and the decryption keys for the stored data from the *non-repudiation & key store*. Then, it queries the blockchain via the *blockchain node* to retrieve the requested usage log entries.

[2]https://github.com/tum-i4/inverse-transparency/tree/kovacs

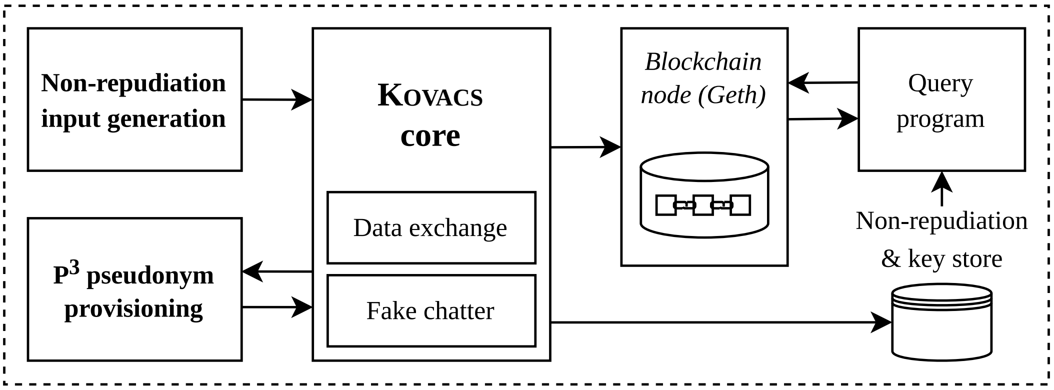

Non-repudiation input generation
Kovacs core
Data exchange
Fake chatter
Blockchain node (Geth)
Query program
Non-repudiation & key store
P³ pseudonym provisioning

Fig. 5. A Kovacs node. The arrows represent communication and data flow between the components. Bold names denote the three logic components: Kovacs core, non-repudiation input generation, and P^3 pseudonym provisioning. The name of the Geth node is italicized to denote it is a third party tool.

5 EVALUATION

To evaluate the Kovacs system, we first analyze the security of the concept and any implementations based on it. Second, we assess the GDPR compliance, specifically focusing on the data stored in the blockchain. Third, we benchmark the performance of our implementation for the most time critical operations. Finally, we measure its scalability for increasing numbers of log entries.

5.1 Security Analysis

We begin by analyzing the security of our system based on two core aspects: its robustness against attacks and the protocol confidentiality.

5.1.1 Robustness Against Attacks. We have described an adversarial model in Sec. 3.1, with adversary α trying to subvert the integrity and confidentiality of the stored logs. Specifically, they try to conduct six attacks. For each, we analyze the robustness of our approach against the attack and potential implications.

First, α tries to (a) repudiate a logged usage $u_{ij}(c_i \rightarrow o_j)$ after receiving the datum. To prevent this, we need to guarantee *non-repudiation of receipt* [97]. Our new-usage protocol is built on the Markowitch and Roggeman non-repudation protocol, inheriting its security properties. As shown by Markowitch and Roggeman [54] and later confirmed by Aldini and Gorrieri [4], the protocol can guarantee non-repudiation of receipt based on the chosen success parameter θ, which influences the choice of the number of rounds n. As θ is chosen by the data owner, they can configure the protocol to make it infeasible for α to repudiate a logged usage, preventing α to repudiate the usage. We discuss the performance implications of this in Sec. 5.3.1.

Second, α tries to (b) fabricate an entry $u_{ij}(c_i \rightarrow o_j)$ for a usage that did not occur, incriminating c_i. This requires them to successfully fabricate a corresponding *non-repudiation of origin* evidence. In our case, this is $\{\ \mathrm{sign_c}(\mathrm{ack}_s, o_j, t), \mathrm{sign_c}(\mathrm{ack}_k, o_j, t)\ \}$ (see Sec. 3.3.1). For this attack to be feasible, let us assume that α previously exchanged data with c_i, thereby receiving their verifiable credential. Given this, fabricating the non-repudiation of origin proof would still necessitate α to retroactively calculate the RSA private key of c_i matching the given verifiable credential. The private key is required to generate the proofs described above. The security of the utilized RSA has been shown previously [see, e.g., 53]. Furthermore, our selected key size of 4096 bits provides higher-than-usual security [41]. This makes it clearly infeasible for α to compute the RSA private key, even if they can utilize significant computing power. Should the chosen key size should become insufficient with rising computing power, though, it can be flexibly increased to harden the security.

Third, α tries to (c) derive from any usage $u_{ij}(c_i \rightarrow o_j)$ the identity of c_i or o_j. As the transaction pseudonyms for both c_i and o_j are created with the same algorithm, this attack depends on being able to reverse the employed pseudonym generation. The one-time cryptographic security of the utilized BLAKE2 algorithm has been shown [see, e.g., 12, 51], guaranteeing it to be irreversible. Furthermore, each transaction uses a new key pair and pseudonym, so their unlinkability is ensured even if the pseudonyms were reversible. This means that multiple usage logs cannot be linked. The metadata of an individual usage only contains keys that are not connected to the identity of the user, though. Therefore, α can gain no information on the identity of c_i or o_j from it.

Fourth, α tries to (d) associate any two usages $u_{ij}(c_i \rightarrow o_j)$, $u_{ik}(c_i \rightarrow o_k)$ with each other, revealing their association with a single *consumer* and, fifth, (e) associate any two usages $u_{ji}(c_j \rightarrow o_i)$, $u_{ki}(c_k \rightarrow o_i)$ with each other, thereby leaking their association with a single *owner*. We can discuss both attacks together, as they hinge on the same security mechanism. The cryptographic security of our algorithm is guaranteed by the cryptographic security of the two underlying algorithms RSA and BLAKE2, which has been shown for both [see 51, 53]. Therefore, this again depends on the ability of α to reverse the transaction pseudonym generation. As we have shown above, this can be considered infeasible.

Finally, α tries to (f) leak the identity of c_j for a stored usage $u_{jj}(p(c_j) \rightarrow p(o_j))$ with $\alpha = o_j$, after c_j has exercised their right to erasure. In the last section, we have detailed that the association of c_j to their pseudonym is known to o_j for blocks storing usages of data that o_j owns. We have no way to technically force o_j to delete this association when c_j exercises their right to erasure. Now, let us assume that o_j does not delete this data and wants to utilize their knowledge, e.g., by publishing the real identity of c_j and their one-time pseudonym $p(c_j)$. By itself, this proves nothing, as there is no technical relationship between the pseudonym and the identity of c_j (see Sec. 3.3.2). To actually prove the association of c_j to u_{jj}, o_j therefore needs to publish their non-repudiation evidence (see Sec. 3.3.1). This evidence, by design, contains their own identity (through their signature) as well [54, pp. 5–6]. This means that o_j would automatically also leak their own identity, making them legally liable. As this scenario is covered by legislation and can be prosecuted accordingly, we consider it a non-issue for most cases. Still, for the most secretive of environments, this might not be enough of a guarantee.

5.1.2 Protocol Confidentiality. For the highest-security deployments, our peer-to-peer architecture enables pseudonym generation and block creation without necessitating a trusted third party, mitigating most attack vectors on the integrity and confidentiality of data. Two potential attack vectors on the confidentiality of the exchanged information remain: The messages sent on block creation, and the block update.

Firstly, when creating a new block for a usage $u_{ij}(c_i \rightarrow o_j)$ following the protocol (see Sec. 3.3.1), communication between c_i and o_j has to occur. Even though HTTP over TLS is utilized, which prevents α from listening in as an eavesdropper [42], they may still deduce that there is some usage association between the nodes. To address this, we have implemented the *fake chatter* protocol (see Sec. 4.2.4). This works much the same way as the regular new-usage protocol, only that a special non-existent datum d_0 is requested. Then, both parties understand that this is just a fake request, and no actual block is added to the blockchain. This fake protocol can be run by nodes in randomized intervals, choosing arbitrary other nodes to request d_0 from. Thereby, we hide real requests in the noise of these fake requests.

