A Secure, Confidential, and Verifiable Decision Support System

Edoardo Marangone[1], Eugenio Nerio Nemmi[1], Daniele Friolo[1], Giuseppe Ateniese[2], Ingo Weber[3], and Claudio Di Ciccio[4]

[1] Sapienza University of Rome {lastname}@di.uniroma1.it
[2] George Mason University ateniese@gmu.edu
[3] Technical University of Munich & Fraunhofer Gesellschaft ingo.weber@tum.de
[4] Utrecht University c.diciccio@uu.nl

Abstract. Decision support systems are increasingly adopted to automate decision-making processes across industries, organizations, and governments. Decision support demands data privacy, integrity, and availability while ensuring customization, security, and verifiability of the decision process. Existing solutions fail to guarantee those properties altogether. To overcome this limitation, we propose SPARTA, an approach based on Trusted Execution Environments (TEEs) that automates decision processes. To guarantee privacy, integrity, and availability, SPARTA employs efficient cryptographic techniques on notarized data with access mediated through user-defined access policies. Our solution allows users to define decision rules, which are translated to certified software objects deployed within TEEs, thereby guaranteeing customization, verifiability, and security of the process. With experiments run on public benchmarks and synthetic data, we show our approach is scalable and adds limited overhead compared to non-cryptographically secured solutions.

Keywords: TEE · Privacy · Security · Decision support.

1 Introduction

Organizations often need to make decisions collaboratively across trust boundaries while processing confidential data contributed by multiple independent parties. Such tasks are commonly supported by Decision Support System (DSS), which are widely adopted across domains due to their ability to process large volumes of data, provide traceability, and enhance decision accuracy [2,4,24]. However, collaborative decision support creates a fundamental tension: the decision logic must be verifiable, and the process must be auditable. At the same time, the underlying data must remain confidential, including from the platform that executes the computation. A practical solution must satisfy four properties simultaneously: *(i)* confidentiality of input data and decision logic against all parties, including the computation platform; *(ii)* verifiability of the decision process without exposing inputs; *(iii)* user-defined decision rules with fine-grained access control; and *(iv)* practical performance on realistic workloads. No

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

existing approach achieves all four properties. Blockchain-based solutions [16] provide transparency but sacrifice data privacy. Homomorphic encryption and other privacy-preserving approaches [25] preserve confidentiality, but often incur prohibitive overhead, limit expressiveness, and may rely on trusted authorities for key management, thereby introducing single points of failure. Cloud-based platforms [19] offer flexible execution but require trusting a third party with plaintext data. Systems that support decision-process customization are typically designed as modules within centralized information systems, thereby guaranteeing neither privacy nor verifiability [5]. Trusted Execution Environment (TEE) and blockchain combinations, such as TEBDS [29], address some of these concerns, but still depend on centralized key management.

We refer to this fourfold combination as the **confidential verifiable decision-support problem**, which introduces fundamental security challenges in multi-party settings. As a solution to that, we present Secure Platform for Automated decision Rules via Trusted Applications (SPARTA), an approach that leverages TEEs to execute automated decision support in multi-party cooperation scenarios securely. SPARTA executes data-driven decision functions based on custom decision logic inside attested trusted applications, stores encrypted data in a distributed file system, and uses a public blockchain as a notarization layer to guarantee traceability and integrity. Unlike prior approaches that rely on centralized key management, SPARTA derives all keys within the TEE from a shared seed exchanged via mutual attestation, thereby eliminating single points of failure. To express decision logic and access control, we adopt well-established standards, namely Decision Model and Notation (DMN) [15] and Abbreviated Language for Authorization (ALFA).[6] SPARTA considers a threat model in which data providers are honest, the decision requester is potentially malicious, and the TEE provides hardware-level isolation, excluding side channel attacks, which we discuss as a limitation. We provide the following contributions:

1. **Security architecture.** We design a protocol for confidential multi-party decision support that guarantees data privacy, integrity, and forward/backward secrecy by combining TEE-based execution with per-record key derivation and decentralized storage, without requiring a persistent trusted third party beyond initial certification.
2. **Verifiable access-controlled computation.** We present a mechanism that translates user-defined decision logic (DMN) and attribute-based access policies (ALFA) into attested, tamper-proof trusted applications, enabling fine-grained and verifiable authorization over confidential computations.
3. **Security analysis and evaluation.** We formally define the security properties (correctness, data privacy, data integrity, forward/backward secrecy) of the system under a stated threat model, provide a security proof, and empirically demonstrate practical scalability on cross-domain benchmarks with an overhead of less than 80 ms compared to unprotected execution.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12

2 Background

Our approach is built upon the core pillars of confidential computing, and decision logic with access control. **Confidential computing** is a paradigm wherein sensitive data undergoes processing within a protected processing unit [1]. Confidential computing underpins the **Trusted Execution Environments (TEEs)** [26]. A TEE satisfies *(i) data confidentiality*: unauthorized entities cannot access data residing within the TEE. *(ii) data integrity*: unauthorized entities cannot modify, delete, or insert data inside the TEE. *(iii) code integrity*, unauthorized entities cannot manipulate, delete, or introduce alterations to the code TEE. Our work leverages Intel Software Guard Extensions (SGX) [12], facilitating the secure execution of code and data within *enclaves*, namely hardware-protected environments. SGX further enables *remote attestation*, allowing external entities to authenticate a target enclave. One of the most well-known languages to model and formalize **decision logic** in business applications is Decision Model and Notation (DMN) [23], a standard by the Object Management Group (OMG)[5] for modeling and formalizing decision logic in multi-party processes and business applications. A Decision Model and Notation (DMN), is a decision table that captures critical knowledge and automates decision logic [9]. A DMN table consists of columns representing the inputs and outputs of a decision, and rows denoting rules (the decision logic). Rules are expressed using Friendly Enough Expression Language (FEEL), a language defined within the DMN standard. Each row has an identifier, a condition for each input column, and one specific value for each output column. A rule is considered applicable when the input values satisfy all associated conditions; in that case, the corresponding output values are produced [9]. To express **authorization policies** over the usage of resources, the ALFA[6] language provides a concise, human-readable format. Built on top of the eXtensible Access Control Markup Language (XACML),[7] ALFA offers a simplified syntax for defining fine-grained, attribute-based access control (ABAC) policies based on users, actions, resources, and context.

In addition, our architecture employs two additional buttresses for the notarization and storage of operations and data. **Blockchains** are Distributed Ledger Technologies (DLT), wherein transactions are sorted, organized into blocks, and interconnected to form a chain possessing resistance against tampering, achieved through cryptographic methods. Public blockchains like Ethereum and Algorand permit the deployment and execution of programs named smart contracts [14]. To mitigate the expenses associated with the invocation of smart contracts, external Peer-to-peer (P2P) systems are frequently employed for storing large volumes of data [30], such as **InterPlanetary File System (IPFS)** [6],[8] a distributed system designed for file storage and access.

