Efficient Public Verification of Confidential Supply-Chain Transactions

Kilian Becher*, Mirko Schäfer*, Axel Schröpfer†, and Thorsten Strufe‡
* Technische Universität Dresden, Germany
† SAP SE, Germany
‡ Karsruhe Institute of Technology, Germany
Email: *{kilian.becher, mirko.schaefer}@tu-dresden.de, †axel.schroepfer@sap.com, ‡thorsten.strufe@kit.edu

2022 IEEE Conference on Communications and Network Security (CNS) | 978-1-6654-6255-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CNS56114.2022.9947231

***Abstract*—Ensuring sustainable sourcing of crude materials and production of goods is a pressing problem in consideration of the growing world population and rapid climate change. Supply-chain traceability systems based on distributed ledgers can help to enforce sustainability policies like production limits.**

We propose two mutually independent distributed-ledger-based protocols that enable public verifiability of policy compliance. They are designed for different supply-chain scenarios and use different privacy-enhancing technologies in order to protect confidential supply-chain data: secret sharing and homomorphic encryption. The protocols can be added to existing supply-chain traceability solutions with minor effort. They ensure confidentiality of transaction details and offer public verifiability of producers' compliance, enabling institutions and even end consumers to evaluate sustainability of supply chains. Through extensive theoretical and empirical evaluation, we show that both protocols perform verification for lifelike supply-chain scenarios in perfectly practical time.

***Index Terms*—Distributed ledger technology, homomorphic encryption, proxy re-encryption, secret sharing, supply-chain verification**

I. Introduction

Today's rapidly growing world population comes with an increasing demand for food and plantations. Palm oil, the world's most consumed vegetable oil [26], is used in numerous products including groceries, beauty products, and biofuel. Most oil palms are cultivated in Indonesia and Malaysia, where palm-oil production is responsible for deforestation and endangerment of wildlife. To achieve sustainable palm-oil production, initiatives like the Roundtable on Sustainable Palm Oil (RSPO)[1] created sustainability standards. They issue certificates for sustainable palm-oil production and use audits to prevent fraudulent behavior like mixing sustainable and non-sustainable palm oil or overselling sustainable palm oil. RSPO allows trading of credits that represent a specific amount of certified sustainable palm oil. This allows manufacturers to buy credits that entitle them to offset a particular amount of palm oil and demonstrate their commitment to sustainability. Despite these efforts, RSPO is also criticized for investing too little in independent and credible audits [29].

An alternative to regular audits is supply-chain transparency through distributed ledgers, such as proposed by the Sustain Consortium.[2] Tracking every supply-chain action on a public ledger allows third parties to verify sustainable sourcing and detect non-compliant or fraudulent behavior. The latter can cause legal consequences and fines, affect supply-chain participants' reputation and credibility, and reduce customers' trust in their suppliers, which can affect future business relations. Therefore, a high chance of uncovering fraud adds pressure on companies to comply with policies and regulations.

Distributed ledgers inherently ensure traceability, transparency, and integrity [25]. Confidentiality of transaction details, however, is not a default feature and intuitively seems to contradict transparency. Thus, supply-chain participants might have privacy concerns: Leaking confidential transaction details could jeopardize their competitive advantages. That is particularly the case for public ledgers, which are often used to enable public verifiability, i.e., verifiability for everyone including end consumers. Existing solutions fail to combine privacy with public verifiability [1, 13, 15].

Even though sustainable palm oil production is a pressing problem itself, we expand the underlying problem to general settings to enable a wide variety of use cases. Our results can easily be applied to other supply-chain scenarios with production restrictions, such as ethical fishing and mining.

We propose two independent verification protocols that work as a confidentiality layer in distributed-ledger-based supply-chain traceability solutions. They are designed for different volatility levels of producer-customer relationships to cover a large variety of supply chains and different stages of production. They could be long-lasting or last just for a single transaction. We target two independent supply-chain scenarios. The first scenario covers long-lasting producer-customer relations, e.g., between palm-oil mills and manufacturers of palm-oil-based products. The second scenario fits short-term relations between static producers and volatile customers, e.g., manufacturers and (end) consumers.

We design individual protocols for each of these two scenarios. The protocol for the first scenario uses secret sharing (Π_S), the second employs a combination of fully homomorphic encryption and proxy re-encryption (Π_{PRE}). These technologies enable different protocol designs and allow them to properly fit the respective supply-chain scenario.

This paper is dedicated to the memory of Axel Schröpfer, whose passion for privacy-enhancing technologies was unsurpassed.

[1]https://rspo.org

[2]https://sustainalliance.org

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

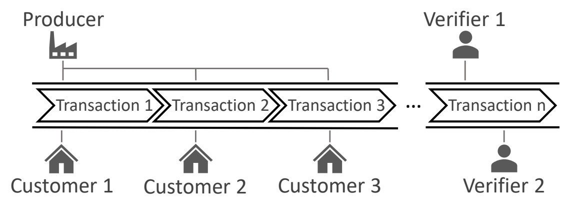

Fig. 1: Example Supply-Chain Traceability Scenario

However, they cause different performances, leading to a trade-off between volatility and performance (see Section V).

II. Background and Related Work

A. Scenario Description

Figure 1 shows an example supply-chain traceability scenario with a producer that performs transactions with three customers via a distributed ledger. Independent verifiers read from the ledger to verify compliance. In the palm-oil case, producers could be refineries, their customers could be food manufacturers, and verifiers could be end consumers or NGOs.

Our solution aims for public verifiability of the producers' compliance with production limits: Supply-chain participants, consumers, NGOs, etc. may verify whether a potentially malicious producer exceeds a specific limit, i.e., maximum amount, which may change over time. To do so, the verifier checks whether the producer has a positive production balance regarding this maximum amount.

We require balance verification to guarantee confidentiality of the transaction amounts. These amounts could leak supply-chain participants' trade secrets like bill of materials and recipes of products as well as warehouse stock and storage capacities. Those ensure participants' competitive advantages.

Distributed-ledger-based supply-chain traceability systems reflect real-life business relations between producers and their customers, which transfer physical amounts. Such relations are trade knowledge in many industries and transactions can be observed in the real world. We address transaction anonymity through pseudonymization of participants' identifiers.

