Trust in a Decentralized World: Data Governance from Faithful, Private, Verifiable, and Traceable Data Feeds

Abstract—**Blockchain technology autonomously executes smart contracts that require external data to facilitate specific applications, underscoring the necessity for Authenticated Data Feeds (ADF). Existing solutions fall short in providing genuine authentication of data, lack private and verifiable computations across multiple data sources, and overlook data traceability, rendering current systems inadequate for complex applications. We present WuKong (WK), a data governance system that offers authenticated, privately verifiable, and traceable data feeds. WK enables a server to collect faithful data through an oracle committee and to prove computation correctness in zero-knowledge proofs, and empowers legal entities to trace a leakage source conditionally. We formally define and prove the security of WK in the universal composability framework. We implement three applications that seamlessly integrate with WK. Experimental results indicate that WK effectively liberates sensitive data from distributed, untrusted, and anonymous providers, making it accessible to various services and establishing trust in a decentralized world.**

1. Introduction

1.1. Background

Blockchain (BC) enables untrusted parties to agree on data and activities via transactions among the parties running a consensus mechanism [1], [2]. Although it is first introduced in Bitcoin [1], BC has been used in a plethora of applications far beyond finance, including insurance [3], [4], identity management [5], [6], supply chain management [7], [8], and digital forensics [9], [10]. Providing BC-based services is a considerable technical route for establishing *trust in a decentralized world*. Among several notable services, Ethereum [11], [12] is one of the most valuable BCs in market capitalization. A distinctive advantage of Ethereum is the utilization of a Smart Contract (SC). It is a piece of self-executing codes specified by Ethereum Virtual Machine (EVM) assemblies to perform predefined logic and manage SC-triggering transactions [13].

To fuel the SCs toward continuously reliable running, *Authenticated Data Feed (ADF)* is of vital importance. Even though data is kept immutable, verifiable, and private after being stored in the BC, ADF stands as the first priority of *data governance* for BC while many existing works overlook this issue. A few efforts in this line of research include Town Crier (TC) [14], PDFS [15], and DECO [16]. However, TC posits on a shakeable assumption that there are trusted data sources, PDFS assumes that data feeds are produced by a trusted content provider, and Deco only proves to the verifier that data came from a TLS server.

Besides the issue of guaranteeing ADF, SC faces a significant privacy concern by nature since the *data and computations* in a SC are propagated across the BC network and publicly visible [17]. To address such concerns, numerous solutions have been proposed, including zkSNARK-assisted transactional privacy-preserving SC system Hawk [17], [18], [19], Non-Interactive Zero-Knowledge (NIZK)-based data-preserving language zkay [20], Σ-Bullets-based payment-preserving SC protocol Zether [21], [22], [23], and oblivious Merkle tree and NIZK-based data and identity-preserving SC system Zapper [24]. Another class of works builds over Private Smart Contract (PSC) [25] to keep transaction inputs and contract states secure, such as Ekiden [26], PrivacyGuard [27], Cloak [28], POSE [29], Nereus [30], and DECLOAK [31], where the SC is running inside a Trusted Execution Environment (TEE), e.g., Intel SGX enclave [32], [33], [34].

Last but not the least, in scenarios like data trading [35], [36] and digital forensics [10], [37], the *ownership and proper usage of data* matter greatly when data buyers and users may misuse the data or leak the data to the public. For instance, it is reported that UK police forces accidentally shared details of crime victims, suspects, and witnesses [38]. Three hundred police officers and staff were caught misusing work computers including some who passed information to criminals [39]. Existing work has paid attention to this issue but only considered some specific scenario, such as cryptocurrency [40] and supply chain [41]. We emphasize that data traceability is critical to dispute resolution that suits a rich range of services.

1.2. Motivations

Given these observations on different phases of *data governance* in a BC, we come to three motivations: **M1. Data from untrusted data providers**. Data sources cannot be trusted all the time and external data from a trusted server may still be fraudulent or deceitful [42]. More useful data is likely to be crowdsourced by distributed data providers, e.g., humans and sensors, which are unreliable and probably anonymous. Unreliable means their data needs a proper screening before entering the BC. Anonymity gives them the guts to arbitrarily submit junk data. **M2. Verifiable**

Computation (VC) under untrusted servers. Outsourcing data on a server causes loss of control to data providers. The untrustworthy nature of the server will raise reasonable suspicions among data users. This, combined with the fact that SCs have inevitable bugs and attacks affecting the data computation [43], [44], [45], [46], [47], necessitates retaining data control and privately proving the computation correctness to users. Note that secure searchable encryption is an interesting problem beyond the scope of this paper. **M3. Traceability to anonymous data users**. Malicious data users may claim ownership of requested data or share it (e.g., a picture of a crime scene) with illegal users. Although we cannot control what the users will do with the data once they get it, it is imperative to exert a deterrent force on such users by maintaining a tracing capability.

1.3. WuKong at First Sight

To address these problems, we present *WuKong (WK)*[1], *a data governance system with faithful, private, verifiable, and traceable data feeds*. The technical crux of is to govern the full life cycle of data flows among entities in a decentralized world.

Our solution to M1 is motivated by Oracle-based Conditional Payment (ObCP) [48]. It attests to an outcome of a real-world event for two mutually distrusted entities [48]. Such oracles are already adopted by Ethereum-based DeFi [49] to input data to the BC [50], [51], [52]. We recruit a committee of n decentralized oracles distinguishing faithful data from falsified data, guaranteeing *Faithful Data Feeds (FDF)*.

For M2, we set up a WK server harmonizing TEE and BC to store screened data and respond to tracing requests, bridging providers and users. Particularly, we design a WK PSC $\mathcal{C}_{\text{psc}}$ to offload the data computation from the BC to an off-chain TEE on the WK server for the sake of privacy and efficiency [53], i.e., decoupling SC's operations from the underlying BC [54]. Since the SC has to be updated, we deploy an EVM inside the TEE, supporting BC's inner logic (state/token transition), to facilitate SC's iterations and avoid complex update configurations when no EVM is deployed.

For M3, we require the data users to hand in an identifier, e.g., a secret key, to the WK server, which watermarks the identifier into the requested data. In the meantime, we hide the identifier from the untrusted server. When the watermarked data is disclosed, $\mathcal{C}_{\text{psc}}$ can extract the identifier from it and find the corresponding data user, i.e., data traitor.

1.4. Technical Challenges

C1. Data governance on the full life circle of crowdsensed data in a decentralized environment. We focus on three phases of decentralized data sharing: data collection, data computation, and data traceability. Unlike data trading [35], data sharing has no "judge" to mediate during disagreement. **C1.1**. A naïve solution is deploying some oracles to authenticate-and-sign the data and later verify signatures one by one. But it incurs heavy computations and time costs. The Dynamic Threshold Public-Key Encryption (DTPKE) [55] is a candidate for the t participating oracles to co-decrypt the same data from data providers after authenticating it. However, its prohibitive costs of sophisticated cryptography (e.g., bilinear pairings and laborious zero-knowledge proofs) will hamper the real-world adoption of WK. **C1.2**. Privacy-preserving proofs on authenticated data allow a server to prove computations over data to third parties in a correct and private way [56]. But it leaks data to the server and only applies to data of one data provider. **C1.3**. A typical approach of user tracing is opening a group signature [57], which assumes a trusted party and is time-consuming. The other approach is embedding and extracting of a watermark. Both of them indicate that traceability is a powerful function that we believe must be conditionally invoked.

C2. Formal security for WK. The WK system runs in an adversarial environment where adversaries will sabotage the services. Without proper formal definitions of adversaries and goals, security will be too ambiguous to analyze. It consists of several entities with complex interactions and different security assumptions, which complicates the security framework. Besides, we construct WK upon cryptographic primitives and blockchain techniques. Although we seamlessly integrate them into the TEE-assisted BC framework, still formality is crucial to capture the security precisely.

C3. Executing complicated computations in PSC with limited computing capabilities. The solutions to the three problems call for utilizing complex cryptographic primitives within PSC. However, both TEE and SC face computation restrictions. TEE is constrained by its limited processing power and inability to efficiently handle highly parallel tasks. SC restricts the operation complexity to ensure consensus and prevent excessive gas costs. Hence, PSC cannot directly process computationally heavy tasks due to these limitations. Finally, a WK server running a PSC as an intermediary between data providers and data users could become a severe bottleneck due to such a contradiction.

1.5. Technical Overview

Constructing a system addressing the three challenges holistically could pave the way towards the envisioned "trust in a decentralized world". To tackle with **C1.1**, we propose an 1-t-1 data authentication protocol $\mathsf{Prot}_{\mathsf{col}}$ among data provider $\mathcal{DP}$, t oracles $\{\mathcal{O}_i\}_{i=1}^{t}$, and server $\mathcal{S}$. Specifically, a $\mathcal{DP}$ submits at least t ciphertexts of the same data d to $\{\mathcal{O}_i\}_{i=1}^{t}$ using public key encryption. We leverage Accountable Threshold Signature (ATS) [58], [59], [60], [61] to await at least n $(< N)$ oracles to authenticate d and sign d to obtain t signature shares $\{\sigma_j\}_{j=1}^{t}$. Next, $\mathcal{S}$ receives t pairs of ciphertexts and signatures $\{(c_i, \sigma_j)\}_{j=1}^{t}$ from the oracles and combines them to generate a complete signature

1. Inspired by a character named Sun Wukong in the Chinese fantasy novel *Journey To the West* in 1592, who can recognize monsters and demons with his fiery eyes and golden pupils.

σ on d. The c and σ will be stored on $\mathcal{S}$' storage $\mathcal{ST}$ and further recorded on BC.

To rise to **C1.2**, we introduce a three-party query and response protocol $\mathsf{Prot}_{\mathsf{que}}$ among data user, BC, and server. It moves the majority of transactions off-chain, speeding up contract execution and minimizing the costly interactions with the BC [29]. It processes the result computation and proof generation inside the enclave, securing all the data. Specifically, the server $\mathcal{S}$ relays a data request via a proxy $\mathcal{P}$ to the PSC inside of the TEE, which searches for data in the storage $\mathcal{ST}$ for data via metadata. We design Universal Homomorphic Signatures for Non-deterministic Computation (UHSNC) that allows the PSC to securely compute upon matched data from different data providers and generate a computation result and a correctness proof via two stages. Here, we do not consider access control [62], [63], which is sometimes correlated to data sharing, for the submitted data since we aim to serve a broad range of services while access control is more suitable for a targeted application within a consortium network [10].

To deal with **C1.3**, we design an 1-t-1 tracing protocol $\mathsf{Prot}_{\mathsf{tra}}$ among data provider $\mathcal{DP}$, t oracles $\{\mathcal{O}_i\}_{i=1}^{t}$, and server $\mathcal{S}$ corresponding to the 1-t-1 data authentication protocol. Particularly, we explore a robust watermarking algorithm [64] to allow the TEE to verifiably embed user's private key sk into requested data without leaking sk to $\mathcal{S}$. To trace from watermarked data wd, a $\mathcal{DP}$ submits a tracing request to awake at least one of the original signing oracles and efficiently locate the previously signed data wd via fine-tuned Location-Sensitive Hashing (LSH) [65], [66]. Specifically, $\mathcal{DU}$ has to prove to t oracles that wd has been authenticated before by asking them to check whether "wd is equal to d". The reason is that wd is different from d after embedding sk, so we pursue the acknowledgment of $wd = d$ by checking the LSH values of wd and d, reducing the impact of the watermark on the equality test. If the request passes the verification, the witness $\mathcal{O}$ encrypts the request, generates a signature, and sends them to $\mathcal{S}$, which relays it to the PSC to extract sk from wd. The computed sk will be securely sent to a forensics authority.

To resolve **C2**, we rigorously model WK in the Universal Composability (UC) framework [17], [67], [68], [69], [70], to acquire a model spanning BC and TEE. We formally prove that WK achieves (1) *data faithfulness* – informally WK only accepts the faithful data. (2) *data privacy* – informally adversaries cannot access the plaintext data. (3) *verifiable computation* – informally WK efficiently proves result correctness to data users. (4) *leakage traceability* – informally WK accurately traces to malicious data users.

To overcome **C3**, we fitst implement a proxy $\mathcal{P}$ to allow the enclave to securely access external data through a trusted pathway that handles the data communication and verification work, relieving the computational pressure on the TEE. By offloading the communication and verification work, we mitigate the performance limitations due to TEE's restricted processing capabilities. Second, we customize the Ethereum Virtual Machine (EVM). In specific, we implement complex operations in low-level and efficient languages (Rust and C++), which are compiled and added to the EVM as precompiled contracts. By doing so, we significantly reduce the computational complexity and improve performance of **PSC**, solving the issue of handling computationally heavy tasks. We also optimize PSC codes to minimize costs by reducing duplicate computations and refining the storage structure.

Our contributions are:

- We propose WK to establish valuable trust among untrustworthy and anonymous entities in a decentralized world. It carefully incorporates several security mechanisms to govern crowdsensed data from collection, to request, to tracing.
- We formally define the security of WK in the UC framework with functionalities to represent both on-chain and off-chain components. We formally prove four security properties, i.e., data faithfulness, data privacy, verifiable computation, and leakage traceability.
- We implement a prototype of WK using Intel SGX2 as TEE and Ethereum as BC. We explore three WK applications to show its ability to support a wide variety of services. Results from extensive experiments demonstrate the feasibility and efficiency of WK.

2. Related Work

In this section, we review some related work that made a great effort in data governance and laid a solid foundation for us to move ahead, namely Town Crier, PDFS, and Deco.

Town Crier is an ADF system that addresses critical barriers to the adoption of decentralized smart contracts [14]. It combines an Ethereum smart-contract front end and an SGX-based trusted hardware back end to communicate with HTTPS-enabled websites, serve source-authenticated data to smart contracts, and allow private data requests with encrypted parameters. Town Crier also formally states and proves the security in the UC framework. It offers a practical means to address the lack of ADF in blockchains and paves the way for formally proving security when incorporating SGX into a security system.

PDFS is a data feed service for smart contracts that aims to fill the gap between oracle solutions and transport-layer authentication [15]. It also allows data providers to create an authoritative contract that enables its owner to update it, other contracts to verify whether the provider indeed produced the data, and data users to make censorship-evident queries for the specific data. A data user creates a relying contract to verify an authoritative contract, associate its location as an oracle, and call the authoritative contract's membership verification method before using the data.

Deco is a provably secure decentralized oracle for Transport Layer Security (TLS). It is source-agnostic and supports any website running standard TLS while working without trusted hardware or server-side modifications. Deco allows a data provider to prove that a piece of data accessed via TLS came from a particular website and selectively prove statements about such data in zero-knowledge, protecting

TABLE 1. COMPARISON WITH EXISTING SCHEMES

Scheme	Data Provider	Data User	Crow. Data	Data Privacy	Data Faithfulness	Computation Verifiability	Leakage Traceability	Oracle	BC	TEE	PSC
Town Crier [14]	◐	●	◐	●	◐	○	○	◐	●	●	◐
PDFS [15]	◐	●	○	◐	◐	○	○	◐	●	○	○
Deco [16]	◐	●	○	●	◐	○	○	◐	●	○	○
WK	●	●	●	●	●	●	●	●	●	●	●

● Full support ◐ Partial support ○ No support. Crow.: Crowdsensed.

data confidentiality. In Deco, the oracles are entities that prove the provenance and properties of online data and anyone can become an oracle for any website.

We summarize the comparison with related work in Table 1. Although promising, the state-of-the-art did not address the data faithfulness, computation verifiability, or leakage traceability. Instead, WK recruits a group of special oracles to screen the submitted data coming from random individuals and websites. It delicately designs a server to collect external data and process the data with confidentiality, integrity, and verification. Once some previously requested data is leaked from a malicious data user, the WK system can accurately trace the traitor.

3. Problem Statement

3.1. System Model

The system model of WK includes five main entities: data provider $\mathcal{DP}$, oracle $\mathcal{O}$, WK server $\mathcal{S}$, data user $\mathcal{DU}$, and blockchain $\mathcal{BC}$. An architectural schematic of WK with interactions among the five components is depicted in Figure 1.

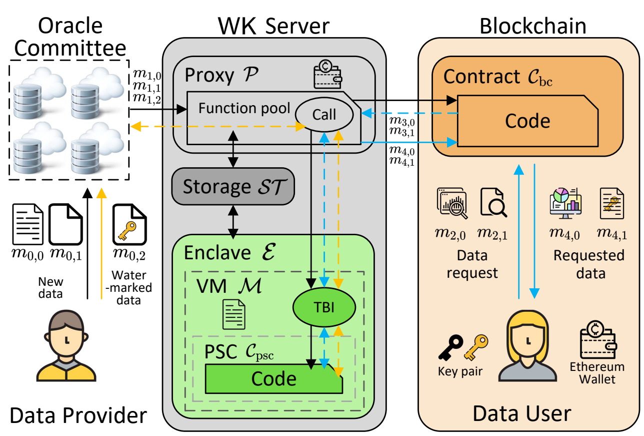

Figure 1. Basic WK Architecture. (Trusted components are depicted in green.)