What remains then is the block creation. Even if fake protocols are run regularly, α could simply watch for blockchain updates and derive from those which two nodes were responsible for the new block. This is possible because the block is added right after the protocol has concluded. The simplest mitigation of this is to add a random wait before the block is added. Then, plausible deniability is enabled, as there are a sufficient number of other potential users that might have been responsible. In fact, nodes might even wait for a certain number of block updates before publishing their update. Here, too, each node can decide itself the level of confidentiality it requires, and act accordingly. Furthermore, traditional blockchain algorithms already (indirectly) protect from α understanding the originator of a blockchain update. As the architecture is designed to be peer-to-peer, the mere fact that a node sends a block update does not give α any additional information about its creation. Nodes forward block updates to other nodes, so the specific node that sends α the update may also simply have forwarded it [58, 96].

5.1.3 Conclusion. The Kovacs system is robust against the most likely attacks as defined in our adversarial model. Furthermore, the new-usage protocol can easily be adapted to fulfill even the highest requirements towards information security. We conclude that usage logs created by Kovacs provide sufficient security even for highly adversarial deployments.

5.2 GDPR Compliance

In the following, we analyze the data stored in the blockchain, assessing the compliance of our solution with the GDPR.

5.2.1 Prerequisites. We have discussed above (see Sec. 2.2) that data can be considered pseudonymous and anonymous. When a possibility for re-identification exists, they count as pseudonymous and therefore fall under the provisions of the GDPR [47]. To comply with the GDPR, we need to enable data subjects to exercise their GDPR rights. We also found, in line with legal analyses, that the main GDPR right that is technically challenging when utilizing blockchain is the *right to erasure* [60, 81] (see Sec. 3.2). Accordingly, we analyze conformance with the GDPR's right to erasure in the following.

5.2.2 Analysis. Each block in our approach gets its own transaction pseudonym. We have shown in Sec. 3.3.2 that these pseudonyms are unlinkable to the individual's identity and to each other. For a usage $u_{ij}(p_x(c_i) \rightarrow p_y(o_j))$, only c_i and o_j know the association of the other party's single one-time pseudonym (p_x or p_y) to their real-world identity. In fact, c_i and o_j have to prove their identity to each other in the first step of the new-usage protocol (see Sec. 3.3.1). Due to the guaranteed unlinkability, this link is only given for the single pseudonym created for that block, i.e. o_j only knows the link $p_x(c_i) \leftrightarrow c_i$. This case is covered by the GDPR provisions as there is an identifiable data controller [c.f. 87]. The association is not stored in the blockchain, but only on the nodes of c_i and o_j. That means the request for deletion simply has to be forwarded to them. Considering an adversarial user, they might just not fulfill that request for deletion (see also Sec. 5.1.1). At first glance, this could imply that our protocol does not offer an advantage over modifying the blockchain and asking all nodes to delete their old copy. Importantly, though, we do not deal with *unknown* nodes. When a deletion request is raised, the data subject *knows* the identity of the offending user, and can *prove* it (see Sec. 3.3.1). That means, the individual can then be made responsible for deletion under the GDPR, and can be sued in case they do not follow through.

As soon as the association of the individual's identity to the pseudonym has been deleted, the data stored in the blockchain are anonymized. Then, they do not qualify as personal data anymore (see also [47] and Sec. 2.2), satisfying the right to erasure [29, Rec. 26].

5.2.3 Conclusion. As we could show, our solution satisfies the GDPR's right to erasure. Thereby, it solves the central challenge for blockchain arising from the GDPR [60]. Indirectly, this also enables the right to rectification, by deleting an entry and adding the rectified version. The other GDPR rights are either not applicable in our case or trivially enabled from a technical point of view (e.g., right of access; see Sec. 3.2). We conclude that the Kovacs system fulfills the technical requirements for GDPR compliance.

5.3 Performance

After the theoretical analysis, we assess the performance of our Kovacs implementation. All measurements were taken on an eight-core Ryzen 7 5800X with 16GB of DDR4 RAM running at 3600 MHz. The timeout duration of the non-repudiation protocol was set to three seconds. Thus, the decryption time needed to be at least three seconds, which was achieved with a bcrypt hash difficulty of 16. All nodes were run in Docker containers. Our adapted *Revolori* SSO was run on the same system, with nginx acting as a reverse proxy to allow the nodes to connect to it using the host system's network address. Geth was configured to be a full node, meaning that each node has a copy of the entire blockchain. Unless otherwise stated, the network was run with fifty peers.

We evaluate two concrete scenarios, namely the exchange duration, which differs for data owner and data consumer, as well as the log entry append time, which is only relevant for the data owner.

5.3.1 Exchange Duration. To analyze the exchange duration, we first need to define its beginning and end. From the perspective of a data consumer, the exchange begins with the start of their node and ends with them receiving and decrypting the requested datum. From the perspective of a data owner, the exchange begins when a data consumer connects to their node and ends after the new-usage protocol is completed and the usage log entry is appended to the blockchain.

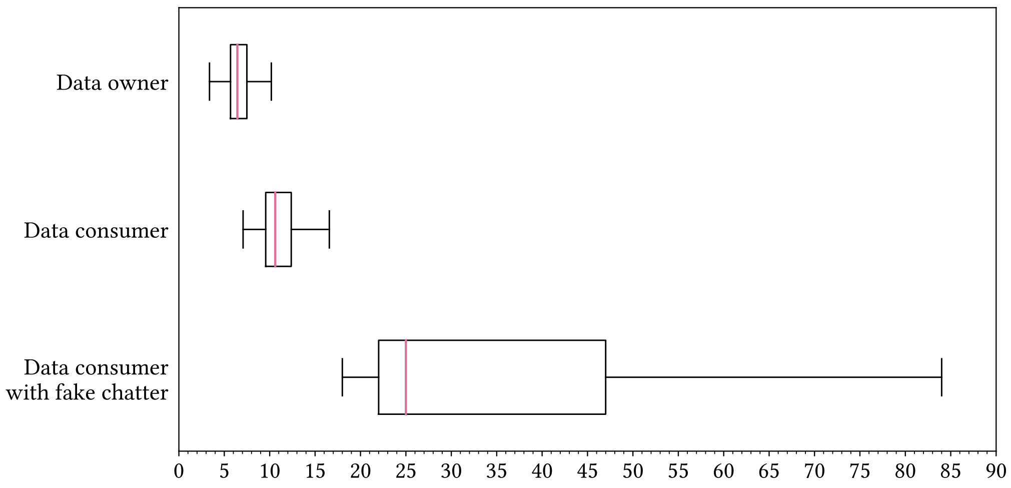


Fig. 6. Duration (in seconds) of an exchange from the perspective of the data owner, the data consumer, and the data consumer with fake chatter enabled (each measured over 1500 runs). The medians of approximately 6.4 seconds for the data owner, 10.6 seconds for the data consumer, and 25.0 seconds for the data consumer with fake chatter are marked with pink lines. The whiskers extend to $1.5 \times IQR$, meaning they show the 25th and 75th percentiles. Outliers beyond the whiskers are not shown.

Measured over 1500 runs, we find that the median exchange durations for data owner and consumer are 6.4 and 10.6 seconds, respectively (see Fig. 6), averaging 6.6 and 12.2 seconds. Thus, the exchange is approximately 5.6 seconds longer for the data consumer. This additional time for the data consumer is mainly spent waiting to time out and decrypting the received datum after the exchange has ended. To contextualize these results, we calculate the additional time effort for our approach compared to simple, repudiable peer-to-peer data sharing. For that, we measure the time for the additional required steps of our exchange, namely setup steps and peer search ($\approx$ 3.1 s, only data consumer), the new-usage protocol ($\approx$ 4.2 s for data owner, $\approx$ 9.0 s for consumer), and account and mining operations relating to the blockchain ($\approx$ 2.5 s, only data owner). The identity verification is negligible with $<$ 0.5 milliseconds. This shows that the largest proportion of time is spent on the new-usage protocol. Depending on network latency, this can rise further.