[5] omg.org/spec/DMN. Accessed: 2026-04-09 [6] alfa.guide/alfa-authorization-language/. Accessed: 2026-04-09 [7] docs.oasis-open.org/xacml. Accessed: 2026-04-09 [8] ipfs.tech. Accessed 2026-03-19.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

3 Problem illustration and requirements

To illustrate the problem we address, we introduce a real-world scenario from the healthcare domain: a vaccine distribution campaign inspired by [18,22] and linked to a research project funded by the European NextGenerationEU.[9] We use this scenario as a running example and motivating case study throughout the paper. In our scenario, a central medical hub (hereafter, medical hub) receives vaccine supplies and manages their dispatch within a federal State; patients receive vaccine injections, vaccination centers administer them, and specialized carriers deliver the vaccines on-site.

In our scenario, we distinguish the following user *roles*: medical hub, patient, vaccination center, and carrier. A role is one of the *attributes* that characterize a user. In the first stage of our running example, users belonging to the *Data Provider* group contribute data to the system; we refer to this stage as *data provision*. Table 1 reports examples of user roles, groups, and attributes, while Table 2 shows part of the data that these users provide to the system. For instance, the National vaccine hub, which serves as a medical hub in Italy, reports vaccine stock levels and storage temperatures. Andre Smith, a patient from Sardinia, submits vaccination requests together with personal information, such as pre-existing conditions and family medical history. Ayala PLC, a vaccination center in Tuscany, reports storage capacity and vaccination progress. PrimeWay, a carrier, registers service-related information such as the number of trucks and refrigeration capacity.

In the subsequent *data usage* stage, authorized users can access the data provided earlier, either at the level of single entities or in aggregated form. In our example scenario, patients can retrieve their declarations at any time, and carriers can obtain information about on-time delivery rates. Users in the *Decider* group can also request automated decision support. The medical hub, e.g., belongs to this group and can request decision support for vaccine supply distribution. Decisions related to vaccine distribution include *(i)* the priority to be assigned to patients for vaccine administration (which we use as a running example throughout this paper), *(ii)* the restocking needs of each vaccination center, and *(iii)* the carrier that should handle each delivery.

In our example, several critical aspects need to be considered. First, all data is confidential and must be protected from unauthorized access. For example, revealing a single patient's metadata constitutes a privacy breach. Second, data

[9] SERICS (PE00000014), funded by the Italian Ministry of University and Research.

Table 1: Examples of users, groups, and attributes

User	Group	Attributes
Andre Smith	Data Provider	`Role="Patient"`, `Region="Sardinia"`
National vaccine hub	Data Provider, Decider	`Role="CentralMedicalHub"`, `Country="Italy"`
Ayala PLC	Data Provider	`Role="VaccinationCenter"`, `Region="Tuscany"`
PrimeWay	Data Provider	`Role="Carrier"`, `Area="North"`

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12

Table 2: An example of the data fields users provide to the system

Central medical hub			**Vaccination center**		
Field	Type	Value	Field	Type	Value
Stock ID	Integer	`10523`	Hub Name	String	`Ayala PLC`
Quantity Available	Integer	`2400`	Max Storage Capacity	Integer	`2000`
Storage Temperature	Float	`-20.5` [°C]	Vaccination Progress	Integer	`686` [per day]

Patient			**Carrier**		
Field	Type	Value	Field	Type	Value
Full Name	String	`Andre Smith`	Carrier Name	String	`PrimeWay`
Pre-existing Conditions	String	`Asthma`	Number of Trucks	Integer	`13`
Family Medical History	String	`Heart Disease`	Refrigeration Capacity	Boolean	`True`

Table 3: Requirements and corresponding actions

	Requirement	**Approach**
R1	The stored **information** shall be tamper-proof, traceable, confidential, permanent, and available	We store the encrypted data in a tamper-proof distributed file storage, and we use a notarization system to keep track of information
R2	**Decision support** shall be user-definable, trusted, reproducible, verifiable, and confidential	We handle data within trusted hardware, encrypting them before being stored outside of it; data access and processing is established through customizable policies, turned into verifiable apps running in trusted hardware
R3	The decision support **system** must be secure, interoperable, available, replicable, and scalable	We employ standard languages to describe authorization policies (expressing what each user can do and what resources they may access) and decision logic (the rules to follow in making decisions)

integrity and availability are critical. Any unauthorized modification could alter decisions or falsify patient records, while system failures or data unavailability could disrupt the decision-making process. Ensuring privacy, integrity, and availability while enabling customizable, secure, and verifiable automated decision-making is an open challenge that motivates our research.

Table 3 presents the requirements identified from the motivating use case that underpins our approach. Ensuring that decision-critical information is consistently available, tamper-evident, and auditable is essential to both the efficiency of healthcare processes and trust in the system. The information source must be immutable and accessible, with traceability ensured to facilitate subsequent validations and verifications. We formalize these needs in Req. R1. Since the system manages sensitive and confidential healthcare data, its security is crucial to ensuring accurate decisions and preventing unauthorized access, data leaks, malicious attacks, and system unavailability. For example, disclosing patients' metadata or their vaccination prioritization would violate their privacy. We enclose these properties in Req. R2. Finally, our approach must be interoperable with existing data storage, access, and aggregation solutions while enabling se-

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12

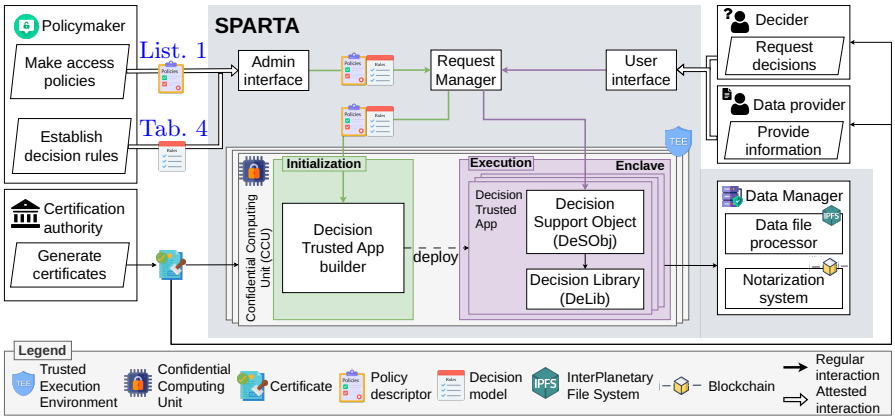


Fig. 1: An overview of the SPARTA approach with the software components, user roles, and main information flow

cure, automated decision-making for confidential data. We consider these aspects in Req. R3. Our solution aims to meet these requirements.

4 Approach

In this section, we present our approach, named Secure Platform for Automated decision Rules via Trusted Applications (SPARTA). In our description, we motivate design choices and operations with the previously discussed requirements (see Sect. 3 and Tab. 3 above). To this end, we first provide an overview of its architecture (Sect. 4.1). Then, we describe how we initialize the system (Sect. 4.2) and thereby execute secure decision processes over confidential data (Sect. 4.3). Finally, we outline a security analysis of our approach (Sect. 4.4).