Data Provider $\mathcal{DP}$ is the data source of the WK system. They can be an individual operating a computer and a website. The $\mathcal{DP}$ voluntarily submits data d to the system. The data is divided into computable data com and independent data ind by its type A_1, and public data pub and private data pri by its sensitivity A_2. Each piece of data during submission is attached with a timestamp ts. Finally, the underlying plaintext message of a $\mathcal{DP}$ is $m_{0,0} = (0, d, \sigma^{\text{vc}}, A_1, A_2, ts)$, where σ^{vc} is a signature on d used for VC, or $\mathcal{DP}$ is $m_{0,1} = (1, d, A_1, A_2, ts)$. To promote posting faithful data, we can utilize an incentive mechanism with an updatable reputation. In addition, we consider the one who provides some watermarked data wd to trace a traitor as a data provider as well. Their submitted plaintext message is $m_{0,2} = (2, wd, A_1, A_2, ts)^2$

Oracle $\mathcal{O}$ is a server run by a legal organization or institute that has professional knowledge and skills to authenticate and verify data. We assume that there is an oracle committee $\mathcal{C}_{\text{auth}} = \{\mathcal{O}_1, \dots, \mathcal{O}_n\}$ working as the line of defense before external data enters the blockchain. When a $\mathcal{DP}$ submits d to an $\mathcal{O}$, the $\mathcal{O}$ authenticates its faithfulness. If the authentication passes, $\mathcal{O}$ stores d in its local dataset. Meanwhile, the $\mathcal{O}_i$ signs d to obtain a signature share σ_i. In this way, the signing oracles make an endorsement for d with a witness message $m_{1,0} = (0, d, \sigma^{\text{vc}}, A_1, A_2, ts, \sigma_i^{\text{ats}})$ or $m_{1,1} = (1, d, A_1, A_2, ts, \sigma_i^{\text{ats}})$, both of which have a signature σ_i on the encrypted version, and notifies $\mathcal{S}$.

For a watermarked data wd from a special data provider, $\mathcal{O}_i$ first searches the local dataset for potential matches and then checks whether the original version of wd has been authenticated by itself before. If so, $\mathcal{O}_i$ signs wd and notifies $\mathcal{S}$ with a message $m_{1,2} = (2, wd, A_1, A_2, ts, \sigma_i^{\text{ats}})$ with σ_i for further key extraction to be discussed later. We define a functionality $\mathcal{F}_{\text{O}_i}$ in Figure 2.

WK Server $\mathcal{S}$ is a powerful computing and communication entity that ingests data feeds from data providers via the oracle committee and fulfills data requests from data users via the blockchain. It could be run by a commercial corporation offering paid data services to general data users or a professional institute managing partially sensitive data within limits. We assume that there are multiple servers in operation. Specifically, $\mathcal{S}$ consists of three components:

- Proxy $\mathcal{P}$. For newly submitted data d, $\mathcal{P}$ uses Call in a function pool to communicate with an enclave $\mathcal{E}$ to store d in $\mathcal{ST}$ and uploads a collect transaction Tx^{col} to $\mathcal{BC}$. For a data request $m_{3,b}$, $\mathcal{BC}$ interacts with $\mathcal{S}$ to respond to a data user. $\mathcal{S}$ uploads Tx^{res} or Tx^{tra} to $\mathcal{BC}$ through a wallet $\mathcal{W}$.
- Enclave $\mathcal{E}$ is equipped with an EVM $\mathcal{M}$ that executes a PSC $\mathcal{C}_{\text{psc}}$. It has two enclave-generated key pairs

2. A_1 and A_2 are not necessary here, but we retain them for interface reusability in programming.

$\mathcal{F}_{\mathbf{O}_i}$**: abstraction for oracle** $\mathcal{O}_i$

Embed: sk_i^{ats}
$\mathsf{code}_{\mathcal{O}_i}$ has three entry points:
Initialize: On receive init from $\mathcal{C}_{\text{auth}}$:
  $\mathsf{outp} := \mathsf{code}_{\mathcal{O}_i}.\mathsf{Initialize}()$
  Output outp
Collect: On receive $\mathsf{Msg}_1^{\text{col}}$ from $\mathcal{DP}$:
  Process $\mathsf{Msg}_1^{\text{col}}$ to obtain $m_{0,b}$ $(b \in \{0, 1\})$
  If $m_{0,b}$ is faithful
    $\sigma_i^{\text{ats}} = \Gamma.\mathsf{Sign}(sk_i^{\text{ats}}, m_{0,b}, \mathcal{S})$//$\mathcal{S}$: signing group
    $c = \Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{0,b}||\sigma_i^{\text{ats}})$
    $\sigma_i = \Pi.\mathsf{Sign}(sk_i^{\text{sig}}, c)$
    Output (c, σ_i)
Trace: On receive $m_{1,2}$ from $\mathcal{DP}$:
  $\mathsf{outp} := \mathcal{C}_{\text{psc}}.\mathsf{Trace}(m_{1,2})$
  Output (c, σ_i)

Figure 2. Formal Abstraction for Oracle Execution.

$(pk_{\mathcal{E}}^{\text{sig}}, sk_{\mathcal{E}}^{\text{sig}})$ and $(pk_{\mathcal{E}}^{\text{enc}}, sk_{\mathcal{E}}^{\text{enc}})$. Its main task is to decrypt ciphertexts to feed plaintexts to $\mathcal{C}_{\mathsf{psc}}$ and signs $\mathcal{C}_{\mathsf{psc}}$'s output.

• PSC $\mathcal{C}_{\mathsf{psc}}$ is a smart contract deployed within $\mathcal{E}$. It has a signing key sk_{psc} for generating a correctness proof. For submitted data d, $\mathcal{C}_{\mathsf{psc}}$ combines its signature shares and verifies the complete ATS signature ,and returns $m_{2,0} = (d, \sigma^{\text{vc}}, A_1, A_2, ts, \sigma^{\text{ats}})$ or $m_{2,1} = (d, A_1, A_2, ts, \sigma^{\text{ats}})$ to $\mathcal{E}$. For a data request $m_{3,0} = (A_1, A_2, \mathsf{f}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$, $\mathcal{C}_{\text{psc}}$ searches an index $\mathcal{I}$ to retrieve data from $\mathcal{ST}$ and computes a result and correctness proofs based on matched data, a function f, a signing key sk_{psc}, and a public key vk_{psc}. For a data request $m_{3,1} = (A_1, A_2, pid, \sigma_{pid}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$, $\mathcal{C}_{\text{psc}}$ embeds a private key into the requested data. Besides, $\mathcal{C}_{\text{psc}}$ tracks from watermarked data by searching $\mathcal{I}$, checking equality, and extracting a private key, which is encrypted before delivering to a target authority, i.e., a court.

• Storage $\mathcal{ST}$ is the storage space inside of $\mathcal{S}$. It is invoked when $\mathcal{E}$ stores data and index into it, and assists $\mathcal{C}_{\text{psc}}$ in retrieving data for a data request.

We adopt a formal abstraction of Intel SGX2 following the UC and generalized UC (GUC) paradigms [14], [67], [68], [69]. Specifically, we use a global UC functionality $\mathcal{F}_{\text{sgx2}}[\mathcal{P}, \mathcal{E}]$ to denote an SGX2 functionality with a signature scheme Π. As described in Figure 3, upon initialization, $\mathcal{F}$ runs $\mathsf{outp} := \mathcal{E}.\mathsf{Initialize}()$ and attests to $\mathcal{E}$'s code $\mathsf{code}_{\mathcal{E}}$ as well as outp. Upon a Process call with encrypted messages $\mathsf{Msg}_2^{\text{col}}$ from $\mathcal{P}$, $\mathcal{F}$ first preprocesses it to decide which of Collect and Trace to call, and outputs a result $\mathsf{Collect}(m_{1,b})$ or $\mathsf{Trace}(m_{1,2})$. Upon a Process call with an encrypted message $\mathsf{Msg}_1^{\text{req}}$ from $\mathcal{BC}$, $\mathcal{F}$ outputs a result $\mathsf{Respond}(m_{3,b})$.

Data User $\mathcal{DU}$. The $\mathcal{DU}$ is the data requester of the WK system and holds a pair of keys $(pk_{\text{du}}, sk_{\text{du}})$. They can be an ordinary user asking for independent data and a professional expecting a statistical result based on computable data. The $\mathcal{DU}$ uses $\mathcal{E}$'s public key $pk_{\mathcal{E}}$ to attest that $\mathcal{E}$ is indeed

$\mathcal{F}_{\mathbf{sgx2}}[\mathcal{P}, \mathcal{E}]$**: abstraction for SGX2**

Embed: $sk_{\mathcal{E}}^{\text{sig}}$
$\mathsf{code}_{\mathcal{E}}$ has six entry points:
Initialize: On receive init from $\mathcal{P}$:
  $\mathsf{outp} := \mathsf{code}_{\mathcal{E}}.\mathsf{Initialize}()$
  $\sigma_{\mathsf{att}} := \Pi_{\mathsf{att}}.\mathsf{Sign}(sk_{\mathcal{E}}^{\text{att}}, (\mathsf{code}_{\mathcal{E}}, \mathsf{outp}))$//Intel EDIP
  Output $(\mathsf{outp}, \sigma_{\mathsf{att}})$
PreProcess: On receive $\mathsf{Msg}_3^{\text{col}}/\mathsf{Msg}_3^{\text{tra}}$ from $\mathcal{P}$:
  Verify, process $\mathsf{Msg}_3^{\text{col}}/\mathsf{Msg}_3^{\text{tra}}$ to obtain $m_{1,b}/m_{1,2}$
  Call Collect or Trace
Collect: On receive $m_{1,b}$ $(b \in \{0, 1\})$:
  $\mathsf{outp} := \mathcal{C}_{\text{psc}}.\mathsf{Collect}(m_{1,b})$
  Output $(\mathsf{H}(\mathsf{outp}), \sigma_{\mathcal{E}})$
Trace: On receive $m_{1,2}$:
  $\mathsf{outp} := \mathcal{C}_{\text{psc}}.\mathsf{Trace}(m_{1,2})$
  Output $(\Sigma.\mathsf{Enc}(pk_{\text{tag}}, \mathsf{outp}), \sigma_{\mathcal{E}})$
Respond: On receive $(\mathsf{Msg}_2^{\text{req}})$ $(b \in \{0, 1\})$ from $\mathcal{P}$:
  Decrypt $\mathsf{Msg}_2^{\text{req}}$ to obtain $m_{3,b}$
  $\mathsf{outp} := \mathcal{C}_{\text{psc}}.\mathsf{Respond}(m_{3,b})$
  Output $(uuid, \Sigma.\mathsf{Enc}(pk_{\text{du}}, \mathsf{outp}), \sigma_{\mathcal{E}})$

Figure 3. Formal Abstraction for SGX2 Execution.

executing correct codes in an SGX2 enclave. Next, the $\mathcal{DU}$ requests data by submitting a data request Tx^{req} with $m_{3,b}$. Here, f is a public function for non-deterministic computations, σ_{pid} is a signature on pseudo identity pid produced by an authority, and pid is only used for watermarking.

Blockchain $\mathcal{BC}$. The $\mathcal{BC}$ is an underlying communication infrastructure that offers a service end to $\mathcal{DU}$s and bridges them with $\mathcal{S}$. The type of $\mathcal{BC}$ depends on the specific scenario. For instance, it could be public BC for a publicly accessible database, consortium BC among several untrusted-but-collaborating institutions, and private BC within a corporation. $\mathcal{BC}$ deploys a smart contract $\mathcal{C}_{\text{bc}}$ to respond to data requests, working as the blockchain front end. For a data request Tx^{req}, $\mathcal{C}_{\text{bc}}$ generates a universally unique identifier $uuid$ and stores $\mathrm{Msg}_1^{\text{req}}$ in a callback pool for $\mathcal{P}$'s retrieval. The underlying plaintext is $(uuid, m_{3,0}) = (uuid, A_1, A_2, \mathsf{f}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$ or $(uuid, m_{3,1}) = (uuid, A_1, A_2, pid, \sigma_{pid}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$. Receiving a respond transaction Tx^{res} with $m_{4,0} = (f, \sigma^{\text{vc}})$ or $m_{4,1} = wd$ from $\mathcal{S}$ underneath, $\mathcal{C}_{\text{bc}}$ stores the ciphertext and signature in a callback pool for $\mathcal{DU}$'s retrieval.

We also define abstractions for off-chain Trusted Computing Base (TCB) and on-chain TCB in Fig. 4.

3.2. Security Model

Now we give briefly introduce the security model for WK. The assumptions of the six entities are as follows.

• **Data provider.** Not all $\mathcal{DP}$s are trustworthy. They may submit faithful/unfaithful data consciously or uncon-

$\mathcal{T}_{\text{off}}$: abstraction for off-chain TCB

Initialize(void):
$(pk_{\mathcal{E}}^{\text{sig}}, sk_{\mathcal{E}}^{\text{sig}}) := \Pi.\mathsf{KeyGen}(1^\lambda)$, Output $pk_{\mathcal{E}}^{\text{sig}}$

PreProcess($\mathsf{Msg}_3^{\text{col}}/\mathsf{Msg}_3^{\text{tra}}$):
Verify $\mathsf{Msg}_3^{\text{col}}/\mathsf{Msg}_3^{\text{tra}}$
Decrypt $\mathsf{Msg}_3^{\text{col}}/\mathsf{Msg}_3^{\text{tra}}$ to call Collect or Trace

Collect($m_{1,b}$):
Call $\mathcal{C}_{\text{psc}}.\mathsf{Collect}(m_{1,b})$ to obtain and store res

Trace($m_{1,2}$):
Call $\mathcal{C}_{\text{psc}}.\mathsf{Trace}(m_{1,2})$ to obtain inf
$c = \Sigma.\mathsf{Enc}(pk_{\text{tag}}, inf)$
$h = \mathsf{H}(inf)$
$\sigma_{\mathcal{E}} = \Pi.\mathsf{Sign}(sk_{\mathcal{E}}^{\text{sig}}, (c, h))$
Output $(c, h, \sigma_{\mathcal{E}})$

Respond($\mathsf{Msg}_2^{\text{req}}$):
Decrypt $\mathsf{Msg}_2^{\text{req}}$ to obtain $(uuid, m_{3,b})$
Call $\mathcal{C}_{\text{psc}}.\mathsf{Respond}(uuid||m_{3,b})$ to obtain $m_{4,b}$
$c = \Sigma.\mathsf{Enc}(pk_{\text{du}}, m_{4,b})$
$\sigma_{\mathcal{E}} := \Pi.\mathsf{Sign}(sk_{\mathcal{E}}^{\text{sig}}, c)$
Output $(uuid, c, \sigma_{\mathcal{E}})$

$\mathcal{T}_{\text{on}}$: abstraction for on-chain TCB

Collect(Tx^{col}): Verify Tx^{col}
Trace(Tx^{tra}): Verify Tx^{tra}
Request(Tx^{req}): Forward $\mathsf{Msg}_1^{\text{req}}$ to $\mathcal{T}_{\text{off}}$
Respond(Tx^{res}): Verify Tx^{res}

Figure 4. Hybrid TCB of WK.

sciously. • **Oracle.** $\mathcal{O}$ has a high level of credibility and it is trusted by data providers, WK server, and data users. We assume that oracles are stable, i.e., offer consistent and online data authentication and pre-tracing services. • **Proxy.** The $\mathcal{P}$ within $\mathcal{S}$ is malicious. It can tamper with or delay communications to and from the enclave. • **Enclave.** $\mathcal{E}$ is honest and executes $\mathsf{code}_{\mathcal{E}}$ as requested. It securely stores two private keys $(sk_{\mathcal{E}}^{\text{sig}}, sk_{\mathcal{E}}^{\text{enc}})$. • **Data user.** $\mathcal{DU}$ is honest-but-curious, i.e., submits a normal data request to the blockchain while prying into data privacy occasionally. • **Blockchain.** We assume that (1) $\mathcal{BC}$ runs on a secure consensus mechanism to provide a robust communication infrastructure, (2) All transactions are authenticated and integrity-protected given that they are signed, and (3) $\mathcal{C}_{\text{psc}}$ behaves honestly as requested.

3.3. Design Objectives

We require that security holds when $\mathcal{DP}$, $\mathcal{DU}$ and $\mathcal{P}$ are malicious. The functionality $\mathcal{F}_{\text{sgx2}}[\mathcal{P}, \mathcal{E}]$ reflects four security guarantees:

Data Faithfulness. Informally, data faithfulness means that adversaries, cannot convince $\mathcal{S}$ to accept data that is unfaithful or differs from the original contents of $\mathcal{DP}$.

