MARTSIA: Enabling Data Confidentiality for Blockchain-based Process Execution

Edoardo Marangone[1], Claudio Di Ciccio[1], Daniele Friolo[1], Eugenio Nerio Nemmi[1], Daniele Venturi[1], and Ingo Weber[2]

[1] Sapienza University of Rome, Rome, Italy
{marangone,diciccio,friolo,nemmi,venturi}@di.uniroma1.it
[2] Technical University of Munich, School of CIT, and Fraunhofer Gesellschaft, Munich, Germany, first.last@tum.de

Abstract. Multi-party business processes rely on the collaboration of various players in a decentralized setting. Blockchain technology can facilitate the automation of these processes, even in cases where trust among participants is limited. Transactions are stored in a ledger, a replica of which is retained by every node of the blockchain network. The operations saved thereby are thus publicly accessible. While this enhances transparency, reliability, and persistence, it hinders the utilization of public blockchains for process automation as it violates typical confidentiality requirements in corporate settings. In this paper, we propose *MARTSIA*: A *Multi-Authority Approach to Transaction Systems for Interoperating Applications*. MARTSIA enables precise control over process data at the level of message parts. Based on Multi-Authority Attribute-Based Encryption (MA-ABE), MARTSIA realizes a number of desirable properties, including confidentiality, transparency, and auditability. We implemented our approach in proof-of-concept prototypes, with which we conduct a case study in the area of supply chain management. Also, we show the integration of MARTSIA with a state-of-the-art blockchain-based process execution engine to secure the data flow.

Keywords: Multi-Authority Attribute Based Encryption · Distributed Ledger Technology · InterPlanetary File System

1 Introduction

Enterprise applications of blockchain technology are gaining popularity because it enables the design and implementation of business processes involving many parties with little mutual trust, among other benefits [40,36]. Standard blockchains yield the capability of enabling cooperation between potentially untrusting actors through transparency: relevant data is made available to all participants of a blockchain network, and hence can be verified by anyone, thereby removing the need for trust [42]. In combination with the high-integrity permanence of data and non-repudiability of transactions offered by the technology, blockchains can be used to realize trustworthy protocols.

However, in multi-party business settings with scarce mutual trust, the involved parties typically have a strong need to keep certain data hidden from some of the business partners, and even more so from most other participants in a blockchain network. In fact, fulfilling security and privacy requirements is a key obstacle when it comes to the adoption and implementation of blockchain technology in general [43,14]. Corradini et al. [10] confirm the importance of security and privacy considerations for the specific case of process execution on blockchain.

Simple cryptographic solutions face severe downsides, as discussed in the following. First, the authors of [10] note that simply encrypting the contents of messages (payload), as previously proposed in the literature, does not guarantee the confidentiality of the information. Using synchronous encryption requires sharing a decryption key among process participants, and thus does not allow the sender of the data to selectively control access to different parts of a single message. Using asynchronous encryption and encrypting a message with the public key of the recipient requires the sender to create multiple copies of each message (one for each intended reader), which means that the sender can send *different* pieces of information to each participant – i.e., integrity is lost. Other proposed solutions address the issue via perimeter security: read access to the (relevant parts of) a blockchain is limited, e.g., by using channels on Hyperledger Fabric or similar [42, Ch. 2], or using private blockchains. However, this approach suffers from the same downsides as the use of synchronous encryption above. Also, permissioned platforms require the presence of trusted actors with the privileged role of managing information exchange *and* the right to be part of the network. In summary, most of the previous approaches offer "all-or-nothing" access: either all participants in some set can access the information in a message, or they receive only private messages and integrity of the data sent to multiple recipients is lost. In previous work [24], we introduced an early approach to control data access at a fine-granular level. However, the architecture relied on a central node for forging and managing access keys, thus leading to easily foreseeable security issues in case this single component were to be compromised or byzantine. Also, its integration with process management systems was yet to be verified. With the objective of overcoming these limitations, we have revised the entire approach from its foundations and devised the new solution we present here.