4.1 Basic components and actors

Fig. 1 provides a bird's eye view of the SPARTA approach. The **Confidential Computing Unit (CCU)** is the core component of SPARTA. It is deployed inside a TEE and can be replicated to improve data availability and system robustness, as required by Reqs. R1 and R2. Dedicated Decision Trusted App (DeTApp)s execute a decision process each within their own enclave inside the CCU. A DeTApp builder encodes and deploys them based on two inputs: *(i)* a *decision model* (see Tab. 4), which defines the functions exposed by the DeTApp, and *(ii)* a *policy descriptor* (see List. 1), which specifies who can invoke those functions. More specifically, those inputs customize what we call a Decision Support Object (DeSObj). Internally, the DeTApp links the DeSObj with a Decision Library (DeLib), namely a shared library we developed for maintainability and encapsulation purposes. It provides common utilities for DeSObjs, such as data

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

retrieval and decryption, policy checking, and certificate validation. The **Request Manager** acts as a façade to the CCU. It can be abstracted as a queue that handles both requests to and responses from the CCU. Its purpose is to save them and preserve their order while the CCU is handling other requests, thereby avoiding concurrency issues. The Request Manager operates outside the TEE, allowing the DeTApp to perform security-critical operations inside the enclave while avoiding overuse of protected memory. Consequently, all data exchanged through the Request Manager must be encrypted, as we explain in Sect. 4.3. The Request Manager serves two entry points: one for system-initialization operations (Admin Interface), such as certificate management and the submission of decision models and policy descriptors, and one for runtime interactions (User Interface), through which Data Providers and the Decider communicate with the CCU. The **Data Manager** is an external component providing two architectural buttresses: *(i)* The Data File Processor, which stores the content of the exchanged data, and *(ii)* the Notarization System, which saves the resource locators pointing to that data. We realize these components through two established technologies to meet Req. R1. We employ InterPlanetary Name System (IPNS) over IPFS for the Data File Processor since it provides permanence, immutability, and content addressing for stored files. We use a public blockchain as the Notarization System to store IPNS names in a tamper-proof, available, and traceable way. Other solutions like STORJ[10] and distributed relational databases [31] may be used in scenarios with less restrictive security requirements.

We identify four main actors, each responsible for a distinct functionality. The **Certification Authority (CA)** is the trusted authority that issues digital certificates to users, certifying their *attributes*. In our example, the role of the Certification Authority (CA) should be fulfilled by a globally recognized Certification Authority (e.g., *DigiCert*, *Entrust*, or *GlobalSign*),[11] akin to what is offered by the National Health Service in the United Kingdom [13]. The **Policymaker** defines the decision model and the policy descriptor needed for the deployment of the DeTApp. Considering the example in Sect. 3, the Policymaker role can be fulfilled by a regulatory body such as the Ministry of Health, or by the consortium directly involved in the decision process, as in [10], upon reaching consensus. The **Data Provider** provides the input data and contextual information used for decision-making. In our running example, the Data Provider group comprises the roles medical hub, patient, vaccination center, and carrier. Finally, the **Decider** receives decision support based on the available data. If permitted by access policies, the Decider may also retrieve raw data and invoke aggregation functions. In our scenario, the medical hub role acts as a Decider.

4.2 System initialization

The system initialization phase goes through three consecutive steps: *(i)* Certification (CERT), *(ii)* CCU initialization (CCU-INIT); *(iii)* Specification of functions and policies (SPEC). We detail them in the following.

[10] storj.io. Accessed: 2026-03-19. [11] DigiCert: digicert.com; Entrust: entrust.com; GlobalSign: globalsign.com. Accessed: 2026-04-07.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

Certification (CERT). In this step, the CA certifies each actor's attributes by issuing a certificate. For our purposes, we consider X.509 certificates. Each certificate contains the actor's metadata, including attributes and public key. For the latter, we leverage Elliptic Curve Digital Signature Algorithm (ECDSA) [21]. Before any interaction, each participant must verify the counterpart's certificate (see Req. R3). The CCU must likewise hold a valid certificate that allows other actors to confirm they are interacting with the intended CCU (see Req. R2). To sign messages, actors use the ECDSA private key corresponding to the public key attested to in their certificate; otherwise, the signature is invalid.

A technical note on the remote attestation (RA). Before making any request, a client (a Data Provider, Decider, or another CCU) performs remote attestation to establish trust in the CCU inside the TEE (see Req. R2). In this phase, we follow the RATS RFC standard [8], the foundation for attestation schemes such as Intel EPID and AMD SEV-SNP. Remote attestation allows the client to verify that they are sending data to a Trusted Application within a TEE and to authenticate the DeSObj as a legitimate CCU entity certified by the CA (see above). We slightly modify the standard procedure. The client: *(i)* retrieves and verifies the CCU's CA-issued certificate using the CA's ECDSA public key, confirming the CCU is the intended party; *(ii)* retrieves and validates the CCU's Transport Layer Security (TLS) certificate (including its ECDSA public key) to establish a secure connection; and *(iii)* obtains and verifies the CCU's attestation report over the attested TLS channel. The report contains the *measurement* of the DeTApp, i.e., the hash of its code and data, signed with the CCU's hardware attestation key. The client validates it by authenticating the CCU, decrypting the report, comparing the reported measurement with the expected reference value, and verifying that the report is consistent with the server's TLS certificate.

CCU initialization (CCU-INIT). This step sets up the replicated CCUs and the secure communication with them. It encompasses two phases: **Seed exchange (SDEX)** and **Communication channel configuration (CONF)**. **SDEX** addresses the following two issues. Relying on a single CCU introduces significant vulnerabilities that could compromise Reqs. R1 and R2. Should the component fail, experience downtime, or be subject to malicious attacks, the entire execution process may halt indefinitely, and recovery may become impossible. Also, if that CCU alone held the cryptographic keys, all associated data would be permanently lost. To address these issues, SPARTA's CCUs share a common seed, enabling all replicas to deterministically derive the necessary cryptographic keys (meeting Req. R2). To describe how SDEX unfolds, we shall use an example with two CCUs: CCU1 and CCU2. CCU1 generates and sends the seed. It proceeds as follows: *(i)* it performs the remote attestation of CCU2; *(ii)* if remote attestation (RA) succeeds, it generates a secure random string (the seed), seals it with a sealing key known only to itself (see Sect. 2), and sends the seed to CCU2 through the secure channel established during RA. After receiving the seed from CCU1, CCU2 performs the same operations: it remotely attests CCU1 and, if successful, seals the seed with a sealing key known only to itself. Once exchanged, the seed can be shared with an arbitrary number of additional

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12

CCUs. We describe how it is used to derive encryption keys in Sect. 4.3. The subsequent phase (**CONF**) generates a TLS certificate. This certificate serves two purposes: It enables the establishment of a secure communication channel with the CCU during remote attestation, and provides the client with the shared secret needed to encrypt data before sending it through the Request Manager. Since the Request Manager operates outside the TEE (see Sect. 4.1), all data exchanged with the system must be encrypted. To do so, we slightly modify the TLS certificate that each CCU generates, allowing it to embed a public key deterministically generated from the shared seed (see SDEX). Since all CCUs share the same seed, they produce identical key pairs. We leverage Elliptic-Curve Diffie-Hellman (ECDH) [3] (X25519 curve [7]) for the key pair generation. The CCU signs the TLS certificate with its ECDSA private key, which is CA-certified (see CERT), allowing any participant to verify it. At the end of the CONF phase (and hence of the CCU-INIT step), the CCU is ready to receive requests and provide responses through the Request Manager.