As described in Sec. 4.2.4, we implement the optional fake chatter protocol to increase the confidentiality of interactions. Naturally, it also reduces the performance of data exchanges, though. In our implementation, only the data consumer performs fake chatter, meaning the exchange duration is only affected for them. Figure 6 accordingly also shows the exchange duration from the perspective of the data consumer with fake chatter enabled. We aim to give an upper bound of the required time and therefore allowed fake chatter to trigger between 12 and up to 321 additional fake connections per exchange. This number is intentionally very high and can be lowered in practice. With this, we find that an exchange with fake chatter takes approximately 2.4× longer for the data consumer than without it, resulting in a median exchange duration of 25.0 seconds.

5.3.2 Appending a Log Entry. After the exchange is completed, the data owner appends the newly created usage log entry to the blockchain. Measured over 1500 runs, we find that the median blockchain append takes 2.3 seconds (see Fig. 7). In some cases, we measured durations of up to 11.9 seconds, but these are rare. We also find that the first blockchain append is consistently significantly slower than the median. We did not investigate further, as we attribute this slowdown to initiation overhead of the Geth client.

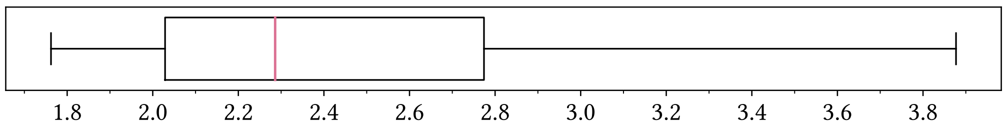


Fig. 7. Duration (in seconds) of a blockchain append, measured over 1500 runs. The median of approximately 2.3 s is marked with a pink line. The whiskers extend to $1.5 \times IQR$, meaning they show the 25th and 75th percentiles. Outliers beyond the whiskers are not shown.

To understand which individual operations take up most time, we can roughly split the blockchain append into three steps: (1) Mining, to earn currency and store the transaction containing the usage log, (2) creating and unlocking an account to create a transaction, and (3) creating the transaction that will store the usage log. Since we use a PoW consensus algorithm, one could expect mining to be the main reason for the slow blockchain append. However, our benchmarks reveal that mining accounts for less than half of the time spent (on average 1.01 s). The remaining time is spent creating and unlocking the account (on average 1.47 s), which is performed by Geth. Finally, the time spent creating a transaction is negligible at about 2 milliseconds.

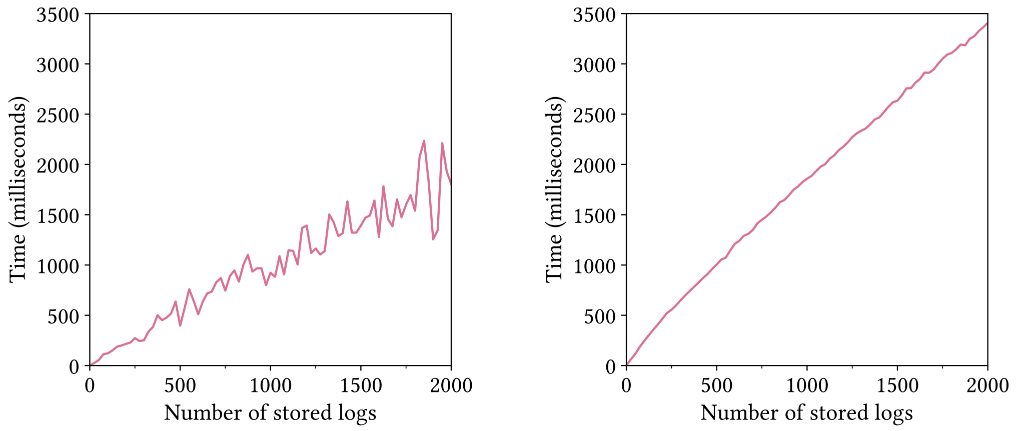
5.3.3 Conclusion. Both the exchange duration and blockchain append times of Kovacs are notably slow, being measured in seconds. While this is expected, as we focus on increased security, we need to note that this is a clear trade-off. For scenarios with lower security requirements, the significantly increased exchange duration specifically can be disqualifying. Yet, given our primary goals of decentralization and high security, we consider the time taken to still be reasonable. Furthermore, this time does not increase even if data from multiple data owners need to be requested, as requests can be parallelized. That means that these operations do not impact scalability.

5.4 Scalability

Finally, we evaluate the scalability of our Kovacs implementation. As above, all measurements were taken on an eight-core Ryzen 7 5800X with 16GB of DDR4 RAM running at 3600 MHz, and all nodes were run in Docker containers. Concretely, we evaluate the log retrieval time as well as the storage requirements for increasing numbers of log entries.

5.4.1 Retrieving Usage Logs. One of the advantages of Kovacs compared to other blockchain-based secure usage logs is that searching through log entries does not require decrypting each block. Therefore, one would expect good scaling behavior when retrieving usage logs. To evaluate this, we measured both retrieval of a single, random usage log, and retrieval of all usage logs. We ran each step 50 times and averaged the results.

Overall, we find that Kovacs returns the result in hundreds of milliseconds in both cases, with, e.g., retrieval of one log out of 1000 stored logs taking on average 0.97 seconds (see Fig. 8a), with a median of 0.92 seconds. Retrieving all logs in this case, as a comparison, takes on average 1.88 seconds (see Fig. 8b), with a median of 1.86 seconds. The high variance that can be observed for the single log retrieval task (see Fig. 8a) can be explained with the random log selection. The newer the selected log is, the more blocks may need to be searched before finding a match, which directly influences the retrieval time. More interesting than the absolute number is that the retrieval time in



(a) Time to retrieve a random usage log. (b) Time to retrieve all usage logs.

Fig. 8. Average retrieval time (in milliseconds) of a specific but random usage log entry (left) and all usage log entries (right), measured for log sizes of 25 through 2000 stored logs, with a step size of 25, and with 50 repetitions for each step.

Kovacs increases only linearly with the number of stored logs. We can estimate the scaling behavior with linear regression. For retrieval of a single, random entry, we obtain a slope of approximately $0.91 \times x$ ms with $R^2 > 0.92$. For retrieval of all entries, we obtain $\approx 1.66 \times x$ ms with $R^2 > 0.99$.

Compared to a centralized database that is stored on a remote machine, we expect the overall retrieval time to be relatively competitive in real-world scenarios, as Kovacs does not need to retrieve data over the network. Instead, it can directly query the local blockchain copy. This, combined with its efficiently queryable blockchain, positively impacts the scalability of Kovacs.

5.4.2 Storage Requirements. Finally, we consider the storage requirements of our Kovacs implementation. This is more relevant in our case than with traditional centralized data stores, as each node stores a copy of the full blockchain. Therefore, we measured the total log size and calculated the average size per log as well.

We find that the total log size measures in megabytes and, more importantly, rises linearly with increasing numbers of stored logs (see Fig. 9). For example, for 1000 logs, the total size is 11.82 MB. With linear regression, assuming a basis storage requirement of 16 KB, we obtain a slope of approximately $12.22 \times x$ KB with $R^2 > 0.93$. On average, an individual log entry takes up at most 20 KB, with the size approximating 12 KB per entry in a log with 2000 entries in total. To contextualize these results, we measured the storage requirements for our usage logs when stored in a minimal SQLite database. There, we find that storing the same log entry needs about 4.5 KB of storage space, which results in about 4.51 MB of storage for 1000 logs. The difference is expected, as blockchain has an increased storage overhead by design. I.e., every block includes the hash of the previous block, transaction details, and the nonce. Importantly, though, the storage size of Kovacs only linearly increases with the number of stored logs, positively impacting the scalability.