In this paper, we propose a Multi-Authority Approach to Transaction Systems for Interoperating Applications (MARTSIA). In MARTSIA, encrypted data is persisted in decentralized storage, which is connected to a public permissionless blockchain system supporting process execution. Data owners define access policies to regulate which users are able to view specific parts of the information. No central authority can generate decryption keys alone. The encryption and decryption of messages are left to the individual nodes. To attain the desired characteristics, this approach employs hybrid encryption in combination with the Ciphertext-Policy variant of Multi-Authority Attribute-Based Encryption (henceforth, MA-ABE for the sake of conciseness), smart contracts, and InterPlanetary File System (IPFS). In our evaluation, we show the integration of our implemented prototype with Caterpillar [22], a state-of-the-art process execution engine, to demonstrate how our approach can complement a business process management system to secure its data flow.

In the following, Sect. 2 presents a running example, to which we will refer throughout the paper, and illustrates the problem we tackle. Section 3 outlines the fundamental notions that our solution is based upon. In Sect. 4, we describe our approach in detail. In Sect. 5, we present our proof-of-concept implementation and the results of the experiments we conducted therewith. Section 6 reviews related work before Sect. 7 concludes the paper and outlines future works.

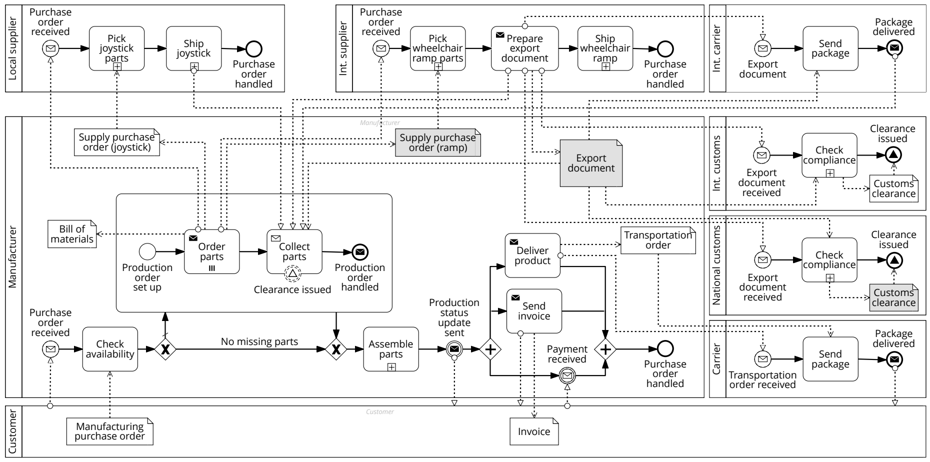


Fig. 1: A multi-party process for the assembly of special car parts.

2 Example, problem illustration, and requirements

Figure 1 shows a Business Process Model and Notation (BPMN) collaboration diagram illustrating a supply chain in the automotive area: the production of a special car for a person with paraplegia. The two base components for that car are a joystick (to turn, accelerate, brake) and a wheelchair ramp to let the person get into the vehicle.

A new process instance is initiated when a *customer* places an order for a modified car from a *manufacturer*. The *manufacturer* then checks the availability of joystick parts and wheelchair ramps in the warehouse, to order the missing ones from a local *joystick supplier* and an international *wheelchair ramp supplier*, respectively. Once all ordered parts have been collected, the suppliers prepare the packages with the products for delivery. The *international customs* verifies the *document* of the international supplier and issues custom clearance once the compliance verification is completed successfully. The *carrier* of the international supplier checks the documents and delivers the package to the *manufacturer* with an international shipment procedure. Upon receipt of the parts, the *manufacturer* proceeds with the assembly process. After informing the *customer* about the progress of the production process, the *manufacturer* sends an invoice and requests a carrier to deliver the package. The process is completed with the delivery of the ordered product.