Specification of functions and policies (SPEC). With this step, the Policymaker specifies the decision functions the system should realize in a decision model, and their access policies in a policy descriptor. To meet Req. R3, the Policymaker expresses these using standard, well-established, and customizable languages: DMN with FEEL and ALFA, respectively (see Sect. 2). Their details transcend the scope of this paper, and illustrate their use by example here. Table 4 shows a DMN table for prioritizing vaccine requests by patients. The *source data* input columns refer to patient data provided in the *data provision* stage in Sect. 3. DMN also allows the specification of FEEL expressions that filter and then aggregate data. For example, the `StockCoverageDays` input column is a FEEL expression that calculates the number of days the currently available vaccine stock can sustain, based on vaccination center activity and vaccination progress. The corresponding FEEL expression is:

```
{ totalAvailableVaccines: sum(medicalHub.QuantityAvailable),
  activeCenters: vaccinationCenters[VaccinationProgress > 0],
  totalDailyVaccinations: sum(activeCenters.VaccinationProgress),
  stockCoverageDays: totalAvailableVaccines / totalDailyVaccinations }.stockCoverageDays
```

The *decision* output column shows the corresponding decision. Listing 1 shows

Table 4: Decision model for the patient prioritization of Sect. 3

When	and	and	and	and	and	and	then
Age	**Pre Existing Conditions**	**Current Medications**	**Previous Vaccinations**	**Family Medical History**	**Consent Form Signed**	**Stock Coverage Days**	**Patient Priority WAggr**
:number	:string	:string	:string	:string	:boolean	:number	:string
≥60	"Asthma", "Diabetes"	"Metformin", "Albuterol"	"COVID-19", "Influenza"	"Diabetes", "Heart Disease"	true	≤3	"High"
[18..60[	"Asthma"	"Metformin"	"Influenza"	-	true	]3..7]	"Medium"
<18	-	"Lisinopril"	"COVID-19"	-	true	>7	"Low"
-	-	-	-	-	false	-	"Ineligible"
source data						*FEEL expr.*	*decision*

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

```
namespace VaccineDispatch {
  condition isMedicalHub Role=="CentralMedicalHub" and Country=="Italy"
  policy PatientPriorityWAggr {
    target clause Action=="PatientPriorityWAggr(Patient,stockCoverageDays)"
    rule accessDecisionWAggr { condition isMedicalHub } } }
```

Listing 1: An excerpt of a policy descriptor file in ALFA

an excerpt of the ALFA descriptor file.[12] In our running example, only a medical hub located in Italy may invoke the `PatientPriorityWAggr` function, which determines the priority level assigned to a patient for vaccine administration based on patient data and the aggregated `StockCoverageDays` input. The `accessDecisionWAggr` rule enforces this restriction. Finally, the DeTApp builder compiles the ALFA policy and the DMN table into a DeSObj, which is deployed as a DeTApp within the TEE enclave. Thereafter, the Decider can invoke the decision function. As we describe next, the DeSObj makes a decision within the enclave, so the invoker reads the result but not the input data.

4.3 Execution

The lilac box in Fig. 1 encloses the components involved in the execution phase of SPARTA. It is comprised of a **writing stage (WS)**, wherein data is provided in an encrypted form, and the **decision-making stage (DECM)**, in which automated decisions are made. These stages correspond, respectively, to the *data provision* and *data usage* stages of the running example in Sect. 3.

Client encrypted writing (WS-i). In this step, the Data Provider encrypts the data before transmitting it through the Request Manager, which operates outside the trusted environment. Since multiple CCUs are possible, the Data Provider needs to encrypt the data with a key known to all CCUs. To achieve this, the Data Provider retrieves a CCU's TLS certificate (the one generated during the CONF phase), and if the remote attestation completes successfully, derives the shared secret from it and generates the symmetric key required to encrypt the data, thereby meeting Req. R1. Upon receiving the request, the CCU verifies the Data Provider's certificate and signature, derives the same shared secret, reconstructs the symmetric key, and decrypts the payload, thus meeting Req. R2. This mechanism relies on an ECDH key exchange over the X25519 curve [3,7], Secure Hash Algorithm (SHA)-256 [17], and Advanced Encryption Standard - Galois/Counter Mode (AES-GCM) [20].

CCU encrypted writing (WS-ii). This step begins after the CCU decrypts the Data Provider's input data. The CCU must encrypt the data before storing it outside the enclave, as it would otherwise be accessible in plaintext. To guarantee this, the CCU generates a distinct symmetric key for each record from the shared seed established during SDEX and a newly generated random number. Thereafter, the CCU encrypts the data with this key using AES-GCM to meet

[12] Full descriptor: github.com/apwbs/SPARTA/blob/main/src/alfaparser/policy.alfa

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

Req. R2. It stores the ciphertext on IPFS and records the required IPNS names in a smart contract on the public blockchain in compliance with Req. R1. The random number is stored in plaintext alongside the encrypted data on IPFS; this does not compromise security, since the encryption key depends on both the random number and the private seed. Consequently, an attacker cannot derive valid keys without access to the seed. Indeed, without a valid CA certificate, an attacker cannot participate in SDEX, while the seed itself remains protected by a sealing key. Finally, we remark that if a CCU becomes unavailable, any other CCU that participated in SDEX can take over, as it possesses the same seed.

Decision-making support (DECM). In this step, the Decider requests decision support. To do so, it first performs remote attestation against the CCU. Then it invokes a decision function exposed by the DeSObj deployed within it (e.g., `PatientPriorityWAggr`), thereby meeting Req. R2. The request includes the Decider's certificate issued by the CA, the name of the invoked function, and the IPNS key name identifying the relevant data. After parsing the request, the CCU verifies the Decider's certificate and checks whether the certified attributes satisfy the access policy associated with the invoked function, as required by Req. R3. In our running example, this policy requires the invoker to be a medical hub located in Italy. If the policy is satisfied, the CCU retrieves the ciphertext from IPFS, recovers the symmetric key, decrypts the data, evaluates the required aggregation functions, and executes the decision function within the enclave, thereby preserving confidentiality and integrity in compliance with Req. R2. In our example, this amounts to computing the `StockCoverageDays` aggregation before evaluating `PatientPriorityWAggr`. Finally, the CCU returns the result to the Decider through the Request Manager.

4.4 Security Analysis

To examine the security of our system, we first provide a threat model and the desired properties we aim to achieve.

Threat Model. Data Provider and CA are honest, the Decider is dishonest and secure authenticated channels can be established between authorized parties (e.g., CA, Decider) and the CCU. We model the CCU as a fully trusted third-party computing decisions through functions $\{f_j\}$ using a secret seed s.

Desired Properties. We informally specify the following required properties:

Correctness. For any function f and any set of messages M such that $f(M) = y$, if M is provided to the CCU for function f by the Data Provider obtaining ciphertexts C, and subsequently, ciphertexts C are provided by the Decider back to the CCU obtaining output y', then $y' = f(M) = y$.