Definition 1 (Data Faithfulness). WK achieves Data Faithfulness if any Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ interacting with $\mathcal{F}_{\text{O}_i}$ and $\mathcal{F}_{\text{sgx2}}$ cannot make an honest verifier accept $(pk_i^{\text{ats}}, pk_i^{\text{sig}}, \overline{m}_{0,x}, \overline{m}_{1,x}, \overline{\sigma}_i^{\text{ats}}, \overline{\sigma}_i, \overline{\sigma}_{\mathcal{E}}^{\text{att}})$ or $(pk_{\mathcal{E}}^{\text{sig}}, pk_{\mathcal{E}}^{\text{att}}, uuid, m_{3,b}, \overline{m}_{4,b}, \overline{\sigma}_{\mathcal{E}}, \overline{\sigma}_{\text{att}})$ where both $\overline{m}_{0,0}$ and $\overline{m}_{0,1}$ include $\overline{d}$ forged by $\mathcal{A}$ and not authenticated by any oracle, and $\overline{m}_{0,2}$ includes watermarked $\overline{wd}$ forged by $\mathcal{A}$ and not authenticated. $m_{3,b}$ is a normal data request and $\overline{m}_{4,b}$ is a request result forged by $\mathcal{A}$. Formally, for any PPT adversary $\mathcal{A}_0$, an $\mathcal{O}$, an $\mathcal{E}$ with $\mathcal{C}_{\text{psc}}$, and a security parameter λ,

$$\Pr_1^{\text{fai}}\begin{bmatrix}(pk_i^{\text{ats}}, pk_i^{\text{sig}}, \overline{m}_{0,x}, \overline{m}_{1,x}, \\ \quad \overline{\sigma}_i^{\text{ats}}, \overline{\sigma}_i, \overline{\sigma}_{\mathcal{E}}^{\text{att}}) \leftarrow \mathcal{A}_0^{\mathcal{F}}(1^\lambda), x \in \{0,1,2\} : \\ (\Gamma.\mathsf{Verify}(pk_i^{\text{ats}}, \overline{m}_{0,x}, \overline{\sigma}_i^{\text{ats}}) = 1) \wedge \\ (\Pi.\mathsf{Verify}(pk_i^{\text{sig}}, \mathsf{Enc}(pk_i^{\text{enc}}, \overline{m}_{1,x}), \overline{\sigma}_i) = 1)\end{bmatrix} \leq \mathsf{negl}(\lambda), \tag{1}$$

$$\Pr_2^{\text{fai}}\begin{bmatrix}(pk_{\mathcal{E}}^{\text{sig}}, pk_{\mathcal{E}}^{\text{att}}, uuid, m_{3,b}, \overline{m}_{4,b}, \\ \quad \overline{\sigma}_{\mathcal{E}}, \overline{\sigma}_{\mathcal{P}}^{\text{wal}}, \overline{\sigma}_{\text{att}}) \leftarrow \mathcal{A}_0^{\mathcal{F}}(1^\lambda), b \in \{0,1\} : \\ ((uuid, m_{4,b}) \neq \mathcal{C}_{\text{psc}}.\mathsf{Respond}(uuid||m_{3,b})) \wedge \\ (\Pi.\mathsf{Verify}(pk_{\mathcal{E}}^{\text{sig}}, \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{4,b}), \sigma_{\mathcal{E}}) = 1) \wedge \\ (\Pi_{\text{att}}.\mathsf{Verify}(pk_{\mathcal{E}}^{\text{att}}, (\mathsf{code}_{\mathcal{E}}, pk_{\mathcal{E}}^{\text{sig}}, pk_{\mathcal{E}}^{\text{enc}}), \sigma_{\text{att}}) = 1)\end{bmatrix} \leq \mathsf{negl}(\lambda). \tag{2}$$

Definition 2 (Data Privacy). Informally, data privacy means that adversaries (both external eavesdroppers and internal entities) cannot access data that is not intended for them to see. In specific, $\mathcal{DP}$ cannot access the submitted sensitive data from other data providers or the sensitive results sent back to data users, $\mathcal{P}$ cannot access the submitted sensitive data or the sensitive results, and $\mathcal{DU}$ cannot access any data but the ones returned to them.

Formally, for any PPT adversary $\mathcal{A}_1$ that eavesdrops on the communication channels of WK, an $\mathcal{O}$, an $\mathcal{E}$ with $\mathcal{C}_{\text{psc}}$, a $\mathcal{DU}$, and a security parameter λ,

$$\Pr_1^{\text{pri}}\begin{bmatrix}m_{0,b}^0, m_{0,b}^1 \leftarrow \mathcal{A}_1, |m_{0,b}^0| = |m_{0,b}^1|, b \in \{0,1\}, d_{0,b}^0 \neq d_{0,b}^1 \\ b_0 \xleftarrow{R} \{0,1\} \\ c_0^{b_0} \leftarrow \mathsf{Enc}(pk_i, m_{0,b}^{b_0}), c_1^{b_0} \leftarrow \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{1,b}^{b_0}) \\ b_0' \leftarrow \mathcal{A}_1(c_0^{b_0}, c_1^{b_0}) \qquad \mathbf{:}\ b_0' = b_0\end{bmatrix} \leq 1/2 + \mathsf{negl}(\lambda), \tag{3}$$

$$\Pr_3^{\text{pri}}\begin{bmatrix}(m_{3,b}^0, m_{3,b}^1) \leftarrow \mathcal{A}_1, |m_{3,b}^0| = |m_{3,b}^1|, b \in \{0,1\} \\ b_2 \xleftarrow{R} \{0,1\} \\ c_0^{b_2} \leftarrow \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{3,b}^{b_2}), c_1^{b_2} \leftarrow \mathsf{Enc}(pk_{\text{du}}^{\text{enc}}, m_{4,b}^{b_2}) \\ b_2' \leftarrow \mathcal{A}_1(c_0^{b_2}, c_1^{b_2}) \qquad \mathbf{:}\ b_2' = b_2\end{bmatrix} \leq 1/2 + \mathsf{negl}(\lambda), \tag{4}$$

$$\Pr_2^{\text{pri}}\begin{bmatrix}(m_{0,2}^0, m_{0,2}^1) \leftarrow \mathcal{A}_1, |m_{0,2}^0| = |m_{0,2}^1|, wd_{0,2}^0 \neq wd_{0,2}^1 \\ b_1 \xleftarrow{\text{R}} \{0,1\} \\ c_0^{b_1} \leftarrow \mathsf{Enc}(pk_i, m_{0,2}^{b_1}), c_1^{b_1} \leftarrow \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{1,2}^{b_1}) \\ b_1' \leftarrow \mathcal{A}_1(c_0^{b_1}, c_1^{b_1}) \qquad\qquad : b_1' = b_1\end{bmatrix}$$
$$\leq 1/2 + \mathsf{negl}(\lambda). \tag{5}$$

In ineq 3, we do not treat the two cases ($m_{0,0}$, $m_{0,1}$) differently because doing so will leak information about $\mathcal{DP}$'s intention. Besides data collection, we explicitly define ineq 4 to stand for the case of data request, because it is possible for $\mathcal{A}_1$ to differentiate $\mathcal{DP}$ and $\mathcal{DU}$ by observing the type of message (Msg and Tx, to be explained in Section 5) and whether there is a returned packet.

Besides eavesdropping, $\mathcal{DP}$ can collude with $\mathcal{P}$ to monitor the flow of data collection of other $\mathcal{DP}$s and data requests. For any PPT adversary $\mathcal{A}_2$ ($\mathcal{DP}$) colluding with $\mathcal{A}_3$ ($\mathcal{P}$), an $\mathcal{O}_i$, an $\mathcal{E}$ with $\mathcal{C}_{\text{psc}}$, a $\mathcal{DU}$, and a security parameter λ,

$$\Pr_4^{\text{pri}}\begin{bmatrix}(m_{0,x}^0, m_{0,x}^1) \leftarrow \mathcal{A}_2, |m_{0,x}^0| = |m_{0,x}^1|, b \in \{0,1,2\} \\ b_0 \xleftarrow{\text{R}} \{0,1\} \\ c_0^{b_0} \leftarrow \mathsf{Enc}(pk_i, m_{0,b}^{b_0}), c_1^{b_0} \leftarrow \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{1,b}^{b_0}) \\ c_2^{b_0} \leftarrow \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{2,b}^{b_0}), c_3^{b_0} = \mathsf{H}(m_{2,b}^{b_0}) \\ b_0' \leftarrow \mathcal{A}_2(c_0^{b_0}, c_1^{b_0}, c_2^{b_0}, c_3^{b_0}) \qquad\qquad : b_0' = b_0\end{bmatrix}$$
$$\leq 1/2 + \mathsf{negl}(\lambda). \tag{6}$$

$$\Pr_5^{\text{pri}}\begin{bmatrix}(m_{3,b}^0, m_{3,b}^1) \leftarrow \mathcal{A}_2, |m_{3,b}^0| = |m_{3,b}^1|, b \in \{0,1\} \\ b_1 \xleftarrow{\text{R}} \{0,1\} \\ c_0^{b_1} \leftarrow \mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{3,b}^{b_1}), c_1^{b_1} \leftarrow \mathsf{Enc}(pk_{\text{du}}, m_{4,b}^{b_1}), \\ b_1' \leftarrow \mathcal{A}_2(c_0^{b_1}, c_1^{b_1}) \qquad\qquad : b_1' = b_1\end{bmatrix}$$
$$\leq 1/2 + \mathsf{negl}(\lambda). \tag{7}$$

$\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{2,b}^{b_0})$ is added in ineq 6 because $\mathcal{P}$ can see the ciphertext sent from $\mathcal{E}$ to $\mathcal{ST}$.

For any PPT adversary $\mathcal{A}_3$ ($\mathcal{DU}$) colluding with $\mathcal{A}_2$ ($\mathcal{P}$), an $\mathcal{O}_i$, an $\mathcal{E}$ with $\mathcal{C}_{\text{psc}}$, a $\mathcal{DP}$, and a security parameter λ, there are $\Pr_6^{\text{pri}}$ and $\Pr_7^{\text{pri}}$ similar to $\Pr_4^{\text{pri}}$ and $\Pr_5^{\text{pri}}$ with one difference that $\mathcal{A}_3$ generates the pairs of $(m_{*,*}^0, m_{*,*}^1)$.

Verifiable Computation. Informally, VC means that adversaries, i.e., a malicious $\mathcal{DP}$, $\mathcal{P}$, cannot convince $\mathcal{C}_B$ or $\mathcal{DU}$ to accept data that is not computed by the function f in $\mathcal{C}_{\text{psc}}$ but pass the verification.

Definition 3 (Verifiable Computation). WK achieves Verifiable Computation if any PPT adversary $\mathcal{A}$ interacting with $\mathcal{F}$ cannot make $\mathcal{BC}$ accept $(pk_{\mathcal{E}}^{\text{sig}}, \sigma_{\mathcal{E}}, m_{3,0}, \overline{m}_{4,0})$ where $m_{3,0}$ is a regular request, $\overline{m}_{4,0} = (\overline{f}, \overline{\sigma}^{\text{vc}})$ is a pair of computation result and VC proof forged by $\mathcal{A}$. Formally, for any PPT adversary $\mathcal{A}_3$, an $\mathcal{E}$ with $\mathcal{C}_{\text{psc}}$, a $\mathcal{DU}$, and a security parameter λ,

$$\Pr^{\text{vc}}\begin{bmatrix}(pk_{\mathcal{E}}^{\text{sig}}, \sigma_{\mathcal{E}}, m_{3,0}, \overline{m}_{4,0}) \leftarrow \mathcal{A}_3(1^\lambda) : \\ (\overline{m}_{4,0} \notin \mathcal{C}_{\text{psc}}.\mathsf{Respond}(uuid \| m_{3,0})) \wedge \\ \Omega.\mathsf{VerSig}(vk_{\text{psc}}, (\mathsf{R}, \boldsymbol{\tau}), \overline{f}, \overline{\sigma}^{\text{vc}}) = 1) \wedge \\ (\Pi.\mathsf{Verify}(pk_{\mathcal{E}}^{\text{sig}}, \Sigma.\mathsf{Enc}(pk_{\mathsf{du}}^{\text{enc}}, \overline{m}_{4,0}), \sigma_{\mathcal{E}}) = 1)\end{bmatrix} \tag{8}$$
$$\leq \mathsf{negl}(\lambda).$$

In ineq 7, we use $\mathcal{BC}$ instead of $\mathcal{DU}$ as the verifier because the computation result is verified by the $\mathcal{BC}$ before being returned to the $\mathcal{DU}$. R is a relation (Section 4.5) and $\boldsymbol{\tau}$ is a sequence of VC signatures $\{\sigma_1^{\text{vc}}, \ldots, \sigma_t^{\text{vc}}\}$.

Leakage Traceability. Informally, leakage traceability means that adversaries, i.e., a malicious $\mathcal{DP}$ colluding with $\mathcal{P}$, cannot force $\mathcal{C}_{\text{psc}}$ to produce a private key $\overline{sk}$ that is not extracted from a watermarked data wd originally embedded with sk or produce a valid private key sk when extracting a value from wd.

Definition 4 (Leakage Traceability). WK achieves Leakage Traceability if any PPT adversary $\mathcal{A}$ interacting with $\mathcal{F}$ cannot force $\mathcal{E}$ and $\mathcal{C}_{\text{psc}}$ to produce $(pk_{\mathcal{E}}^{\text{sig}}, \sigma_{\mathcal{E}}, m_{0,2}, \overline{sk})$ or $(pk_{\mathcal{E}}^{\text{sig}}, \sigma_{\mathcal{E}},\ m_{0,2}', sk)$ where $\overline{sk}$ is given by $\mathcal{A}$ (not extracted by $\mathcal{C}_{\text{psc}}$), $m_{0,2}'$ includes an unwatermarked d, and sk belongs to some $\mathcal{DU}$. Formally, for any PPT adversary $\mathcal{A}_4$, $\mathcal{E}$ with $\mathcal{C}_{\text{psc}}$, $\mathcal{DU}$, and a security parameter λ,

$$\Pr_1^{\text{tra}}\begin{bmatrix}(pk_{\mathcal{E}}^{\text{sig}}, \sigma_{\mathcal{E}}, m_{0,2}, \overline{sk}) \leftarrow \mathcal{A}_4(1^\lambda) : \\ (sk = \mathcal{C}_{\text{psc}}.\mathsf{Extract}(wk, wd)) \wedge (\overline{sk} \neq sk) \wedge \\ (\Pi.\mathsf{Verify}(pk_{\mathcal{E}}^{\text{sig}}, (\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, \overline{sk}), \mathsf{H}(\overline{sk})), \sigma_{\mathcal{E}}) = 1)\end{bmatrix}$$
$$\leq \mathsf{negl}(\lambda), \tag{9}$$

$$\Pr_2^{\text{tra}}\begin{bmatrix}(pk_{\mathcal{E}}^{\text{sig}}, \sigma_{\mathcal{E}}, m_{0,2}', sk) \leftarrow \mathcal{A}_4(1^\lambda) : \\ (sk \neq \perp) \wedge (\perp = \mathcal{C}_{\text{psc}}.\mathsf{Extract}(wk, d)) \wedge \\ (\Pi.\mathsf{Verify}(pk_{\mathcal{E}}^{\text{sig}}, (\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, sk), \mathsf{H}(sk), \sigma_{\mathcal{E}})) = 1)\end{bmatrix}$$
$$\leq \mathsf{negl}(\lambda). \tag{10}$$

Efficiency. The WK system is expected to maintain acceptable costs on computation, storage, and communication. Specifically, the performance metrics to be evaluated include collection time of $m_{0,b}$, response time of $m_{3,b}$, tracing time of $m_{0,2}$, transaction throughput, TCB size of $\mathcal{E}$ and $\mathcal{C}_{\text{bc}}$, length of messages transmitted among entities ($\mathcal{DP}$, $\mathcal{O}_i$, $\mathcal{P}$, $\mathcal{E}$, $\mathcal{C}_{\text{psc}}$, $\mathcal{ST}$, $\mathcal{DU}$, $\mathcal{BC}$), and gas cost of all transactions.

4. Preliminaries

In this section, we provide necessary background on the blockchain, smart contract, SGX, location-sensitive hash, accountable threshold signature, homomorphic signatures for non-deterministic computation, and digital watermarking on which WK builds.

4.1. Blockchain and Smart Contract

BC is a decentralized, tamper-proof, and (optionally) public ledger that was initially used for Bitcoin to record transactions among untrustworthy users in a decentralized network [1]. The transactions are stored into separate blocks by a group of BC nodes via a consensus algorithm and the blocks are sequentially linked into a growing chain through cryptographic hashes. Blockchain nodes and users can download previously recorded transactions from the BC and verify the integrity of received data by comparing it with pertinent transactions.

SC is a piece of codes deployed on the BC with a unique address and state variables. It autonomously executes some predefined and self-executable functions, which are triggered by user-sent transactions. A contract can encode any set of rules represented in its programming language and support general computations on the BC. Executing functions in SCs raises execution fees to incentivize peers and mitigate denial of service attacks. The most remarkable system that supports expressive SC is the Turing-complete SC platform Ethereum.

4.2. SGX

SGX is a hardware extension of Intel Architecture that allows an application to establish an enclave, i.e., a protected execution space [32], [33], [34], to realize hardware protections on user-level code. Firstly, it protects the code and data from hardware attacks and software. Being in an isolated environment, an enclave process is deemed to run correctly with confidentiality. Only the processor and programs in the enclave can access the code and data in plaintext. Secondly, SGX enables a remote system to verify that a piece of software has been correctly instantiated in the enclave via remote attestation and a pair of keys.