We reasonably assume that maximum amounts are public knowledge, e.g., implied by public sustainability standards or legal regulations such as the EU's Renewable Energy Directive.[3] Those standards and regulations can change. As for the palm-oil industry, we allow credit trading. Producers can purchase credits that entitle them to offset a particular amount of palm oil. We take into account that producers can purchase multiple credits over time. Hence, the maximum amount that a producer is entitled to process can also change over time.

For simplicity of notations and without loss of generality, we assume a single producer P that transfers n amounts x_i of a particular good to customers C_i in the physical world. We use i as the transaction index. Every physical-world transaction has a digital counterpart that is reflected on a distributed ledger DL in a privacy-preserving form. We denote by x_{max} the maximum amount that P is entitled to produce or process. This x_{max} could be stored on DL where everyone can access it. Representing physical amounts or masses, x_i and x_{max} are positive by nature. For balance verification, verifiers V aim to determine whether the total transferred amount $x = \sum_{i=1}^{n} x_i$ is below or equal to x_{max}, i.e., whether the balance

$$\delta = x_{max} - \sum_{i=1}^{n} x_i \tag{1}$$

is non-negative. A negative verification result indicates fraud by the producer, which can be reported by publishing an alert entry with details on the accusation on the distributed ledger.

Balance verification should be efficient in terms of computational resources and storage requirements. The overall solution should scale well in the complexity of the supply chain.

B. Adversary Model

Our protocols guarantee detection of producer fraud while hiding the transaction amounts. We consider two adversaries. The first adversary tries to convince verifiers that the producer behaved honestly even if it did not. We assume malicious [27] producers and exclude collusion with customers. This exclusion is reasonable as all supply-chain transactions from sourcing to manufacturing of end products have to be reflected on the ledger to allow traceability. Active involvement of customers in fraud, e.g., in off-ledger transactions, shifts the burden to convince their customers and eventually end consumers to them in their role as supplier. Hence, off-ledger transactions imply that all follow-up transactions have to be performed off-ledger, including purchasing by end consumers.

The second adversary tries to learn amounts x_i of transactions that it was not involved in. It could either corrupt customers or verifiers, but without collusion with the producer. Here, we allow semi-honest [27] customers and verifiers.

For direct communication between parties, we require pairwise secure, i.e., secret and authentic, channels, e.g., via TLS.

C. Related Work

a) Distributed-Ledger-Based Supply-Chain Verification: An investigation of the suitability of blockchains for supply-chain traceability is presented in [15]. It focuses on transparency and suggests to hash confidential information, which rules out further data processing, including verifications.

AgriBlockIoT [13] is a blockchain-based system that enables traceability and auditability for agriculture and food supply chains. Entries are written by IoT devices, smart contracts react autonomously to data anomalies. AgriBlockIoT does not ensure privacy of the data written into the blockchain.

A blockchain-based traceability system for the textile and clothing industry is presented in [1]. It aims to establish technology-based trust among supply-chain participants based on private blockchains. The latter hinders public verifiability.

A protocol for verification of commodity ratios is presented in [4]. Supply-chain transactions and their dependencies are reflected in a distributed ledger as a tree-like data structure. Confidentiality is ensured via homomorphic encryption.

[3]https://eur-lex.europa.eu/eli/dir/2018/2001/oj

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

b) Zero-Knowledge Arguments and Proofs: Zero-knowledge arguments and proofs of knowledge allow a prover P to convince a verifier V that P knows a secret x without revealing anything about x besides that it is known to P [22].

zkSNARKs [6] enable constant-size proofs that can be verified efficiently. They rely on a trusted setup for common reference string (CRS) generation. A multi-party computation protocol for creating CRSs for zkSNARKs is presented in [9]. SNIPs [16] can be used to prove syntactical correctness and are much more efficient than zkSNARKs. zkSTARKs [5] do not require a trusted setup but suffer from larger proofs. Bulletproofs [11] is an efficient non-interactive zero-knowledge proof protocol. It does not require a trusted setup and has logarithmic proof size, typically several hundred bytes long.

We aim for a storage-efficient protocol that writes as little data into the distributed ledger as possible. This rules out complex non-interactive proofs like zkSTARKs. Furthermore, by nature, these arguments and proofs prove particular statements which cannot be adjusted later without creating a new proof. In our scenario, if the maximum amount changes subsequently, so does the balance. Hence, previous proofs would prove outdated statements. Instead, we aim for an approach that allows verification of the balance even if the maximum amount or the verification function itself changes over time, i.e., for variable statements.

c) Distributed-Ledger-Based Privacy-Preserving Transaction Verification: A major line of work that deals with privacy-preserving verifications in distributed-ledger scenarios can be found in the field of cryptocurrencies such as Bitcoin.[4]

Confidential assets [31] is a scheme that hides transaction amounts and asset types of (Bitcoin) transactions and ensures public verifiability of the fact that the input amounts equal the output amounts of transactions. It allows verification of fixed statements and, thus, does not support variable maximum amounts. The same applies to Zether [12], a decentralized confidential payment mechanism for smart-contract platforms such as Ethereum.[5]

Zerocash [7] hides amounts, balances, and identities. Transactions are verified by each node, causing a delay at transaction time. A payment system based on Zerocash is presented in [20]. Both use zkSNARKs for privacy-preserving balance verification, being subject to the above-mentioned limitations.

Provisions [18] allows Bitcoin exchanges to prove their solvency while hiding customers' addresses and balances. It is based on Pedersen commitments [30] and zero-knowledge proofs. zkLedger [28], a system for auditing banks' balances in banking networks, allows public verification and offers rich auditing capabilities. It uses a tabular structure with a row for each transaction and a column for each existing bank. FabZK [24] is an extension to Hyperledger Fabric.[6] It hides transaction details like amounts and transaction relations and enables periodic ledger auditing. It uses a tabular structure to hide the transaction graph. Being based on zero-knowledge proofs, Provisions, zkLedger, and FabZK only allow verification of fixed statements and, therefore, do not lend themselves to our scenario. With their tabular structure, the latter two have quadratic verification complexity and, thus, do not scale well.