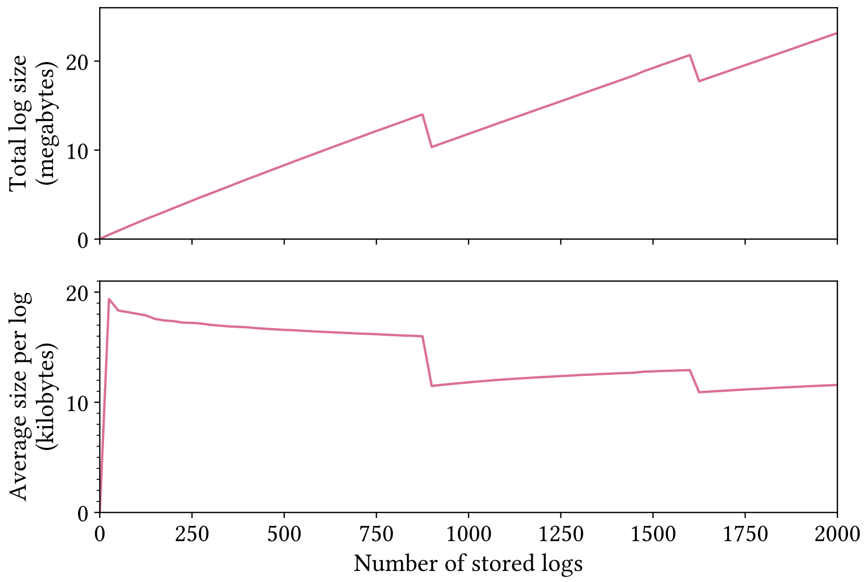


Fig. 9. The storage requirements of the blockchain log. The top graph shows the total log size, which increases linearly. The bottom graph shows the average size per log, which decreases for larger number of logs stored in the blockchain. For increasing log sizes, a single entry converges to a size of 12 KB.

Interestingly, the blockchain size shrinks periodically (see Fig. 9), which we attribute to Geth pruning its state [80]. Furthermore, mining the first block increases the storage space comparatively more, which we hints at initialization overhead of the blockchain and Geth client.

5.4.3 Conclusion. Kovacs shows good scaling behavior. The log retrieval time rises only linearly for increasing numbers of log entries, both when retrieving a specific entry and all logs entries. In absolute terms, it is relatively low, especially considering that no additional network requests are necessary. Similarly, the storage requirements are, while larger than a minimal database, very low considering the additional metadata that need to be stored. This means that storing a full blockchain copy is unproblematic for individual nodes. We conclude that Kovacs is sufficiently scalable for real-world usage and usable even in environments with hundreds of participants.

6 RELATED WORK

We solve two challenges of decentralized inverse transparency: non-repudiable data exchange and GDPR-compliant use of blockchain. In the following, we discuss alternative solution approaches.

6.1 Non-Repudiable Data Exchange

In decentralized scenarios, achieving non-repudiation in data exchange becomes a challenge, as we have noted. Various alternative proposals to solve this exist, of which we discuss notable examples in the following. The overview provided by Wang et al. [89, Sec. 1.2] and the work by Kremer et al. [43] serve as partial foundations and help confirm our research.

6.1.1 Protocols Requiring a Trusted Third Party. Many data exchange protocols exist that require a trusted third party. This includes traditional protocols based on arbitrated exchange [e.g., 1, 17], timing-based protocols [e.g., 66, 95], and optimistic fair exchange protocols [e.g., 10, 44]. This category of protocols does, by design, not fit our requirement of decentralization. Therefore, they are on principle not relevant in our scenario. Contrary to these approaches, we solve the issue of non-repudiable data exchange without a trusted third party.

6.1.2 Smart Contracts as a Trusted Third Party. To benefit from the existence of a trusted third party without necessitating the same trust, some authors propose utilizing a smart contract to fulfill that role [e.g., 23, 25]. Compared to traditional approaches, this has the advantage that, at least theoretically, the behavior of the smart contract can be verified before it is being used. By inspecting the smart contract code, parties can decide if they find it trustworthy. The concrete data sharing schemes can then be modeled similarly to protocols with a regular trusted third party. For example, they can implement arbitrated exchange [e.g., 23] or optimistic fair exchange [e.g., 25].

Utilizing a smart contract may alleviate the issues with a trusted third party to some extent. Compared to our solution, this approach has two main disadvantages, though. First, it depends on the support of smart contracts by the blockchain. Our approach, meanwhile, is compatible with any blockchain, making it more flexible. Second, while the behavior of a smart contract can theoretically be vetted, various vulnerabilities and security issues with existing smart contracts [see, e.g., 16] show that this is a difficult problem. In our approach, users are not expected to perform a security audit or have the technical knowledge to be able to judge the trustworthiness of a smart contract.

6.1.3 Specialized Hardware as a Trusted Third Party. Alternative approaches have been proposed based on trusted hardware [e.g., 48, 94]. Similarly to the examples with smart contracts, the specialized hardware serves as a substitute trusted third party. For example, Intel software guard extensions (SGX) [e.g., 94] or smart card [e.g., 48] can be used. Again, that means that traditional data exchange protocols can be utilized, with the trusted hardware serving as the arbiter.

Compared to using smart contracts, these approaches reduce the security and increase the required trust, though. The utilized hardware is only considered trusted because its manufacturer is assumed to be trusted. While with smart contracts, the actual code that is executed could—at least in theory—be vetted, trusted hardware does not even allow for this. Additionally, this solution introduces a new problem in that only participants with the required hardware can participate. As noted above, our approach instead does not require trust in any party. Furthermore, it does not depend on specialized hardware.

6.1.4 Data Delivery via Blockchain. To not depend on any trusted entity, some authors propose to transmit the data simply by appending it to a public blockchain [e.g., 94]. Immediately, the first issue with this approach becomes apparent: Potentially identifiable data are stored immutably in a blockchain. In Sec. 6.2, we discuss why encryption is not sufficient in this case to ensure compliance with GDPR and similar privacy legislation. Contrary to that, our approach utilizes one-time pseudonyms that guarantee unlinkability as required for anonymization.

Considering non-repudation, this approach at least ensures non-repudiation of origin, as the sending of the data is tracked in the blockchain. Regarding non-repudiation of receipt, though, issues arise. For example, Zhang et al. simply claim that, due to their blockchain's inherently public nature, the receipt of data is simply "undeniable" [94, p. 61]. This mirrors the claim of Paulin and Welzer, who for their protocol claim that, as long as data are freely downloadable, their receipt can be considered as successful [61, p. 211]. We fundamentally disagree with this notion and consider it insufficient for true non-repudiation of receipt. As the simplest example, a recipient can always claim they disconnected from the network, even after successfully receiving the data. Accordingly, non-repudiation cannot be guaranteed in this approach, rendering it insufficient for our problem.

6.1.5 Staged Data Delivery via Blockchain. To generate some evidence of receipt when sharing data publicly, e.g. via blockchain, staged protocols have been proposed [e.g., 61, 89]. Here, the shared datum is split up into parts. To simplify, we can generalize the solution as splitting the data up into two halves, as increasing the number of parts arbitrarily does not change the provided guarantees. The data owner shares the first half of the encrypted data directly with the data consumer. Then, the consumer appends an acknowledgment of receipt to the blockchain. Only then does the owner share the second half of the data, this time via the blockchain network, with the consumer. [89] This improves upon full data delivery via blockchain by solving the issue of GDPR compliance. An unreadable part of the data is also not personally identifiable and can be stored in a blockchain.

More critically, though, the receipt of the last part of the data is not acknowledged in this approach either, as in those discussed above. Independently of how many parts of the data the recipient has acknowledged having received, if they cannot decipher the datum without receiving the last part, they can always repudiate the receipt of the full datum. This is the fundamental issue with non-repudiation of receipt and splitting up the data does thereby not improve upon simply sending the full datum in one transaction. Again, that means that non-repudiation cannot be guaranteed in this approach, meaning it does not represent a sufficient solution either.

6.2 GDPR-Compliant Use of Blockchain

Various proposals for how to solve the conflict between the GDPR requirements and blockchain immutability exist. In the following, we describe important works and discuss how they differ from our approach. The overviews by Pagallo et al. [60] and Politou et al. [62] serve as a foundation. A recent systematic literature review [33] confirms their completeness.