Throughout the paper, we will refer to this scenario as a running example. In particular, we will focus on the information artifacts in Table 1 (marked with a gray background color in Fig. 1), namely: (1) the purchase order of the manufacturer, (2) the export document of the international supplier, and (3) the national customs clearance. The export document encloses multiple records, namely (2.a) the international shipment order, (2.b) the Corporate Sustainability Due Diligence Directive (CSDD), (2.c) the reference to the order, and (2.d) the invoice. These records are meant to be accessed by different players. The shipment order should only be accessible by the international carrier and the two customs bodies, the CSDD can only by read by the customs authorities, the order reference is for the manufacturer, and the invoice is for the manufacturer and the customs bodies. Differently from the purchase

Table 1: An excerpt of the information artifacts exchanged in Fig. 1

Message	Sender	Data	Recipients

Supply purchase order (ramp)	Manufacturer	`Company_name: Alpha`
		`Address: 38, Alpha street`
		`E-mail: cpny.alpha@mail.com`
		`Price: $5000`
			International supplier

Export document	International supplier	`Manufacturer_company: Beta`
		`Delivery_address: 82, Beta street`
		`E-mail: mnfctr.beta@mail.com`
		`Ramp run: 3`
		`Kickplate: 12`
		`Handrail: 7`
		`Baluster: 30`
		`Guardrail: 25`
		`Amount_paid: $5000`
			Manufacturer
			National customs
			International customs
			International carrier

		`Fundamental_workers_rights: Ok`
		`Human_rights: Ok`
		`Protection_biodiversity_and_ecosystems: Ok`
		`Protection_water_and_air: Ok`
		`Combatting_climate_change: Ok`
			National customs
			International customs

		`Manufacturer_company: Beta`
		`Address: 78, Beta street`
		`Order_reference: 26487`
			Manufacturer

		`Invoice_ID: 101711`
		`Billing_address: 34, Gamma street`
		`Gross_total: $5000`
		`Company_VAT: U12345678`
		`Issue_date: 2022-05-12`
			Manufacturer
			National customs
			International customs

Customs clearance	National customs	`Tax_payment: confirmed`
		`Conformity_check: passed`
		`Date: 2022-05-10`
		`Sender: Beta`
		`Receiver: Alpha`
			Manufacturer
			International customs

Table 2: Requirements and corresponding actions in the approach

	Requirement	**CAKE [24]**	**MARTSIA**	See
R1	Access to parts of messages should be controllable in a fine-grained way (attribute level), while integrity is ensured	✓	✓	Sections 4.2 and 5
R2	Information artifacts should be written in a permanent, tamper-proof and non-repudiable way	✓	✓	Sections 4.1 and 5
R3	The system should be independently auditable with low overhead	✓	✓	Sections 4.1 and 5
R4	The decryption key should only be known to the user who requested it	×	✓	Sections 4.1 and 5
R5	The decryption key should not be generated by a single trusted entity	×	✓	Sections 4.1 and 5
R6	The approach should integrate with control-flow management systems	×	✓	Sections 4.2 and 5

order and the customs clearance (messages 1 and 3), the four above entries (2.a to 2.d) are joined in a single document for security reasons: separate messages could be intercepted and altered, replaced or forged individually. Once they are all part of a single entity, every involved actor can validate all the pieces of information. Ideally, in a distributed fashion every node in the network could be summoned to attest to the integrity of that document. However, the need for separation of indicated recipients demands that only a selected group of readers be able to interpret the parts that are specifically meant for them (see the rightmost column of Table 1). In other words, though *visible* for validation, the data artifact should not be interpretable by everyone. The other actors should attest to data encrypted as in Table 3. This aspect gives rise to one of the requirements we discuss next.

Requirements. In recent years, there has been a surge in research on blockchain-based control-flow automation and decision support for processes (see [36] for an overview). Typically, information shared by actors in a collaborative process is commercial-in-confidence, i.e., shared only with the parties that need access to it, and who are in turn expected to not pass the information on. Our research complements this work by focusing on secure information exchange among multiple parties in a collaborative though partially untrusted scenario.