Since SGX imposes limitations regarding memory commitment and reuse of enclave memory, Intel introduces SGX2 to extend the SGX instruction set to include dynamic memory management support for enclaves [71]. SGX2 instructions offer software with more capability to manage memory and page protections from inside an enclave while preserving the security of the SGX architecture and system software.

4.3. Location-Sensitive Hash

Given a distance metric dist, an LSH function projects close items to the same hash value with a higher probability. A hash function family $\mathcal{HF}$ is (dt_1, dt_2, pr_1, pr_2)-sensitive if for any two points p_1, p_2, and $h \in \mathcal{HF}$, two inequations hold:

$$\text{for } \mathsf{dist}(p_1, p_2) \leq dt_1 : \Pr[h(p_1) = h(p_2)] \geq pr_1,$$
$$\text{for } \mathsf{dist}(p_1, p_2) \geq dt_2 : \Pr[h(p_1) = h(p_2)] \leq pr_2.$$

4.4. Accountable Threshold Signature (ATS)

An accountable threshold signature scheme Γ consists of five algorithms [61] :

$\mathsf{KGen}(1^\lambda, n, t)$: given a security parameter λ, a number of signers n, and a threshold t, outputs a private key sk and a public key pk.

$\mathsf{Sign}(sk_i, d, \mathcal{S})$: given a secret key sk_i of signer S_i belonging to a signing group $\mathcal{S} \subseteq [n]$, and message d, outputs a signature share σ_i to a combiner, which has to interact all signers in $\mathcal{S}$.

$\mathsf{Combine}(pk, d, \mathcal{S}, \{\sigma_i\}_{i\in\mathcal{S}})$: given a public key pk, data d, a signing group $\mathcal{S}$, and signature shares $\{\sigma_i\}_{i\in\mathcal{S}}$, abort if $|\mathcal{S} \neq t|$, otherwise outputs a combined signature $\sigma = (R, z, \mathcal{S})$ computed from $\{\sigma_i\}_{i\in\mathcal{S}}$.

$\mathsf{Verify}(pk, d, \sigma = (R, z, \mathcal{S}))$: given a public key pk, data d, and a combined signature σ, outputs 1 if $|\mathcal{S}| = t$ and the Schnorr verification algorithm accepts the triple $(pk_{\mathcal{S}}, d, \sigma')$ where $pk_{\mathcal{S}} = \prod_{i\in\mathcal{S}} pk_i$ and $\sigma' = (R, z)$.

$\mathsf{Trace}(pk, d, \sigma)$: given a public key pk, data d, and a combined signature σ, invokes $\mathsf{Verify}(pk, d, \sigma)$, and if valid, outputs $\mathcal{S}$, otherwise outputs $\perp$.

4.5. Homomorphic Signatures for Non-deterministic Computation (HSNC)

A homomorphic signature scheme Ω for non-deterministic computation consists of four algorithms [56]:

$\mathsf{KGen}(1^\lambda, \mathcal{L}, \mathcal{R})$: given a security parameter λ, a set of labels $\mathcal{L}$, and a universal relation $\mathcal{R}$, outputs a secret signing key ssk and a public key vk.

$\mathsf{Sign}(sk, \tau, x)$: given a signing key sk, a label $\tau \in \mathcal{L}$, and a message $x \in \mathcal{M}$, outputs a signature σ.

$\mathsf{Eval}(vk, \mathsf{R}, y, \sigma_1^{\text{vc}}, \dots, \sigma_t^{\text{vc}}, w)$ on input a verification key vk, a relation $\mathsf{R} \in \mathcal{R}$ over $\mathcal{D}_y \times \mathcal{M}^t \times \mathcal{D}_w$, a statement $y \in \mathcal{D}_y$, signatures $\{\sigma_1^{\text{vc}}, \dots, \sigma_t^{\text{vc}}\}$, and a witness $w \in \mathcal{D}_w$, outputs a new signature σ^{vc}.

$\mathsf{VerSig}(vk, (\mathsf{R}, \boldsymbol{\tau}), y, \sigma^{\text{vc}})$: given a verification key vk, a labelled relation $(\mathsf{R}, \tau_1, \dots, \tau_t)$, a statement $y \in \mathcal{D}_y$ and a signature σ^{vc}, outputs 0 (reject) or 1 (accept).

4.6. Digital Watermarking

A robust digital watermarking scheme Λ consists of three algorithms [64] :

$\mathsf{KGen}(1^\lambda)$: given a security parameter λ, outputs a secret key wk.

$\mathsf{Emb}(wk, d)$: given a secret key wk, data d, generates a watermark wm, embeds wm into d, outputs watermarked data wd.

$\mathsf{Ext}(wk, wd)$: given a secret key wk and watermarked data wd, outputs a watermark wm or NULL.

5. WK Protocol

In this section, we first give an overview of WK and then dive into four detailed phases. More importantly, we elaborate on how we address **C1.1**, **C1.2**, and **C1.3** in Section 5.3, Section 5.4, and Section 5.5, respectively.

5.1. Overview

The WK system contains four phases, namely Setup, Data Collection, Data Request, and Data Tracing. An

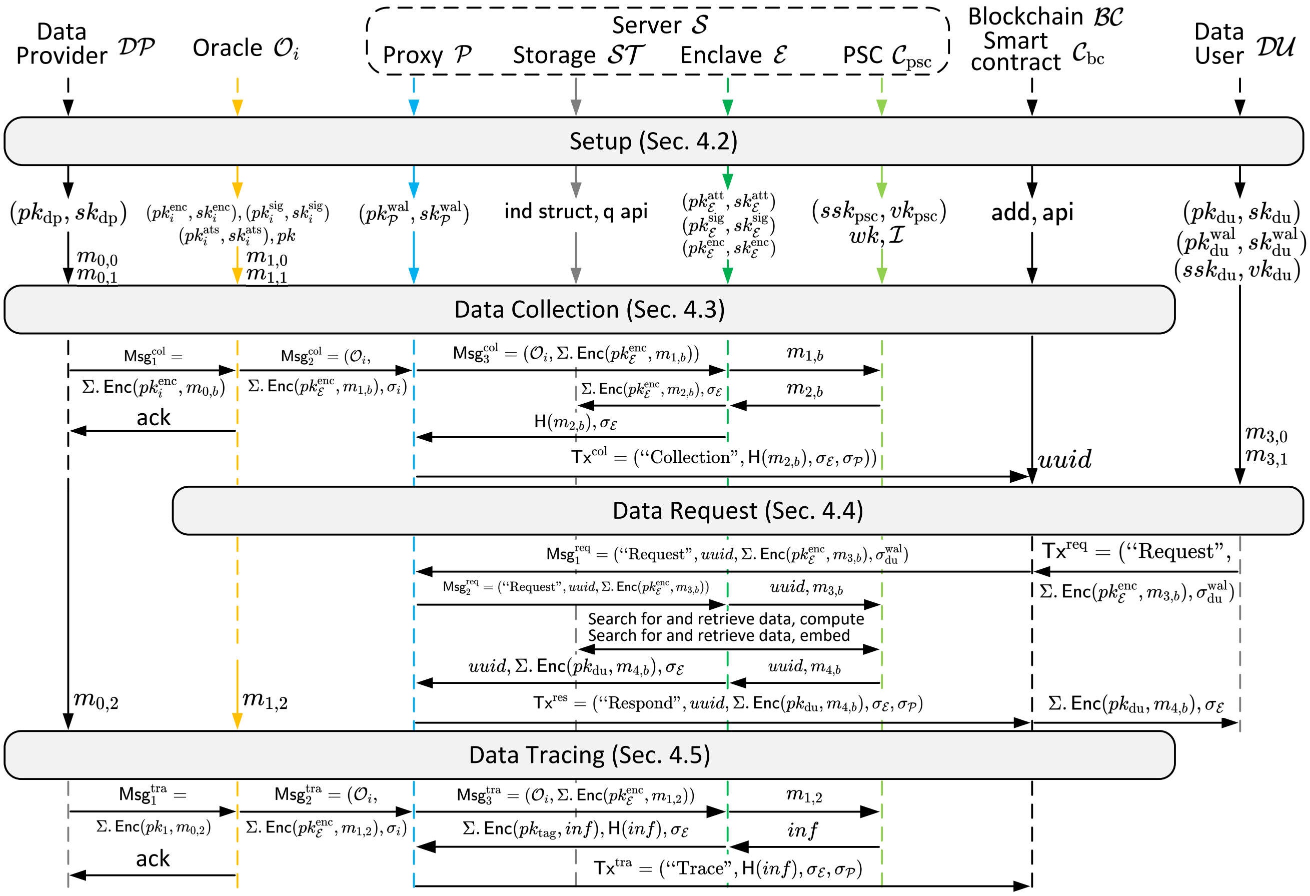

Figure 5. An Overview of the Workflow in WK. (We omit the supplementary information about data, such as A_1, A_2, ts, and $uuid$.)

overview of the workflow in WK with five entities and four phases is specified in Fig. 5. In **Setup**, a committee $\mathcal{CM}_{\mathsf{cred}}$ and an oracle committee $\mathcal{C}_{\mathsf{auth}}$ are up and running. Together with a group of WK servers, the two committees initiate a blockchain $\mathcal{BC}$, and reach an agreement on algorithms and parameters. $\mathcal{E}$s, $\mathcal{DP}$s, and $\mathcal{DU}$s generate or register for keys or credentials while $\mathcal{DP}$s and $\mathcal{C}_{\text{psc}}$s register for VC keys. In **Data Collection**, data providers submit data to at least n_1 oracles in $\mathcal{C}_{\mathsf{auth}}$, which send it to a WK server $\mathcal{S}$ after authentication. $\mathcal{S}$ verifies and stores the data. A record of data collection is written in $\mathcal{BC}$. In **Data Request**, data users submit a request to the $\mathcal{BC}$, which forwards it to a WK server. The latter searches for data and returns a response to data users via $\mathcal{BC}$. In **Data Tracing**, data providers submit watermarked data similarly and ultimately to $\mathcal{S}$ via oracles to extract the potentially embedded information.

5.2. Setup

Two committees, i.e., one of oracles for user credential generation $\mathcal{CM}_{\mathsf{cred}}$ and one of oracles for data authentication $\mathcal{CM}_{\mathsf{auth}} = \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$, are online. After negotiating with servers, $\mathcal{CM}_{\mathsf{cred}}$ and $\mathcal{CM}_{\mathsf{auth}}$ initialize $\mathcal{BC}$ with $\mathcal{C}_{\text{bc}}$ and its address add and function API api, and set $\Sigma = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as asymmetric encryption scheme, $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ as signing scheme, $\Gamma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Verify}, \mathsf{Trace})$ as ATS scheme, f as the target function for VC, $\Omega = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{VerSig})$ as HSNC, $\Lambda = (\mathsf{KGen}, \mathsf{Emb}, \mathsf{Ext})$ as watermarking scheme, H as collision-resistant hash function, and LSH as LSH. Each oracle $\mathcal{O}_i$ has a pair of encryption keys $(pk_i^{\text{enc}}, sk_i^{\text{enc}})$, a pair of signing keys $(pk_i^{\text{sig}}, sk_i^{\text{sig}})$, and a pair of ATS keys $(pk_i^{\text{ats}}, sk_i^{\text{ats}})$. The ATS public key is pk.

Each $\mathcal{S}$ initiates $\mathcal{P}$, $\mathcal{ST}$, $\mathcal{E}$, and $\mathcal{C}_{\text{psc}}$. $\mathcal{P}$ registers a blockchain wallet $\mathcal{W}$ with a pair of keys $(pk_{\mathcal{P}}^{\text{wal}}, sk_{\mathcal{P}}^{\text{wal}})$. $\mathcal{E}$ is embedded a pair of attestation keys $(pk_{\mathcal{E}}^{\text{att}}, sk_{\mathcal{E}}^{\text{att}})$, and self-generates a pair of signing keys $(pk_{\mathcal{E}}^{\text{sig}}, sk_{\mathcal{E}}^{\text{sig}})$ and a pair of encryption keys $(pk_{\mathcal{E}}^{\text{enc}}, sk_{\mathcal{E}}^{\text{enc}})$. $\mathcal{C}_{\text{psc}}$ register to a VC authority for a pair of VC keys, i.e., a secret signing key ssk_{psc} and a public key vk_{psc}, and a watermark key wk. $\mathcal{ST}$ has an index structure ind struct and a query interface q api.

$\mathcal{DU}$s generate a key pair $(pk_{\text{du}}, sk_{\text{du}})$ and interacts with the committee nodes to obtain a credential $\mathsf{cred} = (pubk, \text{“master”}, \sigma^{\mathsf{cred}})$ via a simplified freestanding service CanDID [6] since no attribute value is needed here. Special $\mathcal{DU}$s whose requests are related to the determination of copyright or responsibility have to register to an identity authority for (pid, σ_{pid}). They also register a blockchain wallet with a pair of keys $(pk_{\text{du}}^{\text{wal}}, sk_{\text{du}}^{\text{wal}})$.

$\mathcal{DP}$s generate a key pair $(pk_{\text{dp}}, sk_{\text{dp}})$ similarly. $\mathcal{DP}$s who provide data for VC also register for a pair of VC keys $(sk_{\text{dp}}, vk_{\text{dp}})$.

Program for WK Proxy $\mathcal{P}$

Initialize(void):
  $(pk^{\text{wal}}_{\mathcal{P}}, sk^{\text{wal}}_{\mathcal{P}}) := \Pi.\mathsf{KeyGen}(1^\lambda)$//for uploading a Tx
  Send init to $\mathcal{F}[\mathcal{P}, \mathcal{E}]$
  On receive $(pk^{\text{sig}}_{\mathcal{E}}, pk^{\text{enc}}_{\mathcal{E}}, \sigma_{\text{att}})$ from $\mathcal{F}[\mathcal{P}, \mathcal{E}]$:
    Publish $(pk^{\text{wal}}_{\mathcal{P}}, pk^{\text{sig}}_{\mathcal{E}}, pk^{\text{enc}}_{\mathcal{E}}, \sigma_{\text{att}})$
RelayForDP($\mathsf{Msg}^{\text{col}}_2/\mathsf{Msg}^{\text{tra}}_2$):
  If $\Pi.\mathsf{Verify}(pk_i, \mathsf{Msg}^{\text{col}}_2/\mathsf{Msg}^{\text{tra}}_2, \sigma_i)$
    Relay $\mathsf{Msg}^{\text{col}}_3/\mathsf{Msg}^{\text{tra}}_3$ to $\mathcal{F}[\mathcal{P}, \mathcal{E}]$
  On receive $(\mathsf{H}(\ldots), \sigma_{\mathcal{E}})$ from $\mathcal{F}[\mathcal{P}, \mathcal{E}]$
    $\sigma_P = \Pi.\mathsf{Sign}(sk^{\text{wal}}_{\mathcal{P}}, (\mathsf{H}(\ldots), \sigma_{\mathcal{E}}))$
    Upload $\mathsf{Tx}^{\text{col}}/\mathsf{Tx}^{\text{tra}}$ to $\mathcal{BC}$ via $\mathcal{W}$
RelayForDU($\mathsf{Msg}^{\text{req}}_1$):
  Parse $\mathsf{Msg}^{\text{req}}_1$ as ("Request", $(uuid, c), \sigma^{\text{wal}}_{\text{du}}$)
  If $\Pi.\mathsf{Verify}(pk^{\text{wal}}_{\text{du}}, \mathsf{Msg}^{\text{req}}_1, \sigma^{\text{wal}}_{\text{du}})$
    Relay $\mathsf{Msg}^{\text{req}}_2 =$ ("Request", $uuid, c$) to $\mathcal{F}[\mathcal{P}, \mathcal{E}]$
  On receive $(uuid, c, \sigma_{\mathcal{E}})$ from $\mathcal{F}[\mathcal{P}, \mathcal{E}]$
    $\sigma_P = \Pi.\mathsf{Sign}(sk^{\text{wal}}_{\mathcal{P}}, (uuid, c, \sigma_{\mathcal{E}}))$
    Upload Tx^{res} to $\mathcal{BC}$ via $\mathcal{W}$
Main(void): Loop:{
    Await $\{\mathcal{O}_i\}^n_{i=1}$ to send $\mathsf{Msg}^{\text{col}}_2$ and $\mathsf{Msg}^{\text{tra}}_2$
    Fork a process of **RelayForDP**($\mathsf{Msg}^{\text{col}}_2/\mathsf{Msg}^{\text{tra}}_2$)
    Await $\mathcal{BC}$ to forward $\mathsf{Msg}^{\text{req}}_1$
    Fork a process of **RelayForDU**($\mathsf{Msg}^{\text{req}}_1$)}End

Figure 6. The WK Proxy $\mathcal{P}$.