6.2.1 Hashing Out. A trivial solution would be to not store personal data in a blockchain at all. *Hashing out* specifically refers to the practice of saving only the hash of the data in the blockchain and the data themselves off-chain [60]. This approach is one of the most commonly used ideas to ensure GDPR compliance in blockchain solutions [see, e.g., 72, 88, 91]. This works because the on-chain hash does not contain any private or personal data and the off-chain data can be deleted or modified to comply with a data subject's request.

There are two major downsides, though. Since the data themselves are not stored in the blockchain, this solution is not truly decentralized and requires trust in the authority managing the data [35]. Furthermore, using this approach one can only be sure of the *existence* of entries, not of their *content.* Arbitrary entries or even the complete log could be purged, with only the hashes remaining. To prevent malicious deletion, the party managing the log can be held accountable in case entries are missing. This can provide some protection, but there remain options for *plausible deniability*; e.g. blaming a corrupted hard disk for data loss. That means this approach is effective only as long as the log is not tampered with, but cannot *guarantee* accountability or non-repudiation.

We allow users to benefit from accountability guarantees even for highly capable adversaries—without corruptible intermediaries or plausible deniability—as we require no trusted third party. Meanwhile, we still provide them the same level of confidentiality.

6.2.2 Key Destruction. If personal data *are* to be stored in a blockchain, the next best idea seems to be to encrypt all stored data and delete the decryption key if the data are to be "deleted" [60].

While easy to implement, this approach is flawed. Encryption itself only guarantees pseudonymity of data [29, 45], therefore the data protection requirements still apply [45]. More problematically, though, if the full content of the block is encrypted, querying history becomes all but impossible, which is a requirement in secure logs for efficiently reading past entries. The affected parties would only be able to retrieve their entries with high computational overhead, by going through every block and trying to decrypt it.

Our system in contrast enables efficient querying of entries based on the one-time pseudonyms. The pseudonym provisioning ensures their unlinkability and enables retroactive anonymization of data, which fulfills the requirements of the GDPR's right to erasure [47, 86].

6.2.3 Forgetting Blockchain. Farshid et al. propose to achieve a GDPR-compliant blockchain by automatically deleting blocks from the blockchain after a certain amount of time has passed [26].

As the described network no longer contains a genesis block, joining it becomes a challenge. The authors propose to ask other nodes for the current block and just accept it if all the returned blocks are equal [26]. Since there is no way to verify that the received block reflects the true state of the network, joining it requires trust and does not satisfy the integrity constraint. Secondly, the nature of their approach prevents the existence of a chain history. Applications relying on the full history, specifically in the case of secure logs, would therefore not work with this algorithm. Furthermore, this proposal only achieves eventual GDPR compliance, since a block is only deleted after the predefined time has passed. If a user requests deletion of their data, this request cannot be fulfilled immediately. For this reason, it is questionable if the presented idea is compatible with the GDPR. Most problematically though, the data are only actually deleted if all nodes behave honestly [26]. Any node can simply decide not to delete older blocks, meaning that no additional privacy guarantees can be given.

In contrast to the forgetting blockchain, our solution does not require adaptation of the utilized blockchain software and is therefore easier to integrate into existing blockchains. Furthermore, we do not depend on the honesty of *arbitrary* and *unknown* nodes. In contrast, only one *known* party has *provable* access to additional identity information and can be held liable under the GDPR.

6.2.4 Redactable Blockchain. The reason that the immutability of data stored in blockchain can be guaranteed is the utilized hash function: An ideal hashing algorithm guarantees hashes that are one-way, which means impossible to reverse, and collision-free. Then, blocks cannot be replaced without notice, as any change would result in a new hash, thereby invalidating the chain.

Redactable blockchains utilize so-called *chameleon* hash functions to generate the hash of a block. Such hash functions are collision-resistant as long as a secret known as *trapdoor* is not known.

If one is in possession of said secret, they can efficiently compute colliding hashes [11, 24]. With the power to create hash collisions, any block can be replaced or even removed [11], making the blockchain effectively arbitrarily editable.

In order to function, such a redactable blockchain network needs a trusted third party that is in possession of the trapdoor and can decide which block to edit [11]. This constraint again requires trust, thereby calling into question the value of utilizing blockchain at all [60]. Furthermore, similar to the forgetting blockchain, every individual node needs to be trusted. Redactions are published as chain updates, allowing arbitrary nodes to make a copy of the removed or edited entry before updating their chain [60, 86]. This means that, effectively, no privacy guarantees can be given.

Our solution on the contrary does not require a trusted third party and functions even in the face of adversarial network participants.

6.2.5 Mutability by Consensus. The introduction of a trusted third party that can arbitrarily mutate data is inherently in conflict with the core concept of blockchain. Therefore, various proposals exist to weaken the immutability of blockchain while preserving the decentralized consensus for stored data. Concretely, that means allowing mutations only if consensus for them is ensured.

Deuber et al. create and formally prove an editable blockchain protocol [20, 62]. While any user can propose edits, the protocol ensures consensus-based voting on the proposals to prevent arbitrary edits. This also means that no trusted third party is required. The protocol is compatible with any consensus mechanism and even offers accountability of the performed edits. [20]

While this solution removes the need for a trusted third party, it does not solve the other issue of redactable and forgetting blockchains: every individual node in the network still needs to be trusted, as mutations are published as chain updates as well. Worse yet, the protocol introduces an additional issue in that it requires a majority of miners to act faithfully and actually perform the (legally mandated) mutations—something that it cannot guarantee by design [20].

In contrast to that, our solution functions even with of adversarial network participants, as noted above. Furthermore, we do not depend on the honesty of the miners and, better still, do not require any changes of the blockchain software.

7 LIMITATIONS AND DISCUSSION

Both our solution and its evaluation have limitations. To start with, in our design, we prioritize security and decentralization. That in turn means that other properties, such as data availability or exchange speed, are not optimized for. Regarding data availability, each individual node manages its own data and has to be reachable when accessing data. Should the node crash, be shut off, or otherwise disconnected from the network, the data consumer is prevented from continuing their work. In scenarios where the availability of the nodes is prioritized higher than their security and independence, we can imagine running user's nodes, e.g., on virtual servers. While this adds an attack surface and removes control from the user, it can improve availability. Importantly, though, the created usage logs are highly available, as the blockchain is accessible on all nodes. Considering exchange speed, meanwhile, we find that typical exchanges take at least 7 seconds to complete from the perspective of a data consumer. If fake chatter is active, the median exchange duration increases by a factor of 2.4. In corner cases with many nodes but few exchanges, this can significantly impact scalability in the default setting. However, in case of sufficient other traffic in the network masking the exchange, a simple heuristic could automatically deactivate fake chatter to minimize its impact. Still, the low exchange speed is one of the largest weaknesses of our approach. To alleviate this, requests can be pooled if more than one datum is to be requested from the same node. Beyond that, the only other way to improve this would be to reduce the security in less critical scenarios by, e.g., introducing a name server or lowering the number of protocol rounds. As always, this is a trade-off depending on the specific requirements. Especially outside the workplace or if sufficient employee protection exists, less secure solutions that offer vastly higher performance may be preferable. A sensible trade-off analysis should include security considerations but also cover factors such as cost, energy consumption, or difficulty to maintain, for example. Yet, it is important to acknowledge that critical situations cannot always be predicted. Therefore, we find it important to also build solutions for the most security-critical scenarios, especially if the adversity of an environment is hard to judge in advance.

Next, while our concept is fully decentralized, our implemented identity verification algorithm is built for the use case of an institutional IdP. The IdP is, by definition, a trusted third party. As noted in our concept (see Sec. 3.3.1), the widely known web of trust model can be utilized instead. We made our choice deliberately, though, as we mirror the real-world use case from industry where company-internal IdP servers are utilized for SSO. Furthermore, implementing web of trust is in our view not a technical novelty. Instead, we present a minimal-trust identity verification algorithm for the scenario of a company-internal IdP as a proof of concept. To realize fully decentralized inverse transparency, an alternative identity verification scheme such as web of trust is required, though.