Solidus [14], a protocol for bank-intermediated ledgers, ensures privacy of transaction amounts and the transaction graph based on publicly verifiable oblivious RAM machines. Balance audits involve decryption of transaction ciphertexts by auditors. It does not fit our scenario without trusted auditors.

[4]https://bitcoin.org

[5]https://ethereum.org

[6]https://hyperledger.org

III. Preliminaries

A. Homomorphic Encryption

An asymmetric cryptosystem is a tuple $\mathcal{CS} = (G, E, D)$ consisting of three polynomial-time algorithms. The probabilistic key-generation algorithm $G(\cdot)$ takes as input a security parameter κ and outputs a pair (pk, sk) of a (public) encryption key pk and a (secret) decryption key sk. The (probabilistic) encryption algorithm $E(\cdot)$ takes as input a plaintext $m \in \mathcal{M}$ and pk and outputs the ciphertext $c = E_{pk}(m) \in \mathcal{C}$. $\mathcal{M}$ and $\mathcal{C}$ denote the plaintext and ciphertext space, respectively. We require $\mathcal{M} \subset \mathbb{Z}$, real numbers need to be scaled and rounded prior to encryption. The decryption algorithm $D(\cdot)$ takes as input a ciphertext c and sk and outputs the plaintext $m = D_{sk}(c) = D_{sk}(E_{pk}(m))$.

Homomorphic encryption (HE) schemes allow computations on ciphertexts. Assume ciphertexts $E_{pk}(m_1), E_{pk}(m_2)$ encrypted under the same pk. A cryptosystem $\mathcal{CS}$ is homomorphic if it offers an operation $\circ$ on $\mathcal{C}$ that maps to a homomorphic operation $\bullet$ on $\mathcal{M}$, such that $E_{pk}(m_1) \circ E_{pk}(m_2)$ yields an encryption of $m_1 \bullet m_2$. This can be formalized as $D_{sk}(E_{pk}(m_1) \circ E_{pk}(m_2)) = m_1 \bullet m_2$. Common homomorphic operations are addition and multiplication.

$$D_{sk}(E_{pk}(m_1) \oplus E_{pk}(m_2)) = m_1 + m_2 \tag{2}$$

$$D_{sk}(E_{pk}(m_1) \odot E_{pk}(m_2)) = m_1 \cdot m_2 \tag{3}$$

Partially homomorphic encryption (PHE) schemes enable either addition or multiplication of the underlying plaintexts. Fully homomorphic encryption (FHE) schemes provide both addition and multiplication and allow the privacy-preserving evaluation of arbitrary arithmetic functions. A promising FHE scheme is the Fan-Vercauteren variant [19] of Brakerski's scale-invariant scheme [10], which works over polynomial rings and relies on the assumed hardness of the Ring Learning With Errors (RLWE) problem. We refer to this scheme as the BFV scheme.

B. Proxy Re-Encryption

Re-encryption transforms a ciphertext $c_1 = E_{pk_1}(m)$ encrypted under a key pk_1 into a ciphertext $c_2 = E_{pk_2}(m)$ of the same plaintext, encrypted under a different key pk_2. Proxy re-encryption (PRE) allows an untrusted party to perform this transformation without affecting confidentiality [8]. A standard construction to obtain a PRE scheme from an FHE scheme is described in Gentry's seminal work [21].

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

Following the notation of [32], we define a PRE scheme as a tuple $\mathcal{PRE} = (PG, KG, ReKG, E, D, RE)$ of six procedures. Parameter generation $PG(\cdot)$ computes a set of public parameters related to the security parameter κ. The key generation algorithm $KG(\cdot)$ outputs a key pair (pk, sk). Re-encryption-key generation $ReKG(\cdot)$ takes a secret key sk_i and a public key $pk_{j\neq i}$ and computes a re-encryption key $rk_{i\to j}$. The re-encryption algorithm $RE(\cdot)$ transforms a ciphertext c_i of m encrypted under pk_i into a ciphertext c_j of m such that c_j encrypts m under $pk_{j\neq i}$. $E(\cdot)$ and $D(\cdot)$ are encryption and decryption algorithms, respectively.

C. Secret Sharing

An (n, n) secret sharing scheme enables a party to split some secret $s \in \mathbb{Z}_q$ for a prime q into n parts such that the n parts together represent s [17]. Combining less than n parts reveals nothing about s. We denote a secret-shared value s by $[\![s]\!]$ and the i-th part, called share, by $[\![s]\!]_i$. A secret sharing scheme ensures privacy and correctness. Privacy guarantees that no share leaks any non-trivial information about s. Correctness ensures that s can be reconstructed by combining the n shares. To share a value s, one chooses $n-1$ random numbers uniformly at random from $\mathbb{Z}_q$, i.e., $[\![s]\!]_1, ..., [\![s]\!]_{n-1} \overset{U}{\leftarrow} \mathbb{Z}_q$ and computes the n-th share as follows [33].

$$[\![s]\!]_n = \left[s - \sum_{i=1}^{n-1} [\![s]\!]_i \right]_q \tag{4}$$

D. Distributed Ledger

Distributed ledgers are append-only data structures maintained among a distributed network [25]. Data is represented as transactions. New transactions are distributed through the network and validated by the nodes, which agree on the ledger's state via a distributed consensus procedure. Distributed ledgers can be permissioned, i.e., only parties with permission can join the network, while any party can join a permissionless ledger.

IV. Balance-Verification Protocols

We present two balance-verification protocols for distributed-ledger-based supply-chain traceability systems. To the best of our knowledge, they are the first of their kind that offer public verification for confidential transaction details.

Each protocol consists of two separate subprotocols: one for transactions and one for verification. In the transaction subprotocol, the customer of the respective transaction publishes details about the physical-world transactions on a distributed ledger DL in a privacy-preserving form. These details include the transaction amount. The transaction subprotocol is executed every time the producer P transfers an amount x_i to a customer C_i in the physical world. It does not describe the actual transmission of x_i and instead maintains a distributed-ledger representation of the corresponding real-world activities. The verification subprotocol is executed whenever a verifier V wants to check the producer's honesty.

Being ledger-agnostic, the protocols can be added to existing solutions as they only affect the form in which data is published and the computations that are performed during verification. We leave security and correctness proofs for our protocols to the extended version of this paper.