5.3. Data Collection

Data collection involves $\mathcal{DP}$, at least t $\mathcal{O}$s, $\mathcal{S}$, and $\mathcal{BC}$. Assume that there is a $\mathcal{DP}$ who has data d to be provided to $\mathcal{S}$. $\mathcal{DP}$ submits a collection message $\mathsf{Msg}^{\text{col}}_1 = \Sigma.\mathsf{Enc}(pk^{\text{enc}}_i, m_{0,b})$ $(b \in \{0, 1\})$ to oracle $\mathcal{O}_i$ to check faithfulness. t ciphertexts are sent in total. Each recipient $\mathcal{O}_i$ decrypts the $\mathsf{Msg}^{\text{col}}_1$ to obtain $m_{0,b}$. If the data is faithful, $\mathcal{O}_i$ encrypts $m_{1,b}$, signs the ciphertext c, sends a collection message $\mathsf{Msg}^{\text{col}}_2 = (\mathcal{O}_i, \Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{1,b}), \sigma_i)$ to the $\mathcal{P}$ of an $\mathcal{S}$, and returns a feedback message ack to $\mathcal{DP}$.

As shown in Fig. 6, $\mathcal{P}$ verifies $\mathcal{O}_i$'s signature. If it is valid, $\mathcal{P}$ relays a collection message $\mathsf{Msg}^{\text{col}}_3 = (\mathcal{O}_i, \Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{1,b}))$ to $\mathcal{E}$. Fig. 7 gives the program for WK enclave $\mathcal{E}$. The enclave calls $\mathsf{PreProcess}(\mathsf{Msg}^{\text{col}}_3)$ to decrypt $\Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{1,b})$ to obtain $m_{1,0} = (0, d, \sigma^{\text{vc}}, A_1, A_2, ts)$ if $b = 0$ or $m_{1,1} = (1, d, A_1, A_2, ts)$ if $b = 1$. Then, $\mathcal{E}$ calls $\mathsf{Collect}(m_{1,b})$, which further calls $\mathcal{C}_{\text{psc}}.\mathsf{Collect}(m_{1,b})$ and awaits a feedback.

Program for WK Enclave $\mathcal{E}$

Initialize(void):
  //Initialization call from $\mathcal{F}_{\text{sgx2}}[\mathcal{P}, \mathcal{E}]$, see Fig. 3
  $(pk^{\text{sig}}_{\mathcal{E}}, sk^{\text{sig}}_{\mathcal{E}}) := \Pi.\mathsf{KeyGen}(1^\lambda)$//for remote attestation and signing
  $(pk^{\text{enc}}_{\mathcal{E}}, sk^{\text{enc}}_{\mathcal{E}}) := \Sigma.\mathsf{KeyGen}(1^\lambda)$//for data privacy
  Register wk and pk_{tag}//for watermarking
  Call $\mathcal{C}_{\text{psc}}.\mathsf{Initialize}()$
  Output $(pk^{\text{sig}}_{\mathcal{E}}, pk^{\text{enc}}_{\mathcal{E}})$
PreProcess($\mathsf{Msg}^{\text{col}}_3/\mathsf{Msg}^{\text{tra}}_3$):
  Switch $(\Sigma.\mathsf{Dec}(sk^{\text{enc}}_{\mathcal{E}}, \mathsf{Msg}^{\text{col}}_3/\mathsf{Msg}^{\text{tra}}_3))\{$
    case $m_{1,b}$: Call $\mathsf{Collect}(m_{1,b})$
    case $m_{1,2}$: Call $\mathsf{Trace}(m_{1,2})\}$
Collect($m_{1,b}$):
  Call $\mathcal{C}_{\text{psc}}.\mathsf{Collect}(m_{1,b})$
  On receive $\mathsf{H}(m_{2,b})$ from $\mathcal{C}_{\text{psc}}$
    $\sigma_{\mathcal{E}} = \Pi.\mathsf{Sign}(sk^{\text{sig}}_{\mathcal{E}}, \mathsf{H}(m_{2,b}))$
    Store $\Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{2,b}), \sigma_{\mathcal{E}})$ in $\mathcal{ST}$
    Return $(\mathsf{H}(m_{2,b}), \sigma_{\mathcal{E}})$ to $\mathcal{P}$
Trace($m_{1,2}$):
  Parse $m_{1,2}$ as $(2, wd, A_1, A_2, ts, \sigma_i)$
  Call $\mathcal{C}_{\text{psc}}.\mathsf{Trace}(wd)$
  On receive inf from $\mathcal{C}_{\text{psc}}$
    $c = \Sigma.\mathsf{Enc}(pk_{\text{tag}}, inf)$
    $\sigma_{\mathcal{E}} = \Pi.\mathsf{Sign}(sk^{\text{sig}}_{\mathcal{E}}, c)$
    Return $(c, \sigma_{\mathcal{E}})$ to $\mathcal{P}$
Respond($\mathsf{Msg}^{\text{req}}_2$):
  Parse $\mathsf{Msg}^{\text{req}}_2$ as ("Request", $uuid, c$)
  $m_{3,b} = \Sigma.\mathsf{Dec}(sk^{\text{enc}}_{\mathcal{E}}, c)$
  If parse $m_{3,b} = (A_1, A_2, \mathsf{f}, pk, ts)$
    Call $\mathcal{C}_{\text{psc}}.\mathsf{Respond}(uuid||m_{3,0})$
    On receive $(uuid, f, \sigma^{\text{vc}})$ from $\mathcal{C}_{\text{psc}}$
    Return $(uuid, \Sigma.\mathsf{Enc}(pk_{\text{du}}, (f, \sigma^{\text{vc}})), \sigma_{\mathcal{E}})$ to $\mathcal{P}$
  Else if parse $m_{3,b} = (A_1, A_2, pid, \sigma_{pid}, pk, ts)$
    Call $\mathcal{C}_{\text{psc}}.\mathsf{Respond}(uuid||m_{3,1})$
    On receive $(uuid, wd)$ from $\mathcal{C}_{\text{psc}}$
    Return $(uuid, \Sigma.\mathsf{Enc}(pk_{\text{du}}, wd), \sigma_{\mathcal{E}})$ to $\mathcal{P}$

Figure 7. The WK Enclave $\mathcal{E}$.

Upon receiving enough number of $m_{1,b}$, $\mathcal{C}_{\text{psc}}$ searches for related signature shares $\{\sigma_{ij}\}^t_{j=1}$ by using (d, A_1, A_2) and $\mathcal{I}$. It combines them into a complete signature σ^{ats} with pk [61], as shown in Fig. 8. If the verification on σ^{ats} passes, $\mathcal{C}_{\text{psc}}$ updates $\mathcal{I}$ and returns $m_{2,0} = (d, \sigma^{\text{vc}}, A_1, A_2, ts, \sigma^{\text{ats}})$ or $m_{2,1} = (d, A_1, A_2, ts, \sigma^{\text{ats}})$ to $\mathcal{E}$.

Finally, $\mathcal{E}$ encrypts the received data with $pk^{\text{enc}}_{\mathcal{E}}$ and stores them in $\mathcal{ST}$. It also computes a hash value $\mathsf{H}(\ldots)$ of the received data, signs it, and returns them to $\mathcal{P}$. The proxy uploads a collection transaction $\mathsf{Tx}^{\text{col}} =$ ("Collection", $\mathsf{H}(\ldots), \sigma_{\mathcal{E}}, \sigma_{\mathcal{P}})$) to $\mathcal{BC}$ as a record of valid data collection.

Mark. The proposed 1-t-1 data authentication protocol among $\mathcal{DP}$, $\mathcal{O}$, and $\mathcal{S}$ achieved a secured ATS scheme among the tripartite entities. The submitted data is not only collaboratively authenticated by t oracles, but also stored securely from the $\mathcal{S}$. The protocol constructs a defensive line of screening and authenticating decentralized data feeds efficiently, thereby addressing **C1.1**.

Program for WK PSC $\mathcal{C}_{\text{psc}}$

Initialize(void):
$(sk_{\text{psc}}, vk_{\text{psc}}) := \Omega.\mathsf{KGen}(1^\lambda, \mathcal{L}, \mathcal{R})$
$wk := \Lambda.\mathsf{KGen}(1^\lambda)$, Initialize $\mathcal{I}$
Collect$(m_{1,b})$:
Search for related signature shares $\{\sigma_{ij}^{\text{ats}}\}_{j=1}^{t}$ of d
$\sigma^{\text{ats}} = \Gamma.\mathsf{Combine}(pk, d, \{\mathcal{O}_{ij}\}_{j=1}^{t}, \{\sigma_{ij}^{\text{ats}}\}_{j=1}^{t})$
If $\Gamma.\mathsf{Verify}(pk, d, \sigma^{\text{ats}}) = 1$
Update $\mathcal{I}$//Index size should not be big due to the storage limit of $\mathcal{C}_{\text{psc}}$
Return $m_{2,b}$ to $\mathcal{E}$
Trace$(m_{1,2})$:
Calculate σ^{ats} similarly to Collect
If $\Gamma.\mathsf{Verify}(pk, wd, \sigma^{\text{ats}}) = 1$
$inf = \Lambda.\mathsf{Extract}(wk, wd)$//$inf$ is a valid sk or $\perp$.
Return inf to $\mathcal{E}$
Respond$(uuid||m_{3,b})$:
If $m_{3,b} = (A_1, A_2, \mathsf{f}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$
Search $\mathcal{I}$ to retrieve $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_q\}$ from $\mathcal{ST}$
For $1 \leq i \leq q$:
$f_1 = \mathsf{f}(d_{11}, \ldots, d_{1j})$
$\sigma_1^{\text{vc}} = \Omega.\mathsf{Eval}(vk_{\text{i}}, f, \mathsf{R}, \sigma_{11}^{\text{vc}}, \ldots, \sigma_{1j}^{\text{vc}}, w)$
$//(d_{11}, \sigma_{11}^{\text{vc}}), \ldots, (d_{1j}, \sigma_{1j}^{\text{vc}}) \in \mathcal{D}_1$
$\sigma_{\text{psc},1}^{\text{vc}} = \Omega.\mathsf{Sign}(ssk_{\text{psc}}, \tau, f_1)$
$f = \mathsf{f}(f_1, \ldots, f_q)$
$\sigma^{\text{vc}} = \Omega.\mathsf{Eval}(vk_{\text{psc}}, f, \mathsf{R}, \sigma_{\text{psc},1}^{\text{vc}}, \ldots, \sigma_{\text{psc},q}^{\text{vc}}, w)$
Return $(uuid, f, \sigma^{\text{vc}})$ to $\mathcal{E}$
Else if $m_{3,b} = (A_1, A_2, pid, \sigma_{pid}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$
$wd = \Omega.\mathsf{Emb}(wk, d)$, Return $(uuid, wd)$ to $\mathcal{E}$

Figure 8. The WK PSC $\mathcal{C}_{\text{psc}}$.

5.4. Data Request

Data request involves $\mathcal{DU}$, $\mathcal{BC}$, and $\mathcal{S}$. Since data users interact with $\mathcal{S}$ through $\mathcal{BC}$ ($\mathcal{T}_{\text{on}}$), $\mathcal{DU}$ has to make sure that the public keys of $\mathcal{E}$ have an appropriate SGX2 attestation, i.e., $\mathcal{T}_{\text{on}}$ is backed by a valid $\mathcal{T}_{\text{off}}$ instance [14], before submitting a data request to $\mathcal{BC}$. To do so, $\mathcal{DU}$ checks whether $\mathsf{code}_{\mathcal{E}}$ is the claimed enclave code, checks $\Pi_{\text{att}}.\mathsf{Verify}(pk_{\mathcal{E}}^{\text{att}}, (\mathsf{code}_{\mathcal{E}}, pk_{\mathcal{E}}^{\text{sig}}, pk_{\mathcal{E}}^{\text{enc}}), \sigma_{\text{att}}) \stackrel{?}{=} 1$, and $\mathcal{T}_{\text{on}}$ is correct and parameterized with $(pk_{\mathcal{E}}^{\text{sig}}, pk_{\mathcal{E}}^{\text{enc}})$. Next, $\mathcal{DU}$ submits $\mathsf{Tx}^{\text{req}} = (\text{“Request”}, \Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{3,b}), \sigma_{\text{du}}^{\text{wal}})$ to $\mathcal{C}_{\text{bc}}$.

As specified in Fig. 9, the recipient $\mathcal{C}_{\text{bc}}$ first verifies the validity of Tx^{req} as shown in Fig. 8. If it is a valid transaction, $\mathcal{C}_{\text{bc}}$ generates $uuid$ for it and forwards a request message $\mathsf{Msg}_1^{\text{req}} = (\text{“Request”}, uuid, \Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{3,b}), \sigma_{\text{du}}^{\text{wal}})$ to $\mathcal{P}$. Here, $uuid$ is not encrypted, which does not raise integrity concerns for $\mathcal{P}$, because the integrity of on-chain messages is guaranteed by the consensus mechanism of $\mathcal{BC}$, while the integrity of messages retrieved from $\mathcal{BC}$ relies on the Transport Layer Security protocol.

$\mathcal{P}$ verifies $\sigma_{\text{du}}^{\text{wal}}$. If it is valid, $\mathcal{P}$ relays a request message $\mathsf{Msg}_2^{\text{req}} = (\text{“Request”}, uuid, \Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{3,b}))$ to $\mathcal{E}$. The enclave decrypts $\Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\text{enc}}, m_{3,b})$ to obtain $m_{3,0} = (A_1, A_2, \mathsf{f}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$ or $m_{3,1} = (A_1, A_2, pid, \sigma_{pid}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$. Then, $\mathcal{E}$ calls $\mathcal{C}_{\text{psc}}.\mathsf{Respond}(m_{3,b})$ and awaits a feedback. Upon receiving $m_{3,b}$, $\mathcal{C}_{\text{psc}}$ processes it in two cases.

(1) Verifiable Computation. As described in Fig. 8, given $m_{3,0} = (A_1, A_2, \mathsf{f}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$, $\mathcal{C}_{\text{psc}}$ first searches $\mathcal{I}$ based on (A_1, A_2, f) to retrieve from $\mathcal{ST}$ a relevant dataset $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_q\}$, where $\mathcal{D}_i$ belongs to a $\mathcal{DP}$ and contains a sequence of data $\{d_{i1}, \ldots, d_{ij}\}$. For each $\mathcal{D}_i$, $\mathcal{C}_{\text{psc}}$ computes $(f_i, \sigma_i^{\text{vc}})$ and signs f_i with ssk_{psc} to get σ_i^{vc}. Then, $\mathcal{C}_{\text{psc}}$ calculates a result (f, σ^{vc}) based on the intermediate results $\{f_i, \sigma_{\text{psc,i}}^{\text{vc}}\}_{i=1}^{q}$, and returns $(uuid, m_{4,0}) = (uuid, f, \sigma^{\text{vc}})$ to $\mathcal{E}$. We draw the proposed UHSNC with an example in Fig. 10. Three data providers submit three sets of data to $\mathcal{S}$. $\mathcal{C}_{\text{psc}}$ computes $\{f_1, f_2, f_3\}$ separately based on their data and then signs each of them with ssk_{psc} to obtain $\{\sigma_1^{\text{vc}}, \sigma_2^{\text{vc}}, \sigma_3^{\text{vc}}\}$. $\mathcal{C}_{\text{psc}}$ calculates a final result f and returns $(uuid, f, \sigma^{\text{vc}})$ to $\mathcal{E}$.

Finally, $\mathcal{E}$ encrypts $m_{4,0}$ and returns the ciphertext to $\mathcal{P}$. The proxy uploads a respond transaction $\mathsf{Tx}^{\text{res}} =$ (“Respond, $\Sigma.\mathsf{Enc}(pk_{\text{du}}, m_{4,0}), \sigma_{\mathcal{E}}, \sigma_{\mathcal{P}})$) to $\mathcal{C}_{\text{bc}}$ as a feedback. $\mathcal{C}_{\text{bc}}$ checks the validity of Tx^{res} with $\sigma_{\mathcal{P}}$ and returns $(\Sigma.\mathsf{Enc}(pk_{\text{du}}, m_{4,0}), \sigma_{\mathcal{E}})$ to $\mathcal{DU}$ by storing them in a callback pool for $\mathcal{DU}$'s retrieval.

(2) Tracing. Given $m_{3,1} = (A_1, A_2, pid, \sigma_{pid}, pk_{\text{du}}, \sigma_{\text{du}}, ts)$, $\mathcal{C}_{\text{psc}}$ computes $wd = \Omega.\mathsf{Emb}(wk, d)$ and returns $(uuid, wd)$ to $\mathcal{E}$. Finally, $\mathcal{E}$, $\mathcal{P}$, and $\mathcal{C}_{\text{bc}}$ proceed similarly to the VC process.

Mark. The proposed three-party query and response protocol among $\mathcal{DU}$, $\mathcal{BC}$, and $\mathcal{S}$ for VC achieved a universally private and verifiable computation. It smoothly bridges the gap between the original design of handling user data individually and the need for batch processing several users' data in a secure and efficient manner, thereby addressing **C1.2**.