In our evaluation, we analyze the GDPR compliance of Kovacs. Due to the focus of our paper, no formal legal analysis has been performed, meaning we cannot comprehensively answer this question. Instead, we used insights from related works to deduce the GDPR compliance of our solution. At this moment, if and how blockchain can be used in a GDPR-compliant way has not been comprehensively answered yet, neither from a technical nor a legal perspective [see, e.g., 18, 37, 81, 87]. Also, the concrete application use case is essential in conclusively determining the GDPR compliance of a solution [52, 87]. Therefore, before deploying Kovacs, a full legal analysis including the concrete application scenario is necessary.

Furthermore, our performance and scalability evaluations are limited in their significance due to their artificial nature. With our experiments, we tried to measure common usage scenarios and patterns. Yet, real-world usage may differ from our tests, which can influence the performance. As an example, a network made up of many nodes where only comparatively few nodes actually request data presents a worst case scenario for our fake chatter implementation. The seldom communication by other nodes would require fake chatter to ensure privacy, yet the large number of potential communication partners could mean long wait times until the peer-to-peer connections are established. The relevance of such performance bottlenecks in practice depends on the concrete usage patterns, which means real-world evaluations could be a useful next step.

Our focus was on the security of Kovacs. Even with the best technical protections, though, individual users remain as an often-abused attack surface [see, e.g., 9, 56, 67, 90]. For most data that we store, there is no danger of users unwillingly leaking information about other parties except for themselves, with one exception: *data owners* could be tricked or hacked to reveal the identities of *consumers* of their data. In our current implementation, it is impossible to prevent this case, yet we consider the attack surface to be acceptably small. To get access to a meaningful dataset about the usage pattern of a data consumer, an adversary would have to find and hack or phish each individual data owner of data accessed by said data consumer. We consider that infeasible.

Finally, to expand on these points, there has been broader discussion on what constitutes "good enough" software security and how to make objective judgments about it [see, e.g., 21, 71, 83]. Tøndel et al. suggest to not only consider the system from the perspective of the adversary (as we have done), but to additionally factor in other perspectives such as those of users or operators [83, p. 364]. Following their proposal, it might therefore be sensible to conduct a broader analysis of the system that also covers these perspectives before deploying it. This could be important to ensure user acceptance and usability, as well as to address potential practical issues with deployment and operation that might otherwise hinder adoption.

8 CONCLUSION

The goal of inverse transparency is to protect employees from misusage of their data. Yet, current technical realizations are inherently centralized, which requires trust and opens possibilities for tampering with the logs by, e.g., the employer. Permissionless blockchain therefore is an intuitive choice for inverse transparency logs, as it is by design decentralized and immutable. Realizing fully decentralized inverse transparency with blockchain requires us to tackle two main issues, though: (1) ensuring non-repudiable data exchanges without a trusted third party and (2) complying with GDPR requirements, specifically confidentiality and the right to erasure. With the Kovacs system, we solve both of these issues. For accountable inverse transparency, its new-usage protocol enables decentralized and non-repudiable data exchange. To enable GDPR compliance, its pseudonym generation algorithm guarantees unlinkability and anonymity of stored data, while enabling proof of ownership and authenticity. Our block structure and decentralized deployment architecture allow individuals to efficiently query for and read arbitrary usage log entries concerning their data, while protecting them from attacks on their confidentiality by adversaries.

In our analysis, we find that Kovacs provides a high level of security and protects against expected attacks on the confidentiality of the logs. It fulfills the requirements of the GDPR by enabling confidentiality and the rights to erasure and rectification, while at the same time benefiting from the properties of permissionless blockchain, specifically guaranteeing the integrity of the logged data. Related works require either the use of a permissioned blockchain, necessitating a trusted third party, or modifying the utilized hashing algorithms or blockchain software to make the blockchain mutable. Both approaches entail effectively giving up the advantages of blockchain, thereby calling into question the use of blockchain in the first place. Our performance and scalability evaluations demonstrate the practicality of our implementation. While our focus on security impacts performance, the exchange duration and query times stay manageable for typical workloads. If exchange speed is prioritized, our protocol can be adapted flexibly. Furthermore, we find that Kovacs scales linearly considering both the retrieval time and storage size, showing its practicality. The additional metadata stored mean higher storage requirements than a minimal database, but the logs are still sufficiently small.

To conclude, the Kovacs system realizes decentralized, non-repudiable, secure, and GDPR-compliant inverse transparency based on blockchain. Its design does not require a trusted third party, it can be used with any existing blockchain software without necessitating changes, and it is secure and flexible enough for integration even into highly adversarial settings.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF) under grant no. 5091121.

REFERENCES

[1] Martín Abadi and Neal Glew. 2002. Certified email with a light on-line trusted third party: design and implementation. In *Proceedings of the 11th International Conference on World Wide Web*. ACM, 387–395.

[2] Alfarez Abdul-Rahman. 1997. The PGP trust model. *EDI-Forum: The Journal of Electronic Commerce* 10 (1997), 27–31.

[3] Rafael Accorsi. 2010. BBox: A distributed secure log architecture. In *Proceedings of the 2010 European Public Key Infrastructure Workshop (Lecture Notes in Computer Science, Vol. 6711)*. Springer, 109–124.

[4] Alessandro Aldini and Roberto Gorrieri. 2002. Security analysis of a probabilistic non-repudiation protocol. In *Proceedings of the 2nd Joint International Workshop von Process Algebra and Probabilistic Methods, Performance Modeling and Verification (Lecture Notes in Computer Science, Vol. 2399)*. Springer, 17–36.

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić,

Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the 13th European Conference on Computer Systems*. ACM, Article 30, 15 pages.

[6] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. 2013. Evaluating user privacy in bitcoin. In *Proceedings of the 17th International Conference on Financial Cryptography and Data Security (Lecture Notes in Computer Science, Vol. 7859)*. Springer, 34–51.

[7] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain. In *Proceedings of the 2nd Italian Conference on Cyber Security*.

[8] Benny Applebaum, Naama Haramaty-Krasne, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. 2017. Low-complexity cryptographic hash functions. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (Leibniz International Proceedings in Informatics)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Article 7, 31 pages.

[9] Iván Arce. 2003. The weakest link revisited. *IEEE Security & Privacy* 1, 2 (2003), 72–76.

[10] Nadarajah Asokan, Matthias Schunter, and Michael Waidner. 1997. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*. ACM, 7–17.

[11] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton Andrade. 2017. Redactable blockchain – or – Rewriting history in bitcoin and friends. In *Proceedings of the 2nd IEEE European Symposium on Security and Privacy*. IEEE, 111–126.

[12] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: simpler, smaller, fast as MD5. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security (Lecture Notes in Computer Science, Vol. 7954)*. Springer, 119–135.

[13] Toras Pangidoan Batubara, Syahril Efendi, and Erna Budhiarti Nababan. 2021. Analysis performance BCRYPT algorithm to improve password security from brute force. *Journal of Physics: Conference Series* 1811, 1, Article 012129 (2021).

[14] David Brin. 1998. *The Transparent Society*. Basic Books.

[15] California Consumer Privacy Act. 2018. An act to add Title 1.81.5 (commencing with Section 1798.100) to Part 4 of Division 3 of the Civil Code, relating to privacy (California Consumer Privacy Act of 2018). *Assembly Bill* 375 (2018), 1–24.

[16] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. 2020. A survey on Ethereum systems security: vulnerabilities, attacks, and defenses. *Comput. Surveys* 53, 3 (2020), 1–43.

[17] Tom Coffey and Puneet Saidha. 1996. Non-repudiation with mandatory proof of receipt. *ACM SIGCOMM Computer Communication Review* 26, 1 (1996), 6–17.

[18] Pedro Garcia de Pesquera Villagran. 2022. Blockchain technology and the general data protection regulation: an inevitable conflict? *Amsterdam Law Forum* 14 (2022), 61–64.

[19] Hans Delfs and Helmut Knebl. 2007. Public-key cryptography. In *Introduction to Cryptography*. Springer, Chapter 3, 33–80.