Data Privacy. An adversary cannot distinguish between the encryption of two messages giving the same output for an adversarially chosen function f.

Data Integrity. An adversary cannot tamper with encrypted messages.

Forward/Backward Secrecy/Integrity. If all ciphertext keys at time t are leaked, ciphertexts encrypted at any different time t' remain secure, and their underlying messages cannot be decrypted or forged.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

Theorem 1 (Informal). *Assuming that the authenticated encryption scheme is secure, the hash function is modeled as a random oracle, and the TEE is secure, our system satisfies Correctness, Data Privacy, Data Integrity, and Forward/Backward Secrecy and Integrity.*

The theorem follows from the security of the TEE and the indistinguishability and untamperability properties of the underlying authenticated encryption scheme. For the complete formal analysis and proof, see Appendix A.

5 Implementation and evaluation

Experimental Setup. We implemented SPARTA in Go and used EGo to deploy trusted applications.[13] We implemented the Request Manager as a Redis queue, used *govaluate* to evaluate FEEL expressions and access policies, and *Expr* to evaluate aggregation functions.[14] We ran all experiments on a TEE hosted by an Intel Xeon Gold 5415+ CPU with 128 GB of RAM, Ubuntu 22.04, and Intel SGX enabled with a PRMRR size of 2 GB. Each experiment was repeated 10 times, and we report mean values. Our implementation's source code, together with input data and outputs, is available at github.com/apwbs/SPARTA.

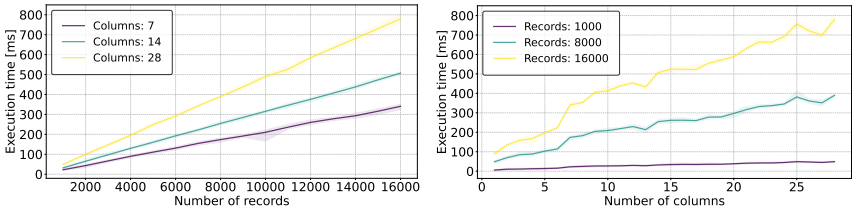
Four questions drive our evaluation: (**Q1**) Does SPARTA scale well with input size? (**Q2**) How much overhead encryption and TEE execution introduce compared to an unprotected baseline running on the same hardware? (**Q3**) Can SPARTA operate with decision models across different domains? (**Q4**) How effectively does the dual-storage system reduce memory consumption?

To answer these questions and test the distinct features of our system, we resort to the following three datasets. **MED (real-world medical dataset)** is the Decision Management Community challenge's dataset (*Determine Medical Service Coverage*, released in Feb. 2021). It contains 16 000 healthcare-related records. We use it for scalability and overhead measurement because of its large size. **CARD (real-world financial dataset)** is the *Card Approval Decision* challenge's dataset (Oct. 2021). Through it, we evaluate cross-domain performance with diversified and complex predicates. **VAX (synthetic vaccine dataset)** is a synthetic vaccine-distribution dataset we built based on our running example. In response to Q1, we use the MED dataset to analyze scalability based on the number of input records, columns, and rules. We address Q2 through a controlled comparison measuring encryption and TEE overhead against a non-encrypted baseline running on the same machine. We demonstrate cross-domain performance (Q3) with the CARD dataset. We use the VAX dataset to answer Q4 measuring the dual-storage memory optimization.

Results. MED. The original dataset contains 16 369 rules, with 7 input and 3 output columns. To increase the number of columns, we artificially replicated the input columns to reach a total of 28. Although artificial, this setting allows us

[13] EGo: edgeless.systems/products/ego; Intel SGX: sgx101.gitbook.io/sgx101. Accessed: 2026-03-19. [14] Redis queue: github.com/rq/rq; govaluate: github.com/Knetic/govaluate; expr: github.com/expr-lang/expr. Accessed: 2026-03-19.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12



(a) With 7, 14, and 28 columns (b) With 1 k, 8 k, and 16 k records

Fig. 2: Decision execution time vs. number of records (a) and columns (b)

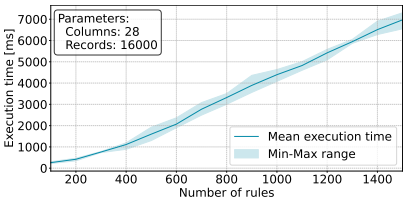


Fig. 3: Runtime vs. number of rules

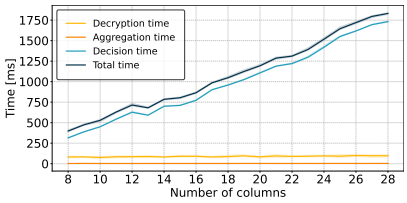


Fig. 4: Runtime with prior aggregation

to evaluate system performance under higher-dimensional input across multiple parameters (i.e., rules, columns, and records). Unless specified, all experiments below used 300 randomly selected rules from the original benchmark. *Scalability.* Fig. 2 shows execution time while increasing the number of records and columns. We vary the number of records from 1000 to 16 000 with 7 , 14 , and 28 columns, and then vary the number of columns from 1 to 28 with 1000, 8000, and 16 000 records. In both cases, the growth is linear. Fig. 3 shows the same trend when increasing the number of rules from 100 to 1500 with 16 000 records and 28 columns, representing the most demanding configuration tested.

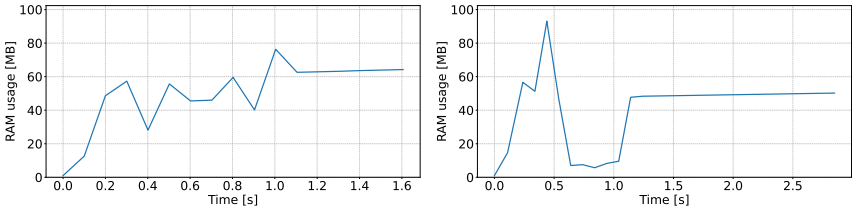
Aggregation overhead. To measure the cost of aggregation, we augment the input with up to 21 columns, each of which computes an aggregation function over different input fields. In particular, Fig. 4 shows that decryption and aggregation add extremely limited overhead to the decision evaluation execution time, and that this overhead grows almost linearly.

Memory usage. Fig. 5 reports enclave memory usage with and without the aggregation step. Each run begins with the CCU's initialization and ends when the decision process terminates. Both images show a reasonable memory footprint throughout the computation, with an expected peak during decryption.

The previous three experiments address Q1 by demonstrating linear scalability with respect to input size, number of columns, and rule complexity, and also bound memory usage within the TEE. Next, we address Q2 by quantifying the overhead introduced by encryption and TEE execution.