A. Protocol Π_S – Secret Sharing

Protocol Π_S uses secret sharing (see Section III-C) to ensure confidentiality of the transaction amounts x_i in the sense that the x_i's are blinded by adding large random shares. It is tailored for long-lasting producer-customer relations, e.g., between palm-oil mills and manufacturers of palm-oil-based products, where customers of subsequent transactions are known in advance. Π_S runs in epochs of K transactions. Balance verification can be performed at the end of an epoch as a form of batch verification. The protocol involves a producer, multiple customers, and an unspecified number of verifiers.

We assume a group $\mathbb{Z}_q$ for a prime q and enable the representation of negative values by allocating the upper half of $\mathbb{Z}_q$ for negative values. In a two's-complement manner, this represents the range from $-(\frac{q-1}{2})$ to $(\frac{q-1}{2})$. To prevent overflows, we choose q such that $x_{max} \ll (\frac{q-1}{2})$, which implies $x_i \ll (\frac{q-1}{2})$ for all reasonable x_i. The two subprotocols are depicted in Algorithms 1 and 2, respectively.

Algorithm 1: Transaction Subprotocol of Π_S

```
   Data: $i, x_i, r_{\Sigma_{i-1}}$
   Result: $t_i, r_1, ..., r_K, r_{\Sigma_i}, r_\Sigma$
1  if $i \equiv_K 1$ then
2      $P$ computes:
3          for $1 \le j \le K-1$ do
4              $r_j \overset{U}{\leftarrow} \mathbb{Z}_q$
5          end
6          $r_K = \left[0 - \sum_{j=1}^{K-1} r_j\right]_q$
7  end
8  $P$ sends to $C_i$:
9      $r_i$, $C_{[i+1]_K}$
10 $C_i$ computes and publishes via $DL$:
11     $t_i = [x_i + r_i]_q$
12 $C_i$ computes and sends to $C_{[i+1]_K}$:
13     if $i \equiv_K 1$ then
14         $r_0 \overset{U}{\leftarrow} \mathbb{Z}_q$
15         $r_{\Sigma_i} = [r_0 + r_1]_q$
16     else
17         $r_{\Sigma_i} = \left[r_{\Sigma_{i-1}} + r_i\right]_q$
18     end
19 if $i \equiv_K K$ then
20     $C_1$ computes and publishes via $DL$:
21         $r_\Sigma = [r_{\Sigma_K} - r_0]_q$
22 end
```

1) Transaction Subprotocol: Lines 3-6 are only performed in the first transaction of an epoch, i.e., if $i \equiv_K 1$. Here, P generates K secret shares $r_1 = [\![r]\!]_1, ..., r_K = [\![r]\!]_K$ of a value r. We suggest $r = 0$. They will be used throughout this epoch for additively blinding the transaction amounts. P chooses $K-1$ shares $r_1, ..., r_{K-1}$ uniformly at random from $\mathbb{Z}_q$ and computes the K-th share as $r_K = [0 - \sum_{j=1}^{K-1} r_j]_q$.

In line 9 of the first transaction, P sends r_1 to C_1 together with the identity of the customer of the next transaction. The

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

following transactions of that epoch start with P taking the i-th previously generated share r_i and sending it to C_i in line 9 together with the next customer's identity. In line 11, C_i takes the amount x_i that it received in the physical world. This x_i could be obtained by counting or weighing the real-world delivery. C_i additively blinds x_i by adding the i-th random share r_i. It publishes the resulting sum t_i on the ledger DL.

The customers maintain a rolling sum r_{Σ_i} of the shares r_i, which is passed on among customers. This allows them to keep track of the r_i's used for blinding and later remove them during balance verification. The computation of r_{Σ_i} differs depending on the transaction index i (see lines 13-18). At the beginning of an epoch, i.e., if $i \equiv_K 1$, C_1 chooses r_0 uniformly at random from $\mathbb{Z}_q$ and additively blinds r_1 with r_0. This sum is sent to C_2, the customer of the second transaction. This prevents C_2 from learning r_1 and thus protects x_1. In all later transactions, C_i maintains r_{Σ_i} by adding r_i and sending it to C_{i+1}. C_K, the last customer of the epoch, returns the rolling sum to C_1.

Line 21 is part of the last transaction of an epoch. It removes the initial random value r_0 from the rolling sum. C_1 subtracts r_0 from r_{Σ_K} to obtain the sum r_Σ. Then, C_1 publishes r_Σ via DL. This concludes the transaction subprotocol.

Algorithm 2: Verification Subprotocol of Π_S

```
Data: $t_1, ..., t_n, x_{max}, r_\Sigma$
Result: $\top$ or $\bot$
23 $V$ computes:
24     $\delta = [x_{max} + r_\Sigma - \sum_{i=1}^{n} t_i]_q$
25     if $\delta \leq \frac{q-1}{2}$ then
26         return $\top$
27     else
28         return $\bot$
29     end
```

2) Verification Subprotocol: After an epoch has ended, any verifier V that has access to DL can verify the balance δ. In line 24, V computes δ according to Equation (1) by reading all t_i from DL, adding them, and subtracting their sum from x_{max}. If $r_\Sigma \neq 0$, V also adds r_Σ. The random shares r_i cancel out, resulting in the balance δ. Lines 25-29 output whether P behaved honestly or not. If $\delta > (\frac{q-1}{2})$, the computed balance represents a negative value in two's complement. Therefore, V detected fraud and the verification subprotocol returns $\bot$. Otherwise, it returns $\top$ to indicate correct behavior of the producer. If $\delta = 0$, publication of another t_i for the same x_{max} in the future immediately indicates fraud.

3) Protocol Discussion: The customer-maintained rolling sum in the transaction subprotocol implies that the previous customer remains active until the next transaction starts. Furthermore, the first customer of an epoch actively contributes to the completion of the epoch. This is reasonable in long-lasting producer-customer relations, e.g., between palm-oil mills and manufacturers of palm-oil-based products. Neither the producer nor the customers participate in the verification.

The epoch-oriented design has two opposing effects: On the one hand, it allows verification only once every K transactions, where smaller K allow verification more frequently. On the other hand, it adds a few maintenance steps at the beginning and at the end of an epoch, which have smaller impact on the overall runtime for larger K. We investigate the trade-off between verification frequency and runtime in Section V-B.