[20] Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. 2019. Redactable blockchain in the permissionless setting. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*. IEEE, 124–138.

[21] John B. Dickson. 2011. Software security: is ok good enough?. In *Proceedings of the 1st ACM Conference on Data and Application Security and Privacy*. ACM, 25–26.

[22] Morris Dworkin. 2007. *Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC*. NIST Special Publication 800-38D. National Institute of Standards and Technology.

[23] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 967–984.

[24] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. 2019. Trapdoor hash functions and their applications. In *Proceedings of the 39th Annual International Cryptology Conference (Lecture Notes in Computer Science, Vol. 11694)*. Springer, 3–32.

[25] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. 2020. OptiSwap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ACM, 543–557.

[26] Simon Farshid, Andreas Reitz, and Peter Roßbach. 2019. Design of a forgetting blockchain: A possible way to accomplish GDPR compatibility. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*. 7087–7095.

[27] Martin Florian, Johannes Walter, and Ingmar Baumgart. 2015. Sybil-resistant pseudonymization and pseudonym change without trusted third parties. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. ACM, 65–74.

[28] Chunpeng Ge, Siwei Sun, and Pawel Szalachowski. 2019. Permissionless blockchains and secure logging. In *Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency*. IEEE, 56–60.

[29] General Data Protection Regulation. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union* 59 (2016), 1–88.

[30] Maren Gierlich-Joas, Thomas Hess, and Rahild Neuburger. 2020. More self-organization, more control—or even both? Inverse transparency as a digital leadership concept. *Business Research* 13, 3 (2020), 921–947.

[31] go-ethereum Authors. 2022. Private networks. https://geth.ethereum.org/docs/interface/private-network

[32] Mateusz Godyn, Michal Kedziora, Yingying Ren, Yongxin Liu, and Houbing Herbert Song. 2022. Analysis of solutions for a blockchain compliance with GDPR. *Scientific Reports* 12, 1 (2022), 15021.

[33] AKM Bahalul Haque, AKM Najmul Islam, Sami Hyrynsalmi, Bilal Naqvi, and Kari Smolander. 2021. GDPR compliant blockchains–a systematic literature review. *IEEE Access* 9 (2021), 50593–50606.

[34] Mike Hintze and Khaled El Emam. 2018. Comparing the benefits of pseudonymisation and anonymisation under the GDPR. *Journal of Data Protection & Privacy* 2, 2 (2018), 145–158.

[35] Luis-Daniel Ibáñez, Kieron O'Hara, and Elena Simperl. 2018. *On blockchains and the general data protection regulation.* Technical Report. EU Blockchain Forum and Observatory. https://eprints.soton.ac.uk/422879/

[36] ISO 25237:2017 2017. *Health informatics — Pseudonymization.* Standard. International Organization for Standardization, Geneva, CH.

[37] Amandine Jambert. 2019. Blockchain and the GDPR: a data protection authority point of view. In *Proceedings of the 12th IFIP WG 11.2 International Conference on Information Security Theory and Practice (Lecture Notes in Computer Science, Vol. 11469).* Springer, 3–6.

[38] Florian Kelbert and Alexander Pretschner. 2018. Data usage control for distributed systems. *ACM Transactions on Privacy and Security* 21, 3, Article 12 (2018), 32 pages.

[39] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. 1998. Secure applications of low-entropy keys. In *Proceedings of the 1st International Workshop on Information Security (Lecture Notes in Computer Science, Vol. 1396).* Springer, 121–134.

[40] Sunny King and Scott Nadal. 2012. *PPCoin: peer-to-peer crypto-currency with proof-of-stake.* White Paper. Peercoin. https://www.peercoin.net/read/papers/peercoin-paper.pdf

[41] Mikko Kiviharju. 2017. On the fog of RSA key lengths: Verifying public key cryptography strength recommendations. In *Proceedings of the 2017 International Conference on Military Communications and Information Systems.* IEEE, 1–8.

[42] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. 2013. On the security of the TLS protocol: a systematic analysis. In *Proceedings of the 33rd Annual Cryptology Conference (Lecture Notes in Computer Science, Vol. 8042).* Springer, 429–448.

[43] Steve Kremer, Olivier Markowitch, and Jianying Zhou. 2002. An intensive survey of fair non-repudiation protocols. *Computer Communications* 25, 17 (2002), 1606–1621.

[44] Tian Lan, Zhiguang Qin, Yang Zhao, Hu Xiong, and Li Liu. 2007. A gradual and optimistic fair exchange protocol. In *Proceedings of the 2007 International Conference on Communications, Circuits and Systems.* IEEE, 452–456.

[45] Cedric Lauradoux, Konstantinos Limniotis, Marit Hansen, Meiko Jensen, and Petros Eftasthopoulos. 2021. *Data pseudonymisation: advanced techniques and use cases.* Technical Report. European Union Agency for Cybersecurity (ENISA). https://www.enisa.europa.eu/publications/data-pseudonymisation-advanced-techniques-and-use-cases/

[46] Arjen K. Lenstra and Eric R. Verheul. 2001. Selecting cryptographic key sizes. *Journal of Cryptology* 14, 4 (2001), 255–293.

[47] Konstantinos Limniotis and Marit Hansen. 2019. *Recommendations on shaping technology according to GDPR provisions – an overview on data pseudonymisation.* Technical Report. European Union Agency for Cybersecurity (ENISA). https://www.enisa.europa.eu/publications/recommendations-on-shaping-technology-according-to-gdpr-provisions

[48] Jing Liu and Laurent Vigneron. 2010. Design and verification of a non-repudiation protocol based on receiver-side smart card. *IET Information Security* 4, 1 (2010), 15–29.

[49] Yang Liu, Debiao He, Mohammad S. Obaidat, Neeraj Kumar, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. 2020. Blockchain-based identity management systems: a review. *Journal of Network and Computer Applications* 166, Article 102731 (2020).

[50] Gabriel Loyola Daiqui and Arnau Oller Prat. 2020. *A Proof of Authority blockchain protocol for secure logging.* Seminar paper. Technical University of Munich. https://mediatum.ub.tum.de/1689644

[51] Atul Luykx, Bart Mennink, and Samuel Neves. 2016. Security analysis of BLAKE2's modes of operation. *IACR Transactions on Symmetric Cryptology* 2016, 1 (2016), 158–176.

[52] Tom Lyons, Ludovic Courcelas, and Ken Timsit. 2018. *Blockchain and the GDPR.* Thematic Report. The European Union Blockchain Observatory and Forum.

[53] Dindayal Mahto, Danish Ali Khan, and Dilip Kumar Yadav. 2016. Security analysis of elliptic curve cryptography and RSA. In *Proceedings of the 2016 World Congress on Engineering*, Vol. I. IAENG, 419–422.

[54] Olivier Markowitch and Yves Roggeman. 1999. Probabilistic non-repudiation without trusted third party. In *Proceedings of the 2nd Conference on Security in Communication Networks*. 25–36.

[55] Aleecia M. McDonald and Lorrie Faith Cranor. 2008. The cost of reading privacy policies. *I/S: A Journal of Law and Policy for the Information Society* 4, 3 (2008), 543–568.

[56] Kevin D. Mitnick and William L. Simon. 2002. *The art of deception: Controlling the human element of security.* John Wiley & Sons, New York, NY.

[57] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel. 2018. A survey on essential components of a self-sovereign identity. *Computer Science Review* 30 (2018), 80–86.

[58] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[59] National Institute of Standards and Technology. 2001. Advanced encryption standard (AES). (2001). https://doi.org/10.6028/NIST.FIPS.197

[60] Ugo Pagallo, Eleonora Bassi, Marco Crepaldi, and Massimo Durante. 2018. Chronicle of a clash foretold: Blockchains and the GDPR's right to erasure.. In *Proceedings of the 31st Annual Conference on Legal Knowledge and Information Systems*. 81–90.

[61] Alois Paulin and Tatjana Welzer. 2013. A universal system for fair non-repudiable certified e-mail without a trusted third party. *Computers & Security* 32 (2013), 207–218.