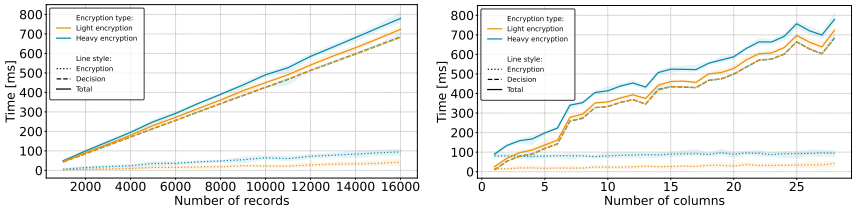
Encryption overhead. We compare a light configuration in which all records share a single symmetric key with the actual heavy configuration, in which each record is encrypted with a different key (see WS-ii stage). Note that the light configu-

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12



(a) Without prior aggregation (b) With prior aggregation

Fig. 5: Memory usage over time

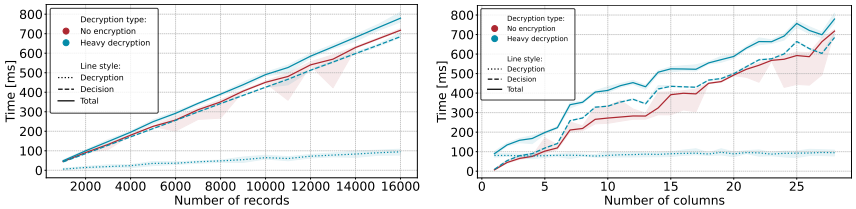


(a) Records for 28 fixed columns (b) Columns for 16 k fixed users

Fig. 6: Comparison between light and heavy encryption

ration is used only as a baseline; from a security perspective, compromising that key would expose the entire dataset, whereas in the heavy one, a key compromise affects at most one record. Fig. 6 shows that the heavy configuration adds a reasonable overhead of at most 80 ms across all tested configurations, while providing higher data confidentiality and resilience against key compromise.

TEE vs. server. We compare SPARTA's performance against an unprotected baseline that executes the same logic outside a TEE (on the same machine) and without encryption. Fig. 7 shows that the additional cost of fully protected execution is limited and mainly attributable to the decryption phase and enclave execution rather than to the decision logic itself. The gap between the two configurations remains stable as the number of records and columns increases, indicating that the overhead does not compound with input size.



(a) Comparison for 28 columns (b) Comparison for 16 k records

Fig. 7: Execution time comparison between server and TEE

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12

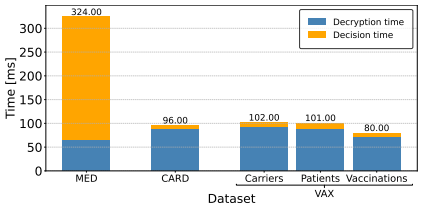


Fig. 8: Execution time with VAX, MED, and CARD datasets

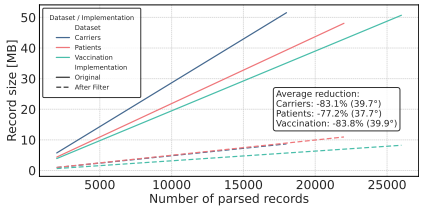


Fig. 9: Memory consumption with the dual-storage strategy

CARD. This benchmark contains 3 input columns and 13 rules with relational and interval-based conditions from the financial domain (to cater to Q3), featuring more complex predicates than those in MED and VAX. Since the original challenge does not provide input data, we generated inputs covering all rules and duplicated them up to 100 000 records. Fig. 8 shows the execution times across the three datasets. MED requires the highest decision time due to its larger rule set. In contrast, the decryption time remains consistent across the datasets, confirming SPARTA's ability to maintain stable performance across domains with varying logic complexity and input scale.

VAX. We built the dataset by generating realistic input examples for a nationwide vaccine campaign for patients, vaccination centers, and carriers, based on the decision logic in DMN (see Tab. 4). We use this synthetic scenario to evaluate the dual-storage strategy introduced in the workflow (see the WS stage in Sect. 4.3) as an answer to Q4. Each patient provides full records, but only a subset of the fields is required for decision making: 6 out of 38 fields for patients, 8 out of 50 for carriers, and 4 out of 35 for vaccination centers. To enhance privacy and efficiency, we store the lightweight version containing only the decision-relevant fields, alongside the original one for traceability. Notice that this dual-storage strategy allows the Data Provider to submit data without knowledge of the decision-critical fields, preventing inference attacks and further preserving logic confidentiality. Fig. 9 displays a memory saving in the range of 77.2 % to 83.8 %.

Finally, to assess scalability beyond the original benchmark size, we conducted an additional stress test using the VAX dataset with the decision logic of Tab. 4 and up to 1 000 000 patient records as input. SPARTA completed decision execution in approximately 0.5 s, confirming that the linear scaling trend extends well beyond the original dataset.

6 Related Work

Lately, numerous approaches have been proposed in the decision-making support systems field. Haarmann et al. [16] propose executing DMN decision models on the Ethereum blockchain. However, approaches that rely on public ledgers are not suitable for confidential data. In contrast, SPARTA enables secure, private decision support while maintaining trust, transparency, and auditability, with

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

access policies that ensure only authorized participants can interact with the system. Zhang et al. [32] introduce a privacy-preserving blockchain-based decision-making system for secure voting, proposing a distributed batch key-generation protocol and a two-stage voting scheme. While this addresses blockchains' privacy issues, it is tailored to voting only, whereas SPARTA supports user-definable decision logic. Other solutions [25] enable privacy-preserving clinical decisions using homomorphic encryption. While effective for automated classification, these approaches incur significant performance overhead and fix the decision logic. In contrast, SPARTA uses AES-GCM encryption, which introduces low overhead and allows for custom decision logic. Liu et al. [19] present HPCS, a hybrid privacy-preserving clinical decision support system for health monitoring and disease prediction, in which a cloud server runs neural network predictions. Compared to SPARTA, HPCS incurs higher overhead and limits decision logic to prediction-based decisions, whereas we support user-defined logic. Xie et al. [29] present TEBDS, a TEE-based blockchain data-sharing system for IoT that relies on a centralized key management center. In contrast, SPARTA derives all keys within the TEE from a shared seed, removing this single point of failure. Moreover, SPARTA uses a public blockchain only to store IPNS names pointing to IPFS content-addressed data, delegating all operations to the TEE.

7 Conclusion, limitations and future work

We presented SPARTA, an approach that enables automated decision-support over confidential data in multi-party settings while preserving data secrecy. SPARTA combines TEE-based execution with per-record key derivation, distributed storage, and blockchain notarization to solve the confidential verifiable decision-support problem, enabling multi-party decision-making that is confidential, verifiable, customizable, and practical. Our experiments on real-world and synthetic benchmarks confirm linear scalability and an overhead of less than 80 ms compared to unprotected execution.

Our solution presents some limitations. SPARTA supports CCU replication but does not yet implement load balancing, which would require non-trivial changes to session management. Our prototype targets Intel SGX; porting to other TEE platforms such as AMD SEV, Arm CCA, or Intel TDX is a natural next step. Our threat model also excludes side-channel attacks [27], though existing mitigation techniques [28] and deployment within protected environments can reduce this risk. Our implementation covers only standard aggregation functions (`avg`, `sum`, `max`, `min`) and single-operation filters over data. Integrating TEE-based smart contracts [11] could further broaden the range of automated decision logic that the system supports. Finally, although our experiments with public benchmarks show good performance and scalability with low overhead, on-field use of SPARTA in real-world settings is paramount for validation.