5.5. Data Tracing

Assume there is a data user who made previously requested data wd public or a $\mathcal{DU}$ who happens to get hold of wd. $\mathcal{DU}$ submits a tracing message $\mathsf{Msg}_1^{\text{tra}} = \Sigma.\mathsf{Enc}(pk_i^{\text{enc}}, m_{0,2})$ to at least t oracles to check acquire endorsement. Each recipient $\mathcal{O}_i$ decrypts it to computes $\mathsf{LSH}(wd)$ and compares it with records in its local dataset. If there is a match, $\mathcal{O}_i$ encrypts $m_{0,2}$, signs the ciphertext, and sends a tracing message $\mathsf{Msg}_2^{\text{tra}} =$

Program for WK Contract $\mathcal{C}_{\text{bc}}$

Initialize(void):

Create Col VerifyResult pool, Req callback pool, Res callback pool

Collect(Tx^{col}):

Parse Tx^{col} as ("Collect", $\mathsf{H}(\ldots), \sigma_{\mathcal{E}}, \sigma_{\mathcal{P}}$)

$\Pi.\mathsf{Verify}(pk^{\text{wal}}_{\mathcal{P}}, \mathsf{Tx}^{\text{col}}, \sigma_{\mathcal{P}})$

$\Pi.\mathsf{Verify}(pk^{\text{sig}}_{\mathcal{E}}, \mathsf{H}(\ldots), \sigma_{\mathcal{E}})$

Trace(Tx^{tra}):

Parse Tx^{tra} as ("Trace", $\mathsf{H}(\ldots), \sigma_{\mathcal{E}}, \sigma_{\mathcal{P}}$)

$\Pi.\mathsf{Verify}(pk^{\text{wal}}_{\mathcal{P}}, \mathsf{Tx}^{\text{tra}}, \sigma_{\mathcal{P}})$

$\Pi.\mathsf{Verify}(pk^{\text{sig}}_{\mathcal{E}}, \mathsf{H}(\ldots), \sigma_{\mathcal{E}})$

Request(Tx^{req}):

Parse Tx^{req} as ("Request", $c, \sigma^{\text{wal}}_{\text{du}}$)

If $\Pi.\mathsf{Verify}(pk^{\text{wal}}_{\text{du}}, \mathsf{Tx}^{\text{req}}, \sigma^{\text{wal}}_{\text{du}}) = 1$

Emit $\mathsf{Msg}^{\text{req}}_1$ to $\mathcal{P}$ for retrieval

Respond(Tx^{res}):

Parse Tx^{res} as ("Respond", $uuid, c, \sigma_{\mathcal{E}}, \sigma_{\mathcal{P}}$)

If $\Pi.\mathsf{Verify}(pk^{\text{wal}}_{\mathcal{P}}, \mathsf{Tx}^{\text{res}}, \sigma_{\mathcal{P}}) = 1$ and

$\Pi.\mathsf{Verify}(pk^{\text{sig}}_{\mathcal{E}}, c, \sigma_{\mathcal{E}}) = 1$

Emit $(c, \sigma_{\mathcal{E}})$ to $\mathcal{DU}$ for retrieval

Figure 9. The WK Contract $\mathcal{C}_{\text{bc}}$.

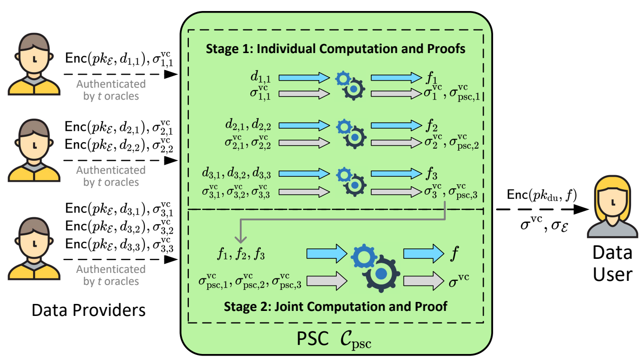

Figure 10. The Framework of Universal Homomorphic Signatures for Non-deterministic Computation (UHSNC).

$(\mathcal{O}_i, \Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{0,2}), \sigma_i)$ to the $\mathcal{P}$. The reason for using LSH is that the submitted data wd was watermarked previously while the $\mathcal{O}$ has to compare wd with each unwatermarked data d that was checked and authenticated by itself. LSH happens to meet this requirement.

Next, $\mathcal{P}$ verifies σ_i and then relays a tracing message $\mathsf{Msg}^{\text{tra}}_3 = (\mathcal{O}_i, \Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{1,2}))$ to $\mathcal{E}$. The enclave calls $\mathsf{PreProcess}(\mathsf{Msg}^{\text{tra}}_3)$ to decrypt $\Sigma.\mathsf{Enc}(pk^{\text{enc}}_{\mathcal{E}}, m_{0,2})$ to obtain $m_{1,2} = (2, wd, A_1, A_2, ts, \sigma^{\text{ats}}_i)$. Then, $\mathcal{E}$ calls $\mathsf{Collect}(m_{1,2})$, which further calls $\mathcal{C}_{\text{psc}}.\mathsf{Collect}(m_{1,2})$.

Upon receiving enough number of $m_{1,2}$, $\mathcal{C}_{\text{psc}}$ calculates a similar σ^{ats}. If the verification on σ^{ats} passes, $\mathcal{C}_{\text{psc}}$ computes $inf = \mathsf{Ext}(wk, wd)$ and returns it to $\mathcal{E}$. Finally, $\mathcal{E}$ encrypts inf with pk_{tar} and delivers it to a target authority. It returns $(\mathsf{H}(inf), \sigma_{\mathcal{E}})$ to $\mathcal{P}$. The proxy uploads a trace transaction $\mathsf{Tx}^{\text{tra}} = (\text{"Trace"}, \mathsf{H}(inf), \sigma_{\mathcal{E}}, \sigma_{\mathcal{P}})$ to $\mathcal{BC}$.

Mark. The proposed 1-t-1 data authentication protocol (Section 5.3) has laid a tracing foundation by embedding sk in d. Echoing the protocol, we propose 1-t-1 tracing protocol among $\mathcal{DP}$, t oracles $\{\mathcal{O}_i\}^t_{i=1}$, and $\mathcal{S}$. It limits the tracing power of each oracle by requiring a collective notarization and enables the oracles to search for matching files efficiently, thereby addressing **C1.3**.

6. Security Analysis

In this section, we formally prove the four security properties defined in Section 3.3.

Theorem 1. The WK system UC-realizes $\mathcal{F}_{\text{O}_i}$, achieving data faithfulness under Definition 1, if the Γ, Π, and Π_{att} are secure signature schemes.

PROOF. We prove show that if the adversary $\mathcal{A}_0$ in Definition 1 succeeds in a forgery with non-negligible probability, we can construct an adversary $\mathcal{A}'_0$ that can either break Π or Γ with non-negligible probability. There are two events to consider for $\mathcal{A}'_0$ to succeed. The first event is when $\mathcal{A}_0$ attacks the data collection phase P^{col}. In such a case, $\mathcal{A}'_0$ has to break Γ and Π simultaneously. The second event is when $\mathcal{A}_0$ attacks the data collection phase P^{req}, where $\mathcal{A}'_0$ has to break Π_{att}. Therefore, we derive two equations from ineq 1:

$$\Pr^{\text{fai}}_1 = \Pr[\mathcal{A}'_0 \text{ succeeds}|\mathcal{A}_0 \text{ succeeds in attacking } P^{\text{col}}],$$
$$\Pr^{\text{fai}}_2 = \Pr[\mathcal{A}'_0 \text{ succeeds}|\mathcal{A}_0 \text{ succeeds in attacking } P^{\text{req}}].$$

• In Event 1, we consider two cases. $\mathcal{A}'_0$ will flip a random coin to guess which case it is and abort if the guess is wrong.

– Case 1: $\mathcal{A}_0$ outputs a signature that uses the same pk^{sig}_i as the functionality $\mathcal{F}_{\text{O}_i}$ (Figure 2). In this case, $\mathcal{A}'_0$ will try to break the Π scheme. $\mathcal{A}'_0$ interacts with a signature challenger $\mathcal{CH}$ who generates a pair of keys (pk^*, sk^*), and passes pk^* to $\mathcal{A}'_0$. $\mathcal{A}'_0$ simulates $\mathcal{F}_{\text{O}_i}$ by setting $pk^{\text{sig}}_i = pk^*$. Whenever $\mathcal{F}_{\text{O}_i}$ is required to sign submitted data, $\mathcal{A}'_0$ passes the query to $\mathcal{CH}$. Since $\overline{d}$ is forged by $\mathcal{A}_0$ and not authenticated by $\mathcal{F}_{\text{O}_i}$, $\mathcal{A}'_0$ cannot have queried $\mathcal{CH}$ on the ciphertext of $\overline{d}$. Therefore, $\mathcal{A}'_0$ simply outputs what $\mathcal{A}_0$ outputs as the signature forgery.

– Case 2: $\mathcal{A}_0$ outputs a signature σ that uses a different pk^{sig}_i as the $\mathcal{F}_{\text{O}_i}$. In this case, $\mathcal{A}'_0$ will try to break the Γ scheme. $\mathcal{A}'_0$ interacts with a signature challenger $\mathcal{CH}$ who generates and passes pk^* to $\mathcal{A}'_0$ similarly. $\mathcal{A}'_0$ simulates $\mathcal{F}_{\text{O}_i}$ by setting $pk^{\text{ats}} = pk^*$. Whenever $\mathcal{F}_{\text{O}_i}$ is required to sign with sk^{ats}_i, $\mathcal{A}'_0$ passes the query to $\mathcal{CH}$. Since $\mathcal{A}$ must produce a valid signature σ^{ats}_i for a different public key to succeed, $\mathcal{A}'_0$ simply outputs what $\mathcal{A}_0$ outputs as the signature forgery.

• In Event 2, we proceed similarly to Event 1. Note that we did not include $\mathcal{P}$ in Definition 1 because once the adversary forges a signature as $\mathcal{E}$, it can deceive $\mathcal{P}$ into signing $(uuid, \Sigma.\mathsf{Enc}(pk_{\mathrm{du}}, m_{4,b}), \sigma_{\mathcal{E}})$.

- Case 1: $\mathcal{A}_0$ outputs a signature that uses the same $pk_{\mathcal{E}}^{\mathrm{sig}}$ as the functionality $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$ (Figure 3). In this case, $\mathcal{A}_0'$ will try to break the Π scheme. $\mathcal{A}_0'$ interacts with a signature challenger $\mathcal{CH}$ who generates a pair of keys (pk^*, sk^*), and passes pk^* to $\mathcal{A}_0'$. $\mathcal{A}_0'$ simulates $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$ by setting $pk_{\mathcal{E}}^{\mathrm{sig}} = pk^*$. Whenever $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$ is required to sign data returned from $\mathcal{C}_{\mathrm{psc}}$, $\mathcal{A}_0'$ passes the query to $\mathcal{CH}$. Since $(uuid, m_{4,b}) \neq \mathcal{C}_{\mathrm{psc}}.\mathsf{Respond}(uuid || m_{3,b})$, $\mathcal{A}_0'$ cannot have queried $\mathcal{CH}$ on the ciphertext of $m_{3,b}$. Therefore, $\mathcal{A}_0'$ simply outputs what $\mathcal{A}_0$ outputs as the signature forgery.
- Case 2: $\mathcal{A}_0$ outputs a signature σ that uses a different $pk_{\mathcal{E}}^{\mathrm{sig}}$ as the $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$. In this case, $\mathcal{A}_0'$ will try to break the Π_{att} scheme. $\mathcal{A}_0'$ interacts with a signature challenger $\mathcal{CH}$ who generates and passes pk^* to $\mathcal{A}_0'$ similarly. $\mathcal{A}_0'$ simulates $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$ by setting $pk_{\mathcal{E}}^{\mathrm{att}} = pk^*$. Whenever $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$ is required to sign with $pk_{\mathcal{E}}^{\mathrm{att}}$, $\mathcal{A}_0'$ passes the query to $\mathcal{CH}$. Since $\mathcal{A}$ must produce a valid signature σ_{att} for a different public key to succeed, $\mathcal{A}_0'$ simply outputs what $\mathcal{A}_0$ outputs as the signature forgery.

Given that Γ, Π, and Π_{att} are secure signature schemes, both of probabilities in Event 1 and Event 2 are negligible. In conclusion, we have the probability of $\mathcal{A}_0'$'s violating data faithfulness of is

$$\Pr^{\mathrm{fai}} = \Pr_1^{\mathrm{fai}} + \Pr_2^{\mathrm{fai}} = \Pr[\text{Evt } 1] + \Pr[\text{Evt } 2] \leq \mathsf{negl}(\lambda).$$

□

Theorem 2. The WK system UC-realizes $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$, i.e., achieving data privacy under Definition 2, verifiable computation under Definition 3, and leakage traceability under Definition 4, if Σ is CPA-secure, $\mathcal{E}$ is confidentiality-preserving, H is one-way, and the Λ is secure.

PROOF. We now prove that the protocol in Figure 5 securely realizes $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$. Specifically, we show that for any real-world adversary $\mathcal{A}$, we can construct an ideal-world simulator Sim, such that no PPT environment $\mathcal{EN}$ can distinguish whether it is in the real or ideal world [16], [70]. We refer readers to [70] for simulation-based proof techniques. We abstract away the details of data collection, data computation, and data traceability in three ideal functionality $\mathcal{F}_{col}$, $\mathcal{F}_{que}$, and $\mathcal{F}_{tra}$, respectively. The proof is captured in the following three lemmas.

Lemma 1. The WK system UC-realizes $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$, e.g., achieving data privacy under Definition 2, if Σ is CPA-secure, $\mathcal{E}$ is confidentiality-preserving, H is one-way, if the Ω and the Π are unforgeable, and the Λ is correct.

PROOF. Given a real-world adversary $\mathcal{A}$, the ideal-world adversary Sim proceeds as follows:

• Sim runs $\mathcal{A}$, $\mathcal{F}_{col}$, $\mathcal{F}_{que}$, $\mathcal{F}_{tra}$, and $\mathcal{F}_{Oracle}$ internally. Here, $\mathcal{F}_{Oracle}$ is an abstraction of the sequence of the formal three ideal functionality. Sim forwards any input en from $\mathcal{EN}$ to $\mathcal{A}$ and keeps track of the messages sent to and from $\mathcal{A}$.

• **Setup.** Upon request from $\mathcal{A}$, Sim executes $\mathsf{Prot}_{\mathsf{set}}$ as $\mathcal{O}_i$ and $\mathcal{E}$. During $\mathsf{Prot}_{\mathsf{set}}$, when $\mathcal{A}$ outputs data d intended for $\mathcal{S}$, Sim forwards it to $\mathcal{F}_{Oracle}$ as $(sid, \mathcal{S}, d)$ and forwards (sid, d) to $\mathcal{A}$ if it receives any messages from $\mathcal{F}_{set}$. By the end, Sim learns sk_i^{enc}, sk_i^{sig}, sk_i^{ats}, $sk_{\mathcal{E}}^{\mathsf{att}}$, $sk_{\mathcal{E}}^{\mathsf{sig}}$, and $sk_{\mathcal{E}}^{\mathsf{enc}}$.

• Upon a collection message from $\mathcal{A}$, Sim executes $\mathsf{Prot}_{\mathsf{col}}$ as an $\mathcal{O}_i$ using sk_i^{enc}, sk_i^{sig}, sk_i^{ats} as inputs. Sim invokes $\mathcal{F}_{col}$ as a sub-routine to execute $\mathsf{Prot}_{\mathsf{col}}$ and forwards messages to $\mathcal{S}$ as above and forwards the response from $\mathcal{O}_i$ to $\mathcal{A}$. Sim records the messages among $\mathcal{A}$, $\mathcal{O}_i$ and $\mathcal{S}$ in $\mathsf{Msg}_1^{\mathsf{col}}$, $\mathsf{Msg}_2^{\mathsf{col}}$, and ack.

• On receiving $(sid, m_{0,b})$ from $\mathcal{A}$, Sim checks the faithfulness of $m_{0,b}$. If the check passes, Sim sends $m_{0,b}$ to $\mathcal{F}_{col}$ and instructs it to send the output to $\mathcal{S}$ and $\mathcal{A}$. Sim outputs whatever $\mathcal{A}$ outputs.

• Upon a request message from $\mathcal{A}$, Sim executes $\mathsf{Prot}_{\mathsf{que}}$ as $\mathcal{BC}$. Sim invokes $\mathcal{F}_{que}$ as a sub-routine to execute $\mathsf{Prot}_{\mathsf{que}}$ and forwards messages to $\mathcal{S}$ as above and forwards the response from $\mathcal{S}$ to $\mathcal{A}$. Sim records the messages among $\mathcal{A}$, $\mathcal{BC}$, and $\mathcal{S}$ in $\mathsf{Tx}^{\mathsf{req}}$ and $(\Sigma.\mathsf{Enc}(pk_{\mathrm{du}}, m_{4,0}), \sigma_{\mathcal{E}})$.