[62] Eugenia Politou, Fran Casino, Efthimios Alepis, and Constantinos Patsakis. 2019. Blockchain mutability: challenges and proposed solutions. *IEEE Transactions on Emerging Topics in Computing* 9, 4 (2019), 1972–1986.

[63] Alexander Pretschner, Manuel Hilty, and David Basin. 2006. Distributed usage control. In *Communications of the ACM*. 39–44.

[64] Alex Preukschat and Drummond Reed. 2021. *Self-sovereign identity.* Manning.

[65] Niels Provos and David Mazieres. 1999. A future-adaptable password scheme. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*. USENIX, 81–91.

[66] Michael O. Rabin. 1983. Transaction protection by beacons. *J. Comput. System Sci.* 27, 2 (1983), 256–267.

[67] Marissa Randazzo, Michelle Keeney, Eileen Kowalski, Dawn Cappelli, and Andrew Moore. 2005. *Insider threat study: Illicit cyber activity in the banking and finance sector.* Technical Report CMU/SEI-2004-TR-021. Software Engineering Institute, Carnegie Mellon University.

[68] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*, Yaniv Altshuler, Yuval Elovici, Armin B. Cremers, Nadav Aharony, and Alex Pentland (Eds.). Springer, 197–223.

[69] Eric Rescorla. 2000. *HTTP Over TLS.* RFC 2818. RFC Editor. https://www.rfc-editor.org/rfc/rfc2818.txt

[70] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[71] Ravi Sandhu. 2003. Good-enough security. *IEEE Internet Computing* 7, 1 (2003), 66–68.

[72] Christian Schaefer and Christine Edman. 2019. Transparent logging with Hyperledger Fabric. In *Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency*. IEEE, 65–69.

[73] Richard Schneider. 2021. Multicast DNS (mDNS). https://github.com/libp2p/specs/blob/master/discovery/mdns.md

[74] Marten Seemann and Ian Davis. 2021. libp2p DHT example implementation. https://github.com/libp2p/go-libp2p/blob/b7bee3855cb86e50440e23b463605ea874c38787/examples/chat-with-rendezvous/chat.go#L128=

[75] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[76] Louis Shekhtman and Erez Waisbard. 2021. EngraveChain: A blockchain-based tamper-proof distributed log system. *Future Internet* 13, 6, Article 143 (2021).

[77] Gurpreet Singh. 2013. A study of encryption algorithms (RSA, DES, 3DES and AES) for information security. *International Journal of Computer Applications* 67, 19 (2013).

[78] Corwin Smith et al. 2023. Proof-of-stake (PoS). https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

[79] Manu Sporny, Dave Longley, and David Chadwick. 2022. *Verifiable credentials data model v1.1.* W3C Recommendation. World Wide Web Consortium. https://www.w3.org/TR/2022/REC-vc-data-model-20220303/

[80] Péter Szilágyi. 2021. Geth v1.10.0 – offline pruning. https://blog.ethereum.org/2021/03/03/geth-v1-10-0/#offline-pruning

[81] Unal Tatar, Yasir Gokce, and Brian Nussbaum. 2020. Law versus technology: Blockchain, GDPR, and tough tradeoffs. *Computer Law & Security Review* 38, Article 105454 (2020), 11 pages.

[82] Paul J. Taylor, Tooska Dargahi, Ali Dehghantanha, Reza M. Parizi, and Kim-Kwang Raymond Choo. 2020. A systematic literature review of blockchain cyber security. *Digital Communications and Networks* 6, 2 (2020), 147–156.

[83] Inger Anne Tøndel, Daniela Soares Cruzes, and Martin Gilje Jaatun. 2020. Achieving "good enough" software security: the role of objectivity. In *Proceedings of the 2020 International Conference on Evaluation and Assessment in Software Engineering*. ACM, 360–365.

[84] Aizhan Tursunbayeva, Stefano Di Lauro, and Claudia Pagliari. 2018. People analytics—a scoping review of conceptual boundaries and value propositions. *International Journal of Information Management* 43 (2018), 224–247.

[85] Aizhan Tursunbayeva, Claudia Pagliari, Stefano Di Lauro, and Gilda Antonelli. 2021. The ethics of people analytics: risks, opportunities and recommendations. *Personnel Review* 51, 3 (2021), 900–921.

[86] David van de Giessen. 2019. *Blockchain and the GDPR's right to erasure*. Essay. University of Twente.

[87] Patrick Van Eecke and Anne-Gabrielle Haie. 2018. Blockchain and the GDPR: the EU blockchain observatory report. *European Data Protection Law Review* 4/2018, 4 (2018), 531–534.

[88] Laurens Van Hoye, Pieter-Jan Maenhaut, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2019. Logging mechanism for cross-organizational collaborations using Hyperledger Fabric. In *Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency*. IEEE, 352–359.

[89] Liang Wang, Jiayan Liu, and Wenyuan Liu. 2021. Staged data delivery protocol: A blockchain-based two-stage protocol for non-repudiation data delivery. *Concurrency and Computation: Practice and Experience* 33, 13, Article e6240 (2021).

[90] Ryan West, Christopher Mayhorn, Jefferson Hardee, and Jeremy Mendel. 2009. The weakest link: A psychological perspective on why users make poor security decisions. In *Social and Human Elements of Information Security: Emerging Trends and Countermeasures*, Manish Gupta and Raj Sharman (Eds.). IGI Global, 43–60.

[91] Christian Wirth and Michael Kolain. 2018. Privacy by blockchain design: A blockchain-enabled GDPR-compliant approach for handling personal data. In *Proceedings of the 2018 ERCIM Workshop on Blockchain Engineering*. EUSSET.

[92] Gavin Wood. 2014. *Ethereum: a secure decentralised generalised transaction ledger*. Yellow Paper. https://gavwood.com/paper.pdf

[93] Gavin Wood. 2015. Proof-of-authority private chains. https://github.com/ethereum/guide/blob/master/poa.md

[94] Liang Zhang, Haibin Kan, Yang Xu, and Jinhao Ran. 2021. Revocable data sharing methodology based on SGX and blockchain. In *Proceedings of the 15th International Conference on Network and System Security (Lecture Notes in Computer Science, Vol. 13041)*. Springer, 61–78.

[95] Ning Zhang and Qi Shi. 1996. Achieving non-repudiation of receipt. *Comput. J.* 39, 10 (1996), 844–853.

[96] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. 2017. An overview of blockchain technology: architecture, consensus, and future trends. In *Proceedings of the 2017 International Congress on Big Data*. IEEE, 557–564.

[97] Jianying Zhou and Dieter Gollmann. 1996. Observations on non-repudiation. In *Proceedings of the 1996 International Conference on the Theory and Application of Cryptology and Information Security (Lecture Notes in Computer Science, Vol. 1163)*. Springer, 133–144.

[98] Valentin Zieglmeier. 2023. The inverse transparency toolchain. (2023). Manuscript in review.

[99] Valentin Zieglmeier and Gabriel Loyola Daiqui. 2021. GDPR-compliant use of blockchain for secure usage logs. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 313–320.

[100] Valentin Zieglmeier, Maren Gierlich-Joas, and Alexander Pretschner. 2022. Increasing employees' willingness to share: Introducing appeal strategies for people analytics. In *Proceedings of the 13th International Conference on Software Business (Lecture Notes in Business Information Processing, Vol. 463)*. Springer, 213–226.

[101] Valentin Zieglmeier and Alexander Pretschner. 2021. Trustworthy transparency by design. arXiv:2103.10769 [cs.SE]

[102] Valentin Zieglmeier and Alexander Pretschner. 2023. Rethinking people analytics with inverse transparency by design. *Proceedings of the ACM on Human-Computer Interaction* 7 (2023). Forthcoming.

[103] Guy Zyskind, Oz Nathan, and Alex 'Sandy' Pentland. 2015. Decentralizing privacy: Using blockchain to protect personal data. In *Proceedings of the 2015 IEEE Security and Privacy Workshops*. IEEE, 180–184.