The epoch-oriented design of Π_S has another major advantage: Repeated balance verification across multiple epochs can be sped up by caching previous epochs in a memory-efficient way, which minimizes the effect of previous epochs. V can cache the sum $\sum_{i=1}^{n} t_i$ after every epoch. In later verifications, V adds all newly published t_i, causing verification complexity that is constant in the epoch-length. The cached sum could as well be computed via smart contracts and be reflected on DL to share the benefit of caching with other verifiers.

Confidentiality of the transaction amounts is ensured through additive blinding with secret, random shares. The balance can be computed by any party with access to DL at the end of an epoch. If K is sufficiently large, e.g., in the order of hundreds of transactions, publishing the balance once an epoch is reasonable. This balance reveals no single unknown x_i to an adversary that does not corrupt customers that are involved in more than $K-2$ transactions. If the second and the last customer of an epoch are the same or collude, they can infer r_0 and reconstruct r_1 and x_1 of the first transaction. Similarly, if C_{i-1} and C_{i+1} are the same or collude, they can infer r_i and x_i of the intermediate transaction. Such combinations are to be prevented by planning epochs in advance.

B. Protocol Π_{PRE} – FHE and PRE

Protocol Π_{PRE} uses homomorphic encryption and proxy re-encryption (see Sections III-A and III-B) to guarantee confidentiality of the transaction amounts x_i. The transaction amounts remain encrypted throughout the protocol. Unlike Π_S, this protocol does not run in epochs. Hence, verification can be performed after every single transaction. Π_{PRE} is tailored for short-term relations between static producers and volatile customers, e.g., manufacturers and (end) consumers.

In addition to the parties P, C_i, and V, Π_{PRE} involves a re-encryption party R. It is semi-honest and must not collude with any of the other parties. R could be hosted by an NGO or an industry consortium, e.g., RSPO in the palm-oil case. It actively contributes to the verification subprotocol.

We require that P has a verification-result key pair (pk_P, sk_P) of an FHE scheme with plaintext space $\mathcal{M}_P \subset \mathbb{Z}$, plaintext modulus t, and ciphertext space $\mathcal{C}_P$. We require $t \gg x_{max}$, which implies $t \gg x_i$ for all reasonable x_i. Each customer C_i has a key pair (pk_{C_i}, sk_{C_i}) of an asymmetric cryptosystem with ciphertext space $\mathcal{C}_{C_i}$. It sends a re-encryption key $rk_{C_i \to P}$ to R at setup time. This key enables the transformation of ciphertexts under pk_{C_i} into ciphertexts under pk_P. The two subprotocols are depicted in Algorithms 3 and 4, respectively.

1) Transaction Subprotocol: C_i takes the physically received amount x_i, encrypts it under pk_{C_i}, and publishes the resulting ciphertext y_i via DL.

2) Verification Subprotocol: Π_{PRE} allows balance verification after every single transaction. Verification starts in line 4

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3: Transaction Subprotocol of Π_{PRE}

```
Data: $i, x_i, pk_{C_i}$
Result: $y_i$
1 $C_i$ computes and publishes via $DL$:
2     $y_i = E_{pk_{C_i}}(x_i)$
```

Algorithm 4: Verification Subprotocol of Π_{PRE}

```
Data: $y_1, ..., y_n, x_{max}, rk_{C_1 \to P}, ..., rk_{C_n \to P}, pk_P, sk_P$
Result: $\top$ or $\bot$
3 $V$ sends to $R$:
4     $n$
5 $P$ sends to $V$:
6     $sk_P$
7 $R$ computes:
8     for $1 \leq i \leq n$ do
9         $E_{pk_P}(x_i) = RE(y_i, rk_{C_i \to P})$
10    end
11    $r_1, r_2 \overset{U}{\leftarrow} \mathcal{M}_P$ s.t. $0 < r_2 < r_1 \ll |\mathcal{M}_P|$
12 $R$ computes and sends to $V$:
13    $E_{pk_P}(\delta') = E_{pk_P}((x_{max} - \sum_{i=1}^{n} x_i) \cdot r_1 + r_2)$
14 $V$ computes:
15    $\delta' = D_{sk_P}(E_{pk_P}(\delta'))$
16    if $\delta' \geq 0$ then
17        return $\top$
18    else
19        return $\bot$
20    end
```

with V choosing which transaction it wants to verify the balance for, e.g., the most recent transaction. The verifier sends the index n of that transaction to R. This tells R that V wants to verify whether the sum of the first n transaction amounts of P exceeds x_{max}, in other words, if $\delta = x_{max} - \sum_{i=1}^{n} x_i < 0$.

On enquiry in line 6, V obtains the decryption key sk_P from producer P. It is later used to decrypt the output. In lines 8-10, R reads the ciphertexts $E_{pk_{C_1}}(x_1), ..., E_{pk_{C_i}}(x_i), ..., E_{pk_{C_n}}(x_n)$ from DL. These are the encrypted transaction amounts that V specified in line 4. Each of these ciphertexts is then re-encrypted by R. Given $E_{pk_{C_i}}(x_i)$ and the respective re-encryption key $rk_{C_i \to P}$, R re-encrypts each $E_{pk_{C_i}}(x_i)$ into $E_{pk_P}(x_i)$. This way, R obtains the full list of transaction amounts encrypted under pk_P.

In line 11, R chooses two numbers $0 < r_2 < r_1 \ll |\mathcal{M}_P|$ uniformly at random. In line 13, R computes the encrypted balance, as in Equation (1). The encrypted balance is then blinded with r_1 and r_2. Hence, without knowledge of r_1 and r_2, the blinded balance δ' reveals nothing about the actual balance δ except for its sign. This works as follows.