• On receiving $(sid, m_{4,b})$ from $\mathcal{A}$, Sim verifies that

$$\Pi.\mathsf{Verify}(\mathsf{Tx}^{\mathsf{req}}, pk_{\mathsf{du}}^{\mathsf{wal}}, \sigma_{\mathsf{du}}^{\mathsf{wal}}) = 1.$$

If the verification passes, Sim sends $m_{3,b}$ to $\mathcal{F}_{que}$ and instructs it to send the output to $\mathcal{S}$ and $\mathcal{A}$. Sim outputs whatever $\mathcal{A}$ outputs.

• Upon a tracing request from $\mathcal{A}$, Sim executes $\mathsf{Prot}_{\mathsf{tra}}$ as an $\mathcal{O}_i$ using sk_i^{enc}, sk_i^{sig}, sk_i^{ats} as inputs. Sim invokes $\mathcal{F}_{tra}$ as a sub-routine to execute $\mathsf{Prot}_{\mathsf{col}}$ and forwards messages to $\mathcal{S}$ as above and forwards the response from $\mathcal{O}_i$ to $\mathcal{A}$. Sim records the messages among $\mathcal{A}$, $\mathcal{O}_i$ and $\mathcal{S}$ in $\mathsf{Msg}_1^{\mathsf{tra}}$, $\mathsf{Msg}_2^{\mathsf{tra}}$, and ack.

• On receiving $(sid, m_{0,2})$ from $\mathcal{A}$, Sim just sends $m_{0,2}$ to $\mathcal{F}_{tra}$ and instructs it to send the output to $\mathcal{S}$ and $\mathcal{A}$. Sim outputs whatever $\mathcal{A}$ outputs.

Now we argue that the ideal world execution with $\mathcal{S}$ is indistinguishable from the real world execution from the perspective of the environment by defining a sequence of hybrid games.

Hybrid $\mathbf{H_0}$. is the real-world execution of $\mathsf{Prot}_{\mathsf{WK}}$.

Hybrid $\mathbf{H_1}$. Hybrid 1 is the same as Hybrid 0 except for the following changes:

- When an honest $\mathcal{DP}$ produces a ciphertext c for $\mathcal{O}_i$, Sim will replace c with $\Sigma.\mathsf{Enc}(pk_i^{\mathsf{enc}}, 0)$ before passing it onto $\mathcal{EN}$.
- After $\mathcal{F}_{col}$ produces a ciphertext, Sim will replace it with $\Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\mathsf{enc}}, 0)$, together with a new signature.
- When an honest $\mathcal{DU}$ produces a ciphertext for $\mathcal{BC}$, Sim will replace this ciphertext with $\Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\mathsf{enc}}, 0)$ with a new signature before passing it onto $\mathcal{EN}$.
- After $\mathcal{F}_{que}$ produces a ciphertext c, Sim will replace c with $\Sigma.\mathsf{Enc}(pk_{\mathsf{du}}, 0)$ together with a new signature.

The first two conditions already cover the honest model in both data collection and data tracing. The last two conditions cover the honest model in data query. It is immediately clear that if Σ is CPA secure, then no PPT $\mathcal{E}$ can distinguish Hybrid 1 from Hybrid 0 except with negligible probability.

Hybrid $\mathbf{H_2}$. Hybrid 2 is the same as Hybrid 1 except for the following changes:

- When a malicious $\mathcal{DP}$ produces a ciphertext for $\mathcal{O}_i$, Sim will produce $q_1 - 1$ random numbers, encrypt them with pk_i^{enc} and pass their ciphertext to $\mathcal{EN}$, where q_1 is the number of data for $\mathcal{E}$ to collect in a batch.
- After $\mathcal{F}_{col}$ produces a batch of q_1 ciphertexts, Sim will replace each of them with $\Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\mathrm{enc}}, 0)$ with a new signature.

The two conditions above deal with the malicious model in both data collection and data tracing. The confidentiality-preserving property of TEE is assumed to be proved by the fact that for any two traces that have equivalent attacker operations and equivalent observations of the enclave execution, but possibly different enclave private states and executions, its sequence of states, is identical [72]. Therefore, we have that if Σ is CPA secure and TEE is confidentiality-preserving, then no PPT $\mathcal{EN}$ can distinguish Hybrid 2 from Hybrid 1 except with negligible probability.

Hybrid $\mathbf{H_3}$. Hybrid 3 is the same as Hybrid 2 except for the following changes:

- When a malicious $\mathcal{DU}$ produces a request transaction for $\mathcal{BC}$, Sim will produce $q_2 - 1$ random numbers, encrypt them with $pk_{\mathcal{E}}^{\mathsf{enc}}$ and pass their ciphertext to $\mathcal{EN}$, where q_2 is the number of data for $\mathcal{E}$ to process queries in a batch.
- After $\mathcal{F}_{que}$ produces a batch of q_2 ciphertexts, Sim will replace each of them with $\Sigma.\mathsf{Enc}(pk_{\mathcal{E}}^{\mathrm{enc}}, 0)$, together with a new signature.

The two conditions above deal with the malicious model in both data request. Similarly, we have that if Σ is CPA secure and TEE is confidentiality-preserving, then no PPT $\mathcal{EN}$ can distinguish Hybrid 3 from Hybrid 2 except with negligible probability.

Hybrid $\mathbf{H_4}$. Hybrid 4 is the same as Hybrid 3 except for the following changes:

- After $\mathcal{F}_{que}$ produces a batch of q_2 hash values, Sim will replace each of them with $\mathcal{H}$ of a random number.

Given the confidentiality-preserving property of TEE and the one-wayness [73] of $\mathcal{H}$, Hybrid 4 is computationally indistinguishable from the ideal simulation to any PPT $\mathcal{EN}$. $\square$

Lemma 2. The WK system UC-realizes $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$, e.g., achieving verifiable computation under Definition 3, if the Ω and the Π are unforgeable.

PROOF. The proof is strightforward in that for $\mathcal{A}_3$ to make $\mathcal{BC}$ accept a wrong answer, $\mathcal{A}_3$ must output $(\overline{f}, \overline{\sigma}^{\mathsf{vc}})$ and $(\Sigma.\mathsf{Enc}(pk_{\mathsf{du}}^{\mathsf{enc}}, \sigma_{\sigma_{\mathcal{E}}})$ that pass the verification, which contradicts the unforgeability of Ω and Π, i.e., $\Pr^{\mathrm{vc}}$ is negligible. $\square$

Lemma 3. The WK system UC-realizes $\mathcal{F}_{\mathrm{sgx2}}[\mathcal{P}, \mathcal{E}]$, e.g., achieving leakage traceability under Definition 4, if the Λ is correct.

PROOF. Leakage traceability focuses on recovering the watermark from the watermarked data. The proof is obvious that due to the correctness of Λ, i.e., $\Pr[\Lambda.\mathsf{Ext}(wk, \Lambda.\mathsf{Emb}(wk, d)) = wm]$=1 such that both $\Pr_1^{\mathrm{tra}}$ and $\Pr_2^{\mathrm{tra}}$ are negligible, WK system achieves leakage traceability. $\square$

7. Implementation and Evaluation

In this section, we build a WK prototype and evaluate its performance. The full version of this paper and the source codes of the WK system can be downloaded from Github [74].

7.1. Applications and Datasets

We explored three applications of WK to demonstrate its practicality and efficiency: medical service, finance, and supply chain management. For medical service, we used the Hospital Consumer Assessment of Healthcare Providers and Systems (HCAHPS) dataset [75], a national standardized survey of hospital patients about their experiences during a recent inpatient hospital stay. For finance, we selected the Bankruptcy dataset [76] from the Taiwan Economic Journal. For supply chain management, we downloaded the Supply Chain Analysis dataset from Kaggle [77].

The three datasets contain only computable data, i.e., numeric data. For the independant data, we took 100 images by using our smartphones and compresed them into ones with a size of 360KB in jpeg format.

7.2. Experimental Settings

Parameters. The key experimental parameters of WK are listed in Table 2.

TABLE 2. KEY EXPERIMENTAL PARAMETERS

Notation	Parameter	Value (bits)
Threshold, Number of oracles	(t, n)	$(3, 8)$
Length of Σ keypair	$len(pk^{\mathrm{enc}}, sk^{\mathrm{enc}})$	$(2048, 2048)$
Length of Π keypair	$len(pk^{\mathrm{sig}}, sk^{\mathrm{sig}})$	$(512, 256)$
Length of ATS keypair	$len(pk^{\mathrm{ats}}, sk^{\mathrm{ats}})$	$(512, 256)$
Length of Ω keypair	$len(vk, sk)$	$(5113280, 768)$
Length of Wallet W keypair	$len(pk^{\mathrm{wal}}, sk^{\mathrm{wal}})$	$(512, 256)$
Length of watermark key	$len(wk)$	5776

Metrics. We evaluate WK using the following metrics. **1) TCB Size**: Lines of code for custom code in $\mathcal{E}$ and on-chain $\mathcal{SC}$. **2) Computational Time**: Average time cost per procedure. **3) Communication Overhead**: Average transmitted bits per procedure. **4) Gas Costs**: Gas consumed for blockchain transactions. **5) Scalability**: Response time of $\mathcal{E}$ under multiple $\mathcal{DP}$s, $\mathcal{DU}$s, and data.

Setup. The WK Server was deployed on an Alibaba Cloud *ecs.g7t.xlarge* instance (4 vCPUs, 16 GB RAM, 8 GB encrypted SGX memory). It was containerized with Gramine [78] to support SGX. The other entities ran on a Lenovo ThinkBook16p (Intel Core i9-13900H, 32 GB

RAM). They were packaged in regular Docker images. An Ethereum network was set up using Geth in development mode. WK operations were implemented in Python 3.9, and SCs in Solidity 0.8.0. HSNC and watermarking were added as precompiled contracts to the EVM and compiled into .so files for low-level execution. The Σ is RSA, the Π in Ethereum is ECDSA, and the Γ is based on Secp256k1. The H is Keccak-256 for on-chain and off-chain consistency. We implemented Average, Maximum, and Minimum as f.

7.3. Performance

TCB Size. The TCB size of $\mathcal{E}$ is 2008 lines, including 679 lines of Python, 292 lines of Solidity, 319 lines of Rust, and 718 lines of C++, excludes comments, blank lines, and 'import' statements. The TCB size of $\mathcal{C}_{bc}$ is 122 lines of Solidity.

Computational Time. For computable data, we utilized two data items from the medical dataset to evaluate basic functionality. The average costs in each procedure are recorded in Table 3. During the data collection, the time for each entity ranges from **0.11 s** to **1.34 s**. During the data request, the time for each entity ranges from **0.0028 s** to **14.54 s**. The computational burden is primarily concentrated on entities $\mathcal{E}$, which would have higher computational capabilities to handle intensive tasks in a real-world scenario.

Communication Overhead. As recorded in Table 4. For computable data, the cost of the whole process ranges from **0.50 KB** to **13.32 KB**.

TABLE 3. COMPUTATIONAL TIME (S)

Entity	Collection	Request	Tracing
$\mathcal{DP}$	1.34 / 1.52	n/a	n/a
$\mathcal{O}$	0.21 / 98.58	n/a	n/a / 23.89
$\mathcal{P}$	0.0031 / 0.091	0.0028 / 0.0078	n/a / 0.28
$\mathcal{E}$	1.39 / 75.72	14.54 / 1.28	n/a / 95.76
$\mathcal{B}$	0.11 / 0.11	0.45 / 0.19	n/a / 0.11
$\mathcal{DU}$	n/a / n/a	0.29 / 23.21	n/a / 0.19

1.34/1.52 represents 1.34 s and 1.52 s for computable data and independent data, respectively

TABLE 4. COMMUNICATION OVERHEAD (KB)

Entity	Collection	Request	Tracing
$\mathcal{DP}$	1.00 / 361.89	n/a	n/a
$\mathcal{O}$	3.22 / 723.78	n/a	n/a / 443.69
$\mathcal{P}$	13.32 / 2174.34	8.29 / 448.49	n/a / 2665.77
$\mathcal{E}$	6.66 / 1085.67	6.57 / 446.09	n/a / 1332.66
$\mathcal{B}$	0.50 / 0.50	2.52 / 2.63	n/a / 0.50
$\mathcal{DU}$	n/a	4.37 / 443.71	n/a / 443.00

Gas Costs. Currently, 1 gas costs about 2.05×10^{-9} ether, which is 4.91×10^{-6} USD [79]. The gas cost of Tx^{col} is 2,305,136 (\$11.31), Tx^{tra} is 2,285,691 (\$11.21), Tx^{req} is 2,305,136 (\$6.59), and Tx^{res} is 2,389,050 (\$11.72). The costs can be greatly reduced by running on a Layer 2 solution [80].

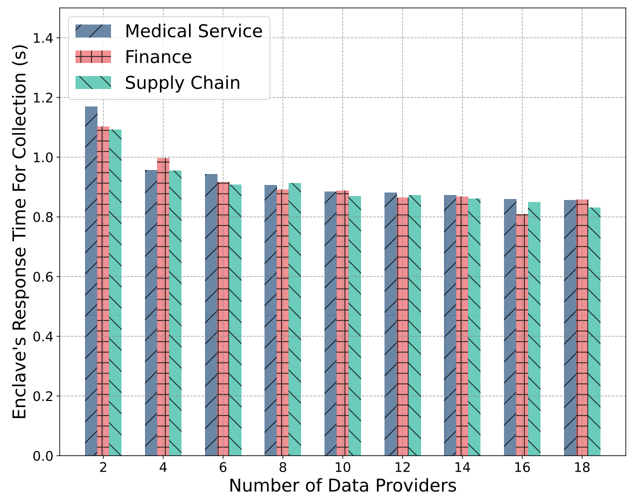

Figure 11. Scalability by varying number of $\mathcal{DP}$s.

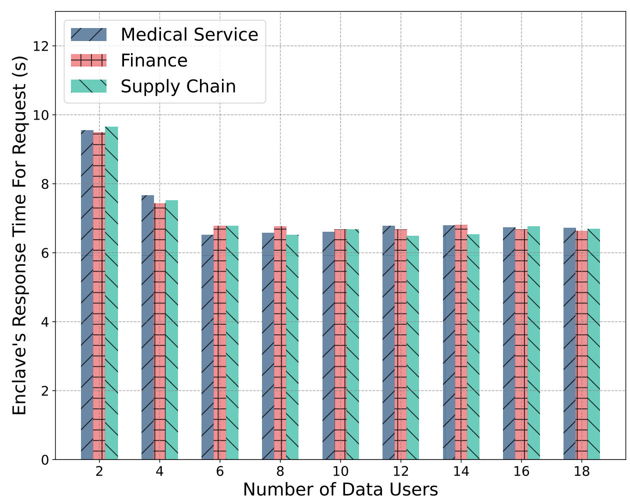

Figure 12. Scalability by varying number of $\mathcal{DU}$s.

Scalability. We evaluated the scalability under different number of concurrent operations performed by data providers and data users in three applications. As shown in Fig. 11 and Fig. 12, an increase in concurrent data collections by providers and data requests by users leads to an increase in total processing time. The average response time for each data provider and each data user remains stable **approximately 1 s** and **7 s**, respectively. From Fig. 13, when varying the number of computable data from 100 to 1000, the request time increases linearly from around 29 s to 365 s. This is because PSC has to process more batches of data when data size is larger.

We also evaluate the scalability of watermark embedding and watermark extraction. We test the time cost of the two processes time by asking one data user to ask for 1 to 100 images. As depicted in Fig. 14 and Fig. 15, the average embedding time is **less than 1 s** and the extraction time is

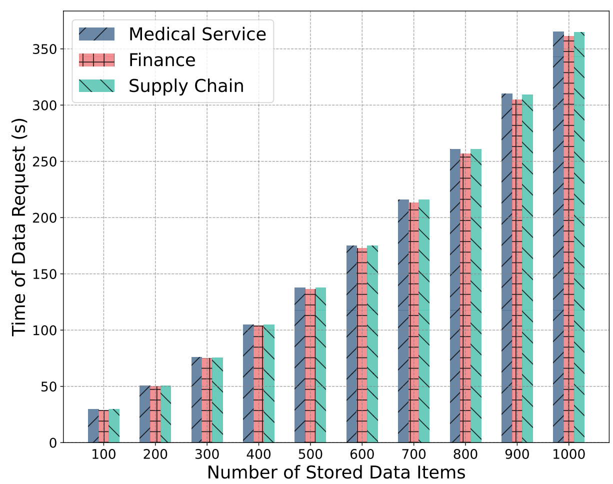

Figure 13. Scalability by varying number of computable data.

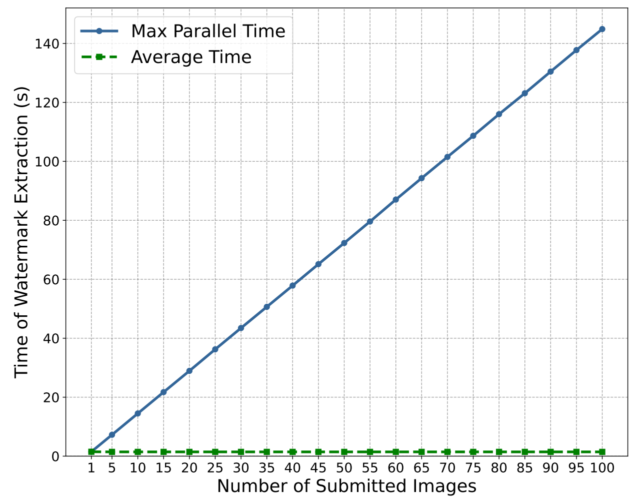

Figure 15. Scalability of Watermark Extraction.

less than 2 s.