Acknowledgments. This work was partly funded by projects SERICS (PE00000014), under the NRRP MUR program funded by the EU, and ASGARD, funded by Sapienza University of Rome under grant B83C23005800001.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

References

1. Confidential Computing: Hardware Based Memory Protection. Springer Nature Singapore (2022)
2. Arnott, D., Pervan, G.: Eight key issues for the decision support systems discipline. Decision Support Systems **44**(3), 657–672 (2008)
3. Barker, E., Chen, L., Roginsky, A., et al.: Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Tech. rep. (2018)
4. Barracosa, B., Felício, J., Carvalho, A., et al.: Decision support tool to define the optimal pool testing strategy for SARS-CoV-2. Decis. Support Syst. **175** (2023)
5. Bazhenova, E., Zerbato, F., Oliboni, B., Weske, M.: From BPMNprocess models to DMN decision models. Information Systems **83**, 69–88 (2019)
6. Benet, J.: IPFS - content addressed, versioned, P2P file system (2014)
7. Bernstein, D.J.: Curve25519: New diffie-hellman speed records. In: Public Key Cryptography. pp. 207–228. Springer (2006)
8. Birkholz, H., Thaler, D., Richardson, M., et al.: Remote attestation procedures (RATS) architecture. RFC **9334**, 1–46 (2023)
9. Calvanese, D., Dumas, M., Laurson, Ü., et al.: Semantics and analysis of DMN decision tables. In: BPM. pp. 217–233. Springer (2016)
10. Cerulli, A., Connolly, A., Neven, G., Preiss, F., Shoup, V.: vetKeys: How a blockchain can keep many secrets. IACR Cryptol. ePrint Arch. p. 616 (2023)
11. Cheng, R., Zhang, F., Kos, J., et al.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: EuroS&P. pp. 185–200 (2019)
12. Costan, V., Devadas, S.: Intel SGX explained. Cryptology ePrint Archive, Paper 2016/086 (2016), https://eprint.iacr.org/2016/086
13. Dalton, C.R.: The NHS as a proving ground for cryptosystems. Inf. Secur. Tech. Rep. **8**(3), 73–88 (2003)
14. Dannen, C.: Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. Apress (2017)
15. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
16. Haarmann, S., Batoulis, K., Nikaj, A., Weske, M.: DMN decision execution on the ethereum blockchain. In: CAiSE. pp. 327–341. Springer (2018)
17. Hansen, T., 3rd, D.E.E.: US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF) (2011)
18. Lim, J., Norman, B.A., Rajgopal, J.: Redesign of vaccine distribution networks. International Transactions in Operational Research **29**(1), 200–225 (2022)
19. Liu, X., Deng, R.H., Yang, Y., et al.: Hybrid privacy-preserving clinical decision support system in fog–cloud computing. Future Gener. Comp. Sy. **78**, 825–837 (2018)
20. McGrew, D.A., Viega, J.: The security and performance of the Galois/Counter Mode (GCM) of operation. In: INDOCRYPT. pp. 343–355. Springer Berlin Heidelberg (2005)
21. National Institute of Standards and Technology: Digital signature standard (DSS). Tech. rep. (2023)
22. Noh, E.B., Nam, H.K., Lee, H.: Which group should be vaccinated first?: a systematic review. Infection & chemotherapy **53**(2), 261 (2021)
23. OMG: Decision model and notation (DMN) v1.5 (2024). Object Management Group **1**(4), 18 (2011), https://www.omg.org/spec/DMN

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

24. Pereira, A.M., Moura, J.A.B., Costa, E.D.B., et al.: Customer models for artificial intelligence-based decision support in fashion online retail supply chains. Decis. Support Syst. **158**, 113795 (2022)
25. Rahulamathavan, Y., Veluru, S., Phan, R.C.W., et al.: Privacy-preserving clinical decision support system using Gaussian kernel-based classification. IEEE J Biomed. Health **18**(1), 56–66 (2014)
26. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: TrustCom 2015. pp. 57–64. IEEE (2015)
27. Wang, W., Chen, G., Pan, X., et al.: Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In: CCS. pp. 2421–2434 (2017)
28. Wichelmann, J., Rabich, A., Pätschke, A., Eisenbarth, T.: Obelix: Mitigating side-channels through dynamic obfuscation. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4182–4199 (2024)
29. Xie, H., Zheng, J., He, T., Wei, S., Hu, C.: TEBDS: A trusted execution environment-and-blockchain-supported IoT data sharing system. Future Generation Computer Systems **140**, 321–330 (2023)
30. Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications. Springer (2019)
31. Yang, Z., Yang, C., Han, F., et al.: Oceanbase: a 707 million tpmC distributed relational database system. Proc. VLDB Endow. **15**(12), 3385–3397 (2022)
32. Zhang, J., Zhang, B., Nastenko, A., Balogun, H., Oliynykov, R.: Privacy-preserving decision-making over blockchain. IEEE Transactions on Dependable and Secure Computing **20**(6), 4648–4663 (2023)

A Formal Analysis

We first introduce the required notation and cryptographic tools.

Notation. Let $\mathbb{N}$ be the set of all natural numbers. We denote the security parameter by $\lambda \in \mathbb{N}$. Every algorithm takes the security parameter λ as input (in unary, i.e., 1^λ). When an algorithm has multiple inputs, 1^λ is typically omitted. A function is called negligible if, for every positive polynomial and all sufficiently large security parameter λ, the function grows at most as the inverse of the polynomial in λ.

Random Oracle. In the Random Oracle Model (ROM), all the parties have access to a truly random hash function. In particular, when a value v is given as an input from a party to the Random Oracle (RO), the latter samples a random answer r, stores the pair (v, r), and outputs r to the party. If the RO is queried on the same value v multiple times, the same answer r is output. It is heuristically shown that cryptographic hash functions such as SHA-256 behave as a RO.

Authenticated Encryption. A Symmetric Key Encryption scheme with key space $\mathcal{K}$ is composed of the following algorithms: *(i)* $\mathsf{Gen}(1^\lambda)$: The randomized key generation algorithm outputs a key k in the key space $\mathcal{K}$. *(ii)* $\mathsf{Enc}(k, m)$: The randomized encryption algorithm, on input the key k and a message m, outputs a ciphertext c. *(iii)* $\mathsf{Dec}(k, c)$: The decryption algorithm, on input the key k and a ciphertext c, outputs a message m, or $\perp$ if decryption fails. An authenticated encryption scheme should satisfy the following properties: *(i)* **Correctness:** For each message m and each $k \in \mathcal{K}$, $\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1$.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-38695-3_12

(ii) **Unforgeability:** The probability that a PPT adversary $\mathcal{A}$ wins the following experiment (i.e., outputs 1) is negligible: *(a)* Obtain k from $\mathsf{Gen}(1^\lambda)$. *(b)* The adversary, after making at most polynomially many queries to the encryption oracle $\mathsf{Enc}(k,\cdot)$, outputs a ciphertext c. *(c)* Let $m = \mathsf{Dec}(k,c)$. Output 1 if $m \neq \perp$ and c was not queried to the encryption oracle by $\mathcal{A}$. Output 0 otherwise. *(iii)* **CCA-Security:** The probability that a PPT adversary $\mathcal{A}$ wins the following experiment (i.e., outputs 1) is negligibly close to 1/2: *(a)* Obtain k from $\mathsf{Gen}(1^\lambda)$. *(b)* The adversary, with oracle access to $\mathsf{Enc}(k,\cdot)$ and $\mathsf{Dec}(k,\cdot)$, outputs messages m_0, m_1. *(c)* Choose a uniform bit $b \in \{0,1\}$, compute ciphertext $c = \mathsf{Enc}(k, m_b)$, and send c to $\mathcal{A}$. *(d)* The adversary, with oracle access to $\mathsf{Enc}(k,\cdot)$ and restricted oracle access to $\mathsf{Dec}(k,\cdot)$ (excluding c), outputs a bit b'. *(e)* The experiment outputs 1 if $b' = b$, else 0.