R homomorphically adds (see Equation (2)) the x_i's encrypted as $E_{pk_P}(x_i)$. It negates the resulting encrypted sum by homomorphic multiplication (see Equation (3)) with -1 encrypted as $E_{pk_P}(-1)$. The result is then homomorphically added to the encrypted maximum amount $E_{pk_P}(x_{max})$. This yields the encrypted balance. To prevent verifiers from learning the exact balance, it is blinded by homomorphic multiplication with $E_{pk_P}(r_1)$ and homomorphic addition with $E_{pk_P}(r_2)$. The result is the encrypted, blinded balance $E_{pk_P}(\delta')$, which is sent to V. This can be formalized as follows.

$$\begin{aligned} E_{pk_P}(\delta') &= E_{pk_P}((x_{max} - \sum_{i=1}^{n} x_i) \cdot r_1 + r_2) \\ &= ((E_{pk_P}(x_{max}) \oplus (E_{pk_P}(-1) \odot \bigoplus_{i=1}^{n} E_{pk_P}(x_i))) \\ &\quad \odot E_{pk_P}(r_1)) \oplus E_{pk_P}(r_2) \end{aligned}$$

Depending on the FHE scheme, addition and multiplication by known values could be performed as ciphertext-plaintext operations for the sake of efficiency. This applies to multiplication by -1 to enable subtraction as well as multiplicative and additive blinding with r_1 and r_2.

In line 15, V decrypts $E_{pk_P}(\delta')$ with sk_P and obtains δ'. If δ' is negative, so is δ. Hence, verification returns $\bot$ to indicate fraud. Otherwise, it returns $\top$, indicating correct behavior.

3) Protocol Discussion: Protocol Π_{PRE} involves an additional party R. Both P and R participate in the verification. Customers can drop out after they completed their transactions. Hence, Π_{PRE} is particularly suited for scenarios with static producers and volatile customers, e.g., transactions between manufacturers and (end) consumers.

Π_{PRE} does not run in epochs. Balance verification can be performed at any time. Compared to Π_S, this gives more flexibility on the verifiers' side. For recurring verifications, the verification complexity can be reduced by having the re-encryption party R cache the encrypted (rolling) sum of transaction amounts whenever it performs a verification. In later verifications, R only adds all newly published encrypted amounts to that rolling sum. Repeating this caching on a regular basis, i.e., once every T transactions for a fixed T, ensures constant verification complexity. Unlike Π_S, caching can be performed centrally by R and therefore reduces the complexity at a global scale for all verifiers at once.

Due to the strong security guarantees of FHE schemes and their ability to compute arithmetic functions on ciphertexts, no party learns P's balance. Verifiers only learn whether it is positive or negative. The customers and the re-encryption party R do not learn anything non-trivial about the balance.

V. Evaluation

We first examine the asymptotic round, computational, and communication complexities of Π_S and Π_{PRE}. Then, we investigate their empirical performance and compare their storage requirements to those of Bulletproofs [11].

A. Theoretical Evaluation

Each protocol consists of two subprotocols with a constant number of steps, independent of the number of transactions or participants. Hence, they have constant round complexity.

The computation complexity of each operation differs between the protocols. Π_S consists mainly of modular additions and subtractions, with a complexity depending on the parameter q of the secret sharing scheme. The computational complexity of Π_{PRE} mostly depends on the used FHE scheme and re-encryption functionality. Due to the computational

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

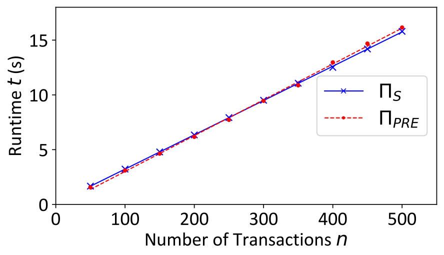

Fig. 2: Transaction Runtime ($K = n$)

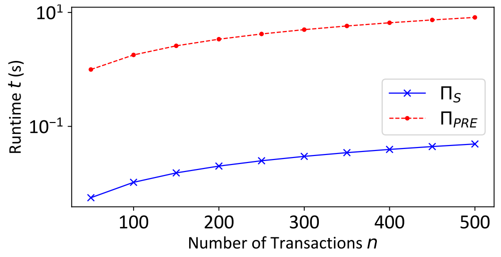

Fig. 3: Verification Runtime ($K = n$)

TABLE I: Computational Complexities

Protocol	**Trans.**	**Verif.**	**Verif. with Caching**
Π_S	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(K)$
Π_{PRE}	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(T)$

overhead of FHE schemes, one can expect that Π_S has better runtime than Π_{PRE}.

Table I shows the asymptotic computational complexity for transaction, verification, and verification with caching. Each protocol consists of a constant number of operations per transaction for both subprotocols, causing $\mathcal{O}(1)$ for transaction and $\mathcal{O}(n)$ for verification. Verification can be optimized. Caching verification results after every epoch in Π_S and after T transactions in Π_{PRE} reduces the verification complexity to $\mathcal{O}(K)$ and $\mathcal{O}(T)$, respectively. Constant K and T cause constant verification complexity, i.e., $\mathcal{O}(1)$.

The epoch-based protocol Π_S performs a constant number of steps at the beginning and at the end of an epoch. These operations influence the overall complexity for small K but should have minor impact for large K.

Each protocol sends a constant number of messages and reads a number of messages linear in n. Caching in intervals of fixed length enables constant communication complexity even for verification. The amount of data stored on the distributed ledger in Π_S and Π_{PRE} depends on the lengths of the random shares and ciphertexts, respectively. We compare these storage requirements for practical scheme parameters to those of Bulletproofs in the empirical evaluation.

B. Empirical Evaluation

To the best of our knowledge, our protocols are the first of their kind that allow public balance verification while ensuring confidentiality of transaction details. The empirical analysis investigates their practicality even for large supply-chain scenarios. It consists of storage-requirement comparisons and two separate runtime experiments. The first experiment investigates the runtime relatively to the number of transactions n. The second focuses on the impact of different epoch lengths K for constant n in Π_S.

1) Experimental Setup: We implemented our protocols in C++ using the libraries Crypto++[7] for big-integer arithmetic and PALISADE[8] for FHE and PRE. Our implementations are parameterized to fit the annual palm oil production of circa 75 metric tons [2]. Π_S uses 512-bit shares to blind 64-bit amounts. Π_{PRE} employs the re-encryption functionalities of the RNS variant of the BFV scheme implemented in PALISADE [23]. We selected the scheme parameters according to the Homomorphic Encryption Standard [3] for a security level of 128 bits. Communication is secured with TLS.