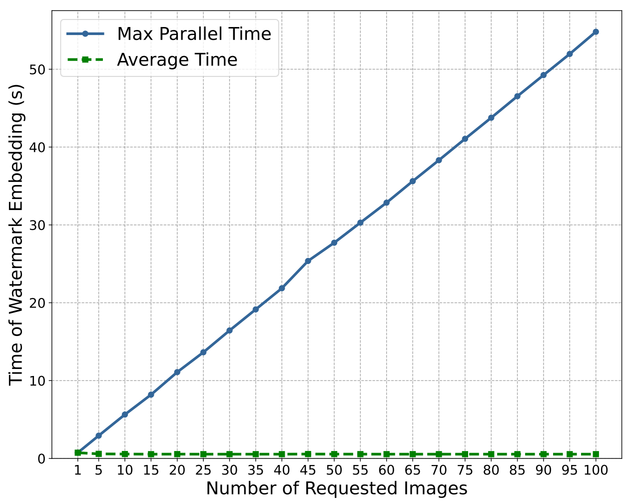

Figure 14. Scalability of Watermark Embedding.

8. Conclusions

We have introduced a data governance system WK to provide faithful, private, verifiable, and traceable data feeds, fostering valuable trust among entities of varying trustworthiness in a decentralized world. WK enables large-scale data collection and data screening. It facilitates privately verifiable computation on crowdsourced data and securely embeds identifiable information into independent files. In the event of data leakage, WK is capable of securely extracting the identifiable information from the leaked file. The three seamlessly integrated phases govern the full lifecycle of data flows, significantly enhancing confidence in data. We have defined and proved through a formal model that WK achieves data faithfulness, data privacy, verifiable computation, and leakage traceability. Comprehensive experimental evaluations based on three real-world datasets further substantiate the practicality and efficiency of WK.

References

[1] S. Nakamoto. (2016) Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] S. Ruoti, B. Kaiser, A. Yerukhimovich, J. Clark, and R. Cunningham, "Blockchain technology: What is it good for?" *Communications of the ACM*, vol. 63, no. 1, pp. 46–53, December 2020.

[3] M. Lounds. (2020) Blockchain and its implications for the insurance industry, https://www.munichre.com/us-life/en/perspectives/underwriting/blockchain-implications-insurance-industry.html. [Online]. Available: https://www.munichre.com/us-life/en/perspectives/underwriting/blockchain-implications-insurance-industry.html

[4] B. N. Ramya, V. Varsha, S. George, D. Masrik, N. Roshan, B. Ernest, W. Songjie, M. J. V. R., H. K. Anuarul, and C. Prasad, "Claimchain: Secure blockchain platform for handling insurance claims processing," in *Proceedings of the 4th IEEE International Conference on Blockchain (ICBC)*, Melbourne, Australia, 2021, pp. 55–64.

[5] IBM. Blockchain for digital identity and credentials, https://www.ibm.com/blockchain-identity. [Online]. Available: https://www.ibm.com/blockchain-identity

[6] M. Deepak, M. Harjasleen, Z. Fan, J.-L. Nerla, F. Alexander, K. Tyler, L. Tyrone, M. Christine, J. Ari, and M. Andrew, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *Proceedings of the 42nd IEEE Symposium on Security and Privacy (IEEE S&P)*, San Francisco, CA, USA, 2021, pp. 1348–1366.

[7] Amazon. Blockchain for supply chain: Track and trace, https://aws.amazon.com/cn/blockchain/blockchain-for-supply-chain-track-and-trace. [Online]. Available: https://aws.amazon.com/cn/blockchain/blockchain-for-supply-chain-track-and-trace

[8] W. Boya, L. Wouter, S. Justinas, N. V. Graf, and T. Carmela, "Not yet another digital id: Privacy-preserving humanitarian aid distribution," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (IEEE S&P)*, San Francisco, CA, USA, 2023, pp. 645–663.

[9] M. Zbrogn. (2023) Blockchain forensics: How investigators track cryptocurrencies, https://www.forensicscolleges.com/blog/blockchain-forensics. [Online]. Available: https://www.forensicscolleges.com/blog/blockchain-forensics

[10] M. Li, Y. Chen, C. Lal, M. Conti, M. Alazab, and D. Hu, "Eunomia: Anonymous and secure vehicular digital forensics based on blockchain," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 20, no. 1, pp. 225–241, January-February 2023.

[11] Ethereum, https://ethereum.org/en. [Online]. Available: https://ethereum.org/en

[12] G. Wood. (2023) Ethereum: A secure decentralised generalised transaction ledger, https://ethereum.github.io/yellowpaper/paper.pdf. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[13] R. Zhu, C. Ding, and Y. Huang, "Efficient publicly verifiable 2pc over a blockchain with applications to financially-secure computations," in *Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, London, UK, 2019, pp. 633–650.

[14] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Vienna, Austria, 2016, pp. 270–282.

[15] J. Guarnizo and P. Szalachowski, "Pdfs: Practical data feed service for smart contracts," in *Proceedings of the 24th European Symposium on Research in Computer Security (ESORICS)*, Luxembourg, 2010, pp. 767–789.

[16] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for tls," in *Proceedings of the 27th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Virtual Event, USA, 2020, pp. 1919–1938.

[17] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proceedings of the 37th IEEE Symposium on Security and Privacy (IEEE S&P)*, San Jose, CA, USA, 2016, pp. 839–858.

[18] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (IEEE S&P)*, Berkeley, CA, USA, 2013, pp. 238–252.

[19] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO)*, Santa Barbara, CA, USA, 2013, pp. 90–108.

[20] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. T. Vechev, "zkay: Specifying and enforcing data privacy in smart contracts," in *Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, London, UK, 2019, pp. 1759–1776.

[21] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *Proceedings of the 24th Financial Cryptography and Data Security (FC)*, Kota Kinabalu, Malaysia, 2020, pp. 423–443.

[22] I. Damgård. (2010) On σ-protocols, https://www.cs.au.dk/~ivan/Sigma.pdf. [Online]. Available: https://www.cs.au.dk/~ivan/Sigma.pdf

[23] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Proceedings of the 38th IEEE Symposium on Security and Privacy (IEEE S&P)*, Francisco, California, USA, 2018, pp. 315–334.

[24] S. Steffen, B. Bichsel, and M. T. Vechev, "Zapper: Smart contracts with data and identity privacy," in *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, USA, 2022, pp. 2735–2749.

[25] H. A. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, Baltimore, MD, USA, 2018, pp. 1353–1370.

[26] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proceedings of the 4th IEEE European Symposium on Security and Privacy (EuroS&P)*, Stockholm, Sweden, 2019, pp. 185–200.

[27] Y. Xiao, N. Zhang, J. Li, W. Lou, and Y. T. Hou, "Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution," in *Proceedings of the 25th European Symposium on Research in Computer Security (ESORICS)*, Guildford, UK, 2020, pp. 610–629.

[28] Q. Ren, Y. Wu, H. Liu, Y. Li, A. Victor, H. Lei, L. Wang, and B. Chen, "Cloak: Transitioning states on legacy blockchains using secure and publicly verifiable off-chain multi-party computation," in *Proceedings of the 38th Annual Computer Security Applications Conference (ACSAC)*, Austin, TX, USA, 2022, pp. 117–131.

[29] T. Frassetto, P. Jauernig, D. Koisser, D. Kretzler, B. Schlosser, S. Faust, and A.-R. Sadeghi, "Pose: Practical off-chain smart contract execution," in *Proceedings of the 30th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2023.

[30] M. Li, Y. Chen, C. Lal, M. Conti, F. Martinelli, and M. Alazab, "Nereus: Anonymous and secure ride-hailing service based on private smart contracts," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 20, pp. 2849–2866, July-August 2023.

[31] R. Qian, L. Yue, W. Yingjun, W. Yuchen, L. Hong, W. Lei, and C. Bangdao, "Decloak: Enable secure and cheap multi-party transactions on legacy blockchains by a minimally trusted tee network," *IEEE Transactions on Information Forensics and Security (IFS)*, vol. 18, pp. 1–1, September 2023.

[32] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, Tel-Aviv, Israel, 2013, pp. 1–8.

[33] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. V. Rozas, "Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," in *Proceedings of the 5th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, Seoul, South Korea, 2016, pp. 1–9.

[34] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, Tel-Aviv, Israel, 2013, p. 10.

[35] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, 2018, pp. 967–984.

[36] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, and H. Jin, "Sdte: A secure blockchain-based data trading ecosystem," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 15, pp. 725–737, August 2020.

[37] J. Frankle, S. Park, D. Shaar, S. Goldwasser, and D. J. Weitzner, "Practical accountability of secret processes," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, Baltimore, MD, USA, 2018, pp. 657–674.

[38] T. Bristow. (2023) Uk police forces accidentally shared victims' details in data breach, https://www.politico.eu/article/crime-victims-details-accidentally-included-in-police-foi-responses. [Online]. Available: https://www.politico.eu/article/crime-victims-details-accidentally-included-in-police-foi-responses

[39] S. Doughty. (2014) Met shame as officers leak data from force computer to criminals: 300 uniform and civilian staff caught abusing system, https://www.dailymail.co.uk/news/article-2621016/Met-shame-officers-leak-data-force-computer-criminals-300-uniform-civilian-staff-caught-abusing-system.html. [Online]. Available: https://www.dailymail.co.uk/news/article-2621016/Met-shame-officers-leak-data-force-computer-criminals-300-uniform-civilian-staff-caught-abusing-system.html

[40] A. Kumar, C. Fischer, S. Tople, and P. Saxena, "A traceability analysis of monero's blockchain," in *Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS)*, Oslo, Norway, 2017, pp. 153–173.

[41] Y. Chen, A. Zhou, X. Liang, N. Xie, H. Wang, and X. Li, "A traceability system of livestock products based on blockchain and the internet of things," in *Proceedings of the 39th IEEE International Performance Computing and Communications Conference (IPCCC)*, Austin, TX, USA, 2021, pp. 1–5.

[42] X. Yang, Z. Ning, L. Wenjing, and H. Y. Thomas, "A decentralized truth discovery approach to the blockchain oracle problem," in *Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM)*, New York City, NY, USA, 2023, pp. 1–10.

[43] P. Tsankov, A. M. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. T. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, 2018, pp. 67–82.

[44] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2019.

[45] P. Bose, D. Das, Y. Chen, Y. Feng, C. Kruegel, and G. Vigna, "Sailfish: Vetting smart contract state-inconsistency bugs in seconds," in *Proceedings of the 43rd IEEE Symposium on Security and Privacy (IEEE S&P)*, San Francisco,CA, USA, 2022, pp. 161–178.

[46] S. Cui, G. Zhao, Y. Gao, T. Tavu, and J. Huang, "Vrust: Automated vulnerability detection for solana smart contracts," in *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, USA, 2022, pp. 639–652.

[47] K. Babel, P. Daian, M. Kelkar, and A. Juels, "Clockwork finance: Automated analysis of economic security in smart contracts," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (IEEE S&P)*, San Francisco, CA, USA, 2023, pp. 2499–2516.

[48] V. Madathil, S. A. K. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sanchez, "Cryptographic oracle-based conditional payments," in *Proceedings of the 30th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, 2023.

[49] Defi, https://defi.org. [Online]. Available: https://defi.org

[50] S. Ellis, A. Juels, and S. Nazarov. Chainlink: A decentralized oracle network, https://link.smartcontract.com/whitepaper. [Online]. Available: https://link.smartcontract.com/whitepaper

[51] S. Eskandari, M. Salehi, W. C. Gu, and J. Clark, "Sok: Oracles from the ground truth to market manipulation," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies (AFT)*, Arlington, VA, USA, 2021, pp. 1919–1938.

[52] A. M. Pasdar, Y. C. Lee, and Z. Dong, "Connect api with blockchain: A survey on blockchain oracle implementation," *ACM Computing Surveys*, vol. 55, pp. 1–39, October 2023.

[53] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," in *Proceedings of the 28th USENIX Security Symposium (USENIX)*, Santa Clara, CA, USA, 2019, pp. 801–818.

[54] R. Li, Q. Wang, Q. Wang, D. Galindo, and M. Ryan, "Sok: Tee-assisted confidential smart contract," in *Proceedings on Privacy Enhancing Technologies (PETS)*, August 2022, pp. 711–731.

[55] C. Delerablée and D. Pointcheval, "Dynamic threshold public-key encryption," in *Proceedings of the 28th Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, CA, USA, 2018, pp. 317–334.

[56] D. Fiore and I. Tucker, "Efficient zero-knowledge proofs on signed data with applications to verifiable computation on data streams," in *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Los Angeles, CA, USA, 2022, pp. 1067–1080.

[57] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Proceedings of the The Cryptographers' Track at the RSA Conference (CT-RSA)*, San Francisco, CA, USA, 2016, pp. 111–126.

[58] S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures: extended abstract," in *Proceedings of the 8th ACM conference on Computer and Communications Security (CCS)*, Philadelphia, Pennsylvania, USA, 2001, pp. 245–254.

[59] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," *Designs, Codes, and Cryptography*, vol. 87, no. 9, pp. 2139–2164, February 2019.

[60] N. Jonas, R. Tim, and S. Yannick, "Musig2: Simple two-round schnorr multi-signatures," in *Proceedings of the 41st Annual International Cryptology Conference (CRYPTO)*, Cham, 2021, pp. 189–221.

[61] D. Boneh and C. Komlo, "Threshold signatures with private accountability," in *Proceedings of the 42nd Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, CA, USA, 2022, pp. 551–581.

[62] S. Agrawal and M. Chase, "Fame: Fast attribute-based message encryption," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Dallas, TX, USA, 2017, pp. 665–682.

[63] S. Dougherty, R. Tourani, G. Panwar, R. Vishwanathan, S. Misra, and S. Srikanteswara, "Apecs: A distributed access control framework for pervasive edge computing services," in *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Virtual Event, Republic of Korea, 2021, pp. 1405–1420.

[64] Z. Ma, W. Zhang, H. Fang, X. Dong, L. Geng, and N. Yu, "Local geometric distortions resilient watermarking scheme based on symmetry," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 31, pp. 4826–4839, December 2021.

[65] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, Dallas, Texas, USA, 1998, pp. 604–613.

[66] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the 20th Annual Symposium on Computational Geometry (SOCG)*, Brooklyn, New York, USA, 2004, pp. 253–262.

[67] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, Las Vegas, Nevada, USA, 2001, pp. 136–145.

[68] R. Canetti and T. Rabin, "Universal composition with joint state," in *Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, California, USA, 2003, pp. 265–281.

[69] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Proceedings of the 4th Theory of Cryptography Conference (TCC)*, Amsterdam, The Netherlands, 2007, pp. 61–85.

[70] R. Canetti. (2020) Universally composable security: A new paradigm for cryptographic protocols*, https://eprint.iacr.org/2000/067.pdf. [Online]. Available: https://eprint.iacr.org/2000/067.pdf

[71] (2022) Which platforms support intel® software guard extensions (intel® sgx) sgx2?, https://www.intel.com/content/www/us/en/support/articles/000058764/software/intel-security-products.html. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000058764/software/intel-security-products.html

[72] R. Pass, E. Shi, and F. Tramèr, "Formal abstractions for attested execution secure processors," in *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, Paris, France, 2017, pp. 260–289.

[73] A. Boldyreva, D. Cash, M. Fischlin, and B. Warinschi, "Foundations of non-malleable hash and one-way functions," in *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, Tokyo, Japan, 2009, pp. 524–541.

[74] (2024) Full version, https://github.com/zxkcjasdm/Wukong.

[75] Centers for Medicare and Medicaid Services (CMS). (2024) Hospital consumer assessment of healthcare providers and systems (hcahps) - patient survey data, https://data.cms.gov/provider-data/dataset/dgck-syfz. [Online]. Available: https://data.cms.gov/provider-data/dataset/dgck-syfz

[76] Fedesoriano. (2020) Company bankruptcy prediction - bankruptcy data from the taiwan economic journal for the years 1999-2009, https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction. [Online]. Available: https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction

[77] H. Singh. (2023) Supply chain analysis - become a supply chain management pro!, https://www.kaggle.com/datasets/harshsingh2209/supply-chain-analysis. [Online]. Available: https://www.kaggle.com/datasets/harshsingh2209/supply-chain-analysis

[78] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[79] Etherscan. (2024) Ethereum gas tracker, https://etherscan.io/gastracker. [Online]. Available: https://etherscan.io/gastracker

[80] Arbitrum. (2024) Arbitrum gas price tracker, https://tokentool.bitbond.com/gas-price/arbitrum. [Online]. Available: https://tokentool.bitbond.com/gas-price/arbitrum