We can now state the formal properties we require from our protocol.

Correctness. For any function f and any set of messages M such that $f(M) = y$, if M is provided to the CCU for function f by the Data Provider obtaining ciphertexts C, and subsequently, ciphertexts C are provided by the Decider back to the CCU obtaining output y', then $y' = f(M) = y$.

Data Privacy. A PPT adversary $\mathcal{A}$ wins the following experiment with probability negligibly close to 1/2: *(i)* $\mathcal{A}$ chooses a function f. $\mathcal{A}$ has oracle access to the CCU acting as Data Provider (sending arbitrary sets of messages M and receiving corresponding ciphertexts C) and oracle access to the CCU acting as Decider (sending arbitrary sets of ciphertexts C and receiving back $f(M)$ if M correctly decrypts from C). Then, $\mathcal{A}$ chooses two sets of messages M_0, M_1 such that $f(M_0') = f(M_1')$ where M_b' can be any combination of messages in M_b. *(ii)* Choose a random bit b, produce ciphertexts C_b of messages M_b, and send C_b to $\mathcal{A}$. *(iii)* $\mathcal{A}$ has oracle access to the CCU as a Data Provider to send arbitrary sets of messages M, and obtain the corresponding ciphertexts C back. It has further oracle access to the CCU as a Decider by sending arbitrary ciphertexts C and obtaining back $f(M)$ if M correctly decrypts C. Finally, outputs a bit b'. *(iv)* If $b' = b$, output 1; otherwise, output 0.

Data Integrity. A PPT adversary $\mathcal{A}$ wins the following experiment with negligible probability: *(i)* $\mathcal{A}$, acting as Data Provider, has oracle access to the CCU submitting arbitrary message sets M and receiving ciphertexts C. $\mathcal{A}$ outputs ciphertexts C^*. *(ii)* Decrypting ciphertexts C^* yields messages M^*. Output 1 if no message in M^* equals $\perp$ and no subset of M^* was previously queried; else, output 0.

Forward/Backward Secrecy/Integrity. If all ciphertext keys at time t are leaked, ciphertexts encrypted at any different time t' remain secure, and their underlying messages cannot be decrypted or forged.

The protocol. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an authenticated encryption scheme with key space $\mathcal{K}$ and $H : \{0,1\}^* \to \mathcal{K}$ be a cryptographic hash function modeled as a RO. The protocol is as follows: **(1) Setup:** The CCU is initialized with functions $\{f_j\}$ and a uniform seed $s \in \{0,1\}^\lambda$. **(2) Data Storage:** *(i)* The

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

Data Provider establishes a secure channel with the CCU, chooses message set M, and sends it to the CCU. *(ii)* The CCU generates a unique ID T_i, key $k_i = H(s, T_i)$, ciphertexts $c_i = \mathsf{Enc}(k_i, m_i)$ for $m_i \in M$, and stores ciphertexts publicly. **(3) Decision:** *(i)* Decider establishes a secure channel with the CCU and sends ciphertext set C and function index j. *(ii)* CCU takes the unique ID T_i, keys $k_i = H(s, T_i)$, decrypts ciphertexts to messages m_i, and outputs decision $f_j(\{m_i\})$.

Theorem 1. *Assuming that* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is an Authenticated Encryption scheme, H is modeled as a Random Oracle and the CCU is a secure TEE, the protocol described above satisfies Correctness, Data Privacy, Data Integrity, and Forward/Backward Secrecy and Integrity.*

Proof. Recall that the CCU is modeled as a secure TEE. Hence, the fixed functions $\{f_j\}$ cannot be modified, internal states remain secret, and outputs are always correctly computed with respect to these functions. Correctness is ensured by the correctness property of the authenticated encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and by the fact that the CCU's functions are fixed and unmodifiable. Data Privacy is guaranteed since the internal CCU state (i.e., the seed s) remains secret and the derived encryption keys $k_i = H(s, T_i)$ appear indistinguishable from random due to the RO assumption. Thus, Data Privacy reduces directly to the CCA-security of $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Data Integrity follows from the Unforgeability property of the authenticated encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, combined with the trusted execution guarantees provided by the CCU. Forward/Backward Secrecy and Integrity results directly from the RO-based key derivation. Given a leaked key $k = H(s, T)$, the one-way property of the RO prevents recovery of the seed s. Therefore, keys corresponding to ciphertexts encrypted at other timestamps $T' \neq T$ remain computationally infeasible to derive.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-38695-3_12

This document is a pre-print copy of the manuscript
(Marangone et al. 2026)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-032-38695-3_12

References

Marangone, Edoardo, Eugenio Nerio Nemmi, Daniele Friolo, Giuseppe Ateniese, Ingo Weber, and Claudio Di Ciccio (2026). "A Secure, Confidential, and Verifiable Decision Support System". In: *ESORICS*. Ed. by Angelo Spognardi, Kun Sun, Fabio De Gaspari, and Luigi V. Mancini. Vol. 16969. Lecture Notes in Computer Science. Springer, pp. 233–253. ISBN: 978-3-032-38694-6. DOI: 10.1007/978-3-032-38695-3_12.

BibTeX

```
@InProceedings{   Marangone.etal/ESORICS2026:SecureConfidentialVerifiableDSS,
  author        = {Marangone, Edoardo and Nemmi, Eugenio Nerio and Friolo,
                   Daniele and Ateniese, Giuseppe and Weber, Ingo and Di
                   Ciccio, Claudio},
  booktitle     = {ESORICS},
  title         = {A Secure, Confidential, and Verifiable Decision Support
                   System},
  year          = {2026},
  pages         = {233--253},
  crossref      = {ESORICS2026},
  doi           = {10.1007/978-3-032-38695-3_12},
  keywords      = {TEE; Privacy; Security; Decision support}
}
@Proceedings{     ESORICS2026,
  title         = {Computer Security - {ESORICS} 2026 - 31st European
                   Symposium on Research in Computer Security, Rome, Italy,
                   September 14-18, 2026, Proceedings, Part {II}},
  year          = {2027},
  editor        = {Spognardi, Angelo and Sun, Kun and De Gaspari, Fabio and
                   Mancini, Luigi V.},
  isbn          = {978-3-032-38694-6},
  publisher     = {Springer},
  series        = {Lecture Notes in Computer Science},
  volume        = {16969}
}
```