We deployed all protocol participants on cloud instances with 32 cores, $128\,GiB$ of memory, and a $10\,Gbps$ network interface. For lifelike communication, participants were deployed in different data centers distributed across Europe with a distance of several hundred kilometers. We used Multichain[9] as a distributed ledger, which poses as the core of the underlying supply-chain traceability system.

2) Storage Requirements: For Π_S, we distinguish between ordinary transactions and final transactions of epochs. The latter publish additional data and require more storage on the ledger. In final transactions, Π_S publishes a blinded secret and a sum of secret shares, 512 bits each, i.e., 128 bytes in total. Ordinary transactions store a single blinded secret of 64 bytes. Π_{PRE} publishes a ciphertext of 381 kilobytes per transaction.

A single Bulletproof for 64-bit ranges is 688 bytes long [11]. Π_{PRE} requires more storage. For final transactions, Π_S requires 5.3 times less storage than Bulletproofs. For ordinary transactions, Π_S requires 10.8 times less storage. For long epochs of hundreds of transactions, the effect of the final transaction is negligible.

3) Effect of the Number of Transactions n: In the first experiment, we investigated the subprotocols' runtimes relatively to the number of transactions n for each $n \in \{i \cdot 50 | 1 \leq i \leq 10\}$, with 100 runs each. The theoretical analysis implies constant complexity for each transaction and linear (in n) complexity for a verification of n transactions. For the first experiment, the epoch length K in Π_S is equal to the number of written transactions n.

Figure 2 depicts the runtimes of both transaction subprotocols relatively to n. The graph shows linear growth of the runtime. This confirms our theoretical analysis. With a runtime of $15.80\,s$ for 500 transactions, Π_S performs slightly better than Π_{PRE} with $16.15\,s$. Hence, Π_S and Π_{PRE} perform single transactions in $31.60\,ms$ and $32.30\,ms$, respectively.

[7]https://cryptopp.com

[8]https://palisade-crypto.org

[9]https://multichain.com

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

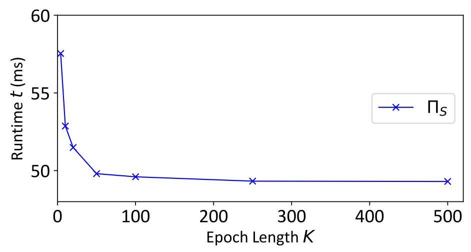

Fig. 4: Verification Runtime of Π_S ($n = 500$)

Figure 3 depicts the verification runtime for n transactions in logarithmic scale. It grows linearly in n for both protocols. While a verification of 500 transactions took $49.30\,ms$ for Π_S, the execution of Π_{PRE} took $8,283.04\,ms$. Hence, Π_S and Π_{PRE} had an average runtime per transaction of $0.097\,ms$ and $16.013\,ms$, respectively.

In summary, the first experiment proves constant complexity for single transactions and linear verification complexity.

4) Effect of the Epoch Length K: The second experiment aims for an understanding of the trade-off between verification frequency and runtime in Π_S. Π_{PRE} is not considered here.

We analyzed the impact of different epoch lengths K on the verification runtime for constant n. We chose $n = 500$ and conducted 100 runs for each $K \in \{10, 20, 50, 100, 250, 500\}$. Figure 4 shows that verification of n transactions takes longer with small K but does not decrease substantially for $K \geq 250$. Thus, for the trade-off between runtime and verification frequency, we deem an epoch-length of $K = 250$ most suitable.

5) The Computation's Impact on the Runtime: Additionally, we analyzed the impact that the computation has on the runtime. Table II shows the ratios between time spent on computation, reading from and writing to the distributed ledger, as well as communication between parties. In all subprotocols, the largest share of the runtime is spent on communication and distributed-ledger operations. The latter is inherent in the underlying supply-chain traceability system. These ratios indicate that a setup with lower network latency and a more efficient distributed ledger will improve the runtimes substantially.

6) Summary: We analyzed our protocols theoretically and empirically. The empirical evaluation confirms the results of the theoretical analysis. Our protocols feature an asymptotic behavior that is linear in the number of transactions and, thus, constant for single transactions. They verify 500 transactions in $8.3\,s$ (Π_{PRE}) and $49.3\,ms$ (Π_S), respectively. Even though Π_S performs substantially better than Π_{PRE}, its communication between customers limits its suitability to scenarios with long-lasting producer-customer relations. As both protocols allow caching of the balance in regular intervals, e.g., once every 250 transactions, they are suitable even for supply-chain scenarios with a very large number of transactions. This proves practicality of both protocols. Most notably, Π_S requires up to 10.8 times less storage on the ledger than Bulletproofs.

TABLE II: Ratios Between Time Spent on Computation, Distributed-Ledger Operations, and Communication

Subprotocol	**Comp.**	**DL**	**Comm.**
Π_S Trans.	$0.013\,\%$	$49.646\,\%$	$50.341\,\%$
Π_S Verif.	$28.222\,\%$	$71.778\,\%$	
Π_{PRE} Trans.	$16.789\,\%$	$83.211\,\%$	
Π_{PRE} Verif.	$8.760\,\%$	$85.876\,\%$	$3.758\,\%$

C. Deployment Considerations and Lessons Learned

The runtimes of our protocols are mostly caused by communication between parties as well as reading from and writing to the distributed ledger. The latter is inherent in any supply-chain traceability system. Our findings indicate that a setup with lower network latency and a more efficient distributed ledger would improve the runtimes substantially in production. Similarly, stronger cloud instances with more cores could take full advantage of the highly parallelizable nature of our protocols, especially for the verification subprotocol of Π_{PRE}.

For evaluation, we used Multichain due to its simplicity and compatibility with the Bitcoin ecosystem. Our protocols are designed to be ledger-agnostic and could perform even better with specialized distributed ledgers. Ledgers that offer more flexibility in ciphertext encoding could further improve efficiency, especially for Π_{PRE}. Hence, future work could investigate potential performance gains from using other ledgers.

The BFV implementation that we used for evaluation only offers a plaintext space of ≤ 59 bits. This limits the precision of inputs and the size of random blinding values. Other libraries or implementations with a larger plaintext space could enable larger inputs while still allowing sufficiently large random blinding values. The latter ensures confidentiality. Moreover, future FHE schemes with more efficient PRE operations could further improve the performance of Π_{PRE}.

VI. Conclusion

Sustainable sourcing and production are major challenges that result from a growing world population and rapid climate change. Supply-chain traceability systems with balance-verification capabilities and public verifiability of compliance can help to enforce sustainability policies like production limits. However, a lack of confidentiality hinders adoption.

We propose two independent protocols for privacy-preserving balance verification over distributed ledgers: Π_S and Π_{PRE}. They are designed for different volatility levels of producer-customer relations to cover a large variety of supply chains and stages of production. We demonstrate their practicality in an extensive theoretical and empirical evaluation.

Each protocol has constant transaction overhead and verification complexity that is linear in the number of transactions. Being well suited for caching at regular intervals, both protocols allow constant verification time. Most notably, Π_S requires 10.8 times less storage on the distributed ledger than recent storage-efficient non-interactive zero-knowledge proofs.

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.

Hence, our protocols prove to be perfectly practical in their respective target scenarios, even for complex supply chains. Being ledger-agnostic, they can be used as stand-alone solutions on any distributed ledger. Alternatively, they can easily be added as a confidentiality layer to existing distributed-ledger-based supply-chain traceability systems. This enables verification of compliance with sustainability policies while protecting confidential supply-chain data.

References

[1] T. Agrawal. "Contribution to development of a secured traceability system for textile and clothing supply chain". PhD thesis. University of Borås, 2019.

[2] United States Department of Agriculture, Foreign Agriculture Service, Crop Explorer. *Oil, Palm 2020*. https://ipad.fas.usda.gov/cropexplorer/cropview/commodityView.aspx?cropid=4243000. 2020.

[3] M. Albrecht et al. *Homomorphic Encryption Security Standard*. Tech. rep. Toronto, Canada: HomomorphicEncryption.org, 2018.

[4] K. Becher, J. A. G. Lagodzinski, and T. Strufe. "Privacy-Preserving Public Verification of Ethical Cobalt Sourcing". In: *IEEE TrustCom'20*. 2020.

[5] E. Ben-Sasson et al. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive. https://eprint.iacr.org/2018/046. 2018.

[6] E. Ben-Sasson et al. "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge". In: *Advances in Cryptology – CRYPTO'13*. 2013.

[7] E. Ben-Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. 2014.

[8] M. Blaze, G. Bleumer, and M. Strauss. "Divertible protocols and atomic proxy cryptography". In: *Advances in Cryptology – EUROCRYPT'98*. 1998.

[9] S. Bowe, A. Gabizon, and I. Miers. *Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model*. Cryptology ePrint Archive. https://eprint.iacr.org/2017/1050. 2017.

[10] Z. Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology – CRYPTO'12*. 2012.

[11] B. Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. 2018.

[12] B. Bünz et al. *Zether: Towards Privacy in a Smart Contract World*. Cryptology ePrint Archive. https://eprint.iacr.org/2019/191. 2019.

[13] M. Caro et al. "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation". In: *IEEE IOT Tuscany'18*. 2018.

[14] E. Cecchetti et al. "Solidus: Confidential Distributed Ledger Transactions via PVORM". In: *CCS'17*. 2017.

[15] B. Cook. *Blockchain: Transforming Seafood Supply Chain Traceability*. https://dj8xp7a0ejkvv.cloudfront.net/downloads/draft_blockchain_report_1_4_1.pdf. WWF-New Zealand, 2018.

[16] H. Corrigan-Gibbs and D. Boneh. "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics". In: *USENIX NSDI'17*. 2017.

[17] R. Cramer, I. Damgård, and J. Nielsen. *Secure Multiparty Computation and Secret Sharing*. 1st ed. Cambridge University Press, 2015.

[18] G. Dagher et al. "Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges". In: *CCS'15*. 2015.

[19] J. Fan and F. Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive. https://eprint.iacr.org/2012/144. 2012.

[20] C. Garman, M. Green, and I. Miers. "Accountable Privacy for Decentralized Anonymous Payments". In: *Financial Cryptography and Data Security 2016*. 2017.

[21] C. Gentry. "A fully homomorphic encryption scheme". PhD thesis. Stanford University, 2009.

[22] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". In: *ACM STOC'85*. 1985.

[23] S. Halevi, Y. Polyakov, and V. Shoup. "An Improved RNS Variant of the BFV Homomorphic Encryption Scheme". In: *CT-RSA'19*. 2019.

[24] H. Kang et al. "FabZK: Supporting Privacy-Preserving, Auditable Smart Contracts in Hyperledger Fabric". In: *2019 IEEE/IFIP DSN*. 2019.

[25] N. Kannengiesser et al. "What Does Not Fit Can be Made to Fit! Trade-Offs in Distributed Ledger Technology Designs". In: *HICCS-52*. 2019.

[26] O. Lai, C. Tan, and C. Akoh, eds. *Palm Oil: Production, Processing, Characterization, and Uses*. 1st ed. Academic Press and AOCS Press, 2015.

[27] Y. Lindell. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Springer, 2017.

[28] N. Narula, W. Vasquez, and M. Virza. "Zkledger: Privacy-preserving Auditing for Distributed Ledgers". In: *USENIX NSDI'18*. 2018.

[29] World Wildlife Fund for Nature. *WWF's position on the adopted 2018 RSPO Principles and Criteria*. https://wwf.panda.org/?337932. 2018.

[30] T. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology – CRYPTO'91*. 1992.

[31] A. Poelstra et al. "Confidential Assets". In: *Financial Cryptography and Data Security 2018 International Workshops, BITCOIN, VOTING, and WTSC*. 2018.

[32] Y. Polyakov et al. "Fast Proxy Re-Encryption for Publish/Subscribe Systems". In: *ACM TOPS* (2017).

[33] D. Stinson. *Cryptography: Theory and Practice*. 3rd ed. Chapman & Hall/CRC, 2006.

Authorized licensed use limited to: KIT Library. Downloaded on October 05,2023 at 15:54:12 UTC from IEEE Xplore. Restrictions apply.