Table 2 lists the requirements stemming from the motivating use case that drives our approach and a research project in which two authors of this paper are involved.[3] The table highlights the limitations of our past work [24] that we overcome and indicates the sections in which we discuss the action taken to meet them. Different parties should be granted access to different sections of a confidential information source (**R1**, as in the case of the export document in our motivating scenario). The information source should remain available, immutable, and accountability should be granted for subsequent validations and verifications (**R2**, as for the check of the invoice by customs and, more in general, for process mining and auditing [17]), without major overheads (**R3**) for practical feasibility. In a distributed scenario such as that of the process in Sect. 4, where multiple authorities and actors are involved, it is necessary to secure the infrastructure by avoiding that any party can acquire (**R4**) or forge (**R5**) decryption keys alone. Finally, our approach should complement existing process execution engines to intercept and secure the data flow that characterized multi-party collaborations (**R6**). Next, we discuss the background knowledge that our approach is based on.

3 Background

Distributed Ledger Technologies (DLTs), and specifically programmable blockchain platforms, serve as the foundation for our work together with Multi-Authority Attribute-Based Encryption (MA-ABE). Here, we explain the basic principles underneath these building blocks.
Distributed Ledger Technologies. DLTs are protocols that allow for the storage, processing, and validation of transactions among a network of peers without the need for a central authority or intermediary. These transactions are timestamped and signed cryptographically, relying on asymmetric or public key cryptography with a pair of a private and a public key. In DLTs, every user has an account with a unique address, associated with such a key pair. The shared transaction list forms a ledger that is accessible to all participants in the network. A **blockchain** is a specific type of DLT in which transactions are strictly ordered, grouped into blocks, and linked together to form a chain. DLTs, including blockchains, are resistant to tampering due to the use of cryptographic techniques such as hashing (for the backward linkage of blocks to the previous one), and the distributed validation of transactions. These measures ensure the integrity and security of the ledger. Blockchain platforms come endowed with consensus algorithms that allow the distributed networks to reach eventual consistency on the content of the ledger [27]. Public blockchains like Ethereum [41] charge fees for the inclusion and processing of transactions. Ethereum supports expressive smart contracts, which are user-defined programs. They are deployed and invoked through transactions, i.e., their code is stored on chain and executed by many nodes in the network. Outcomes of contract invocations are part of the blockchain consensus, thus verified by the blockchain system and fully traceable. The execution of smart contract code, like transactions, incurs costs measured as *gas* in the Ethereum platform. Gas cost is based on the complexity of the computation and the amount of data exchanged and stored. To lower the costs of invoking smart contracts, external Peer-to-peer (P2P) systems are often utilized to store large amounts of data [42]. One of the enabling technologies is

[3] Cyber 4.0 project BRIE: `https://brie.moveax.it/en`. Accessed: 09 June 2023.

InterPlanetary File System (IPFS),[4] a distributed system for storing and accessing files that utilizes a Distributed Hash Table (DHT) to scatter the stored files across multiple nodes. Like DLTs, there is no central authority or trusted organization that retains control of all data. IPFS uses content-addressing to uniquely identify each file on the network. Data stored on IPFS is linked to a resource locator through a hash, which–in a typical blockchain integration–is then sent to a smart contract to be stored permanently on the blockchain [21]. In a multi-party collaboration setting like the one presented in Sect. 2, the blockchain provides an auditable notarization infrastructure that certifies transactions among the participants (e.g., purchase orders or customs clearances). Smart contracts ensure that the workflow is carried out as agreed upon, as described in [12,25,40]. Documents like purchase orders, transportation orders, and customs clearances can be stored on IPFS and linked to transactions that report on their submission. However, data is accessible to all peers on the blockchain. To take advantage of the security and traceability of the blockchain while also controlling access to the stored information, it is necessary to encrypt the data and manage read and write permissions for specific users.
Attribute-Based Encryption (ABE). ABE is a form of public key encryption in which the *ciphertext* (i.e., an encrypted version of a *plaintext* message) and the corresponding decryption key are connected through attributes [35,6]. In particular, Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [4,20] associates each potential user with a set of attributes. Policies are expressed over these attributes using propositional literals that are evaluated based on whether a user possesses a particular property. In the following, we shall use the teletype font to format attributes and policies. For example, user `0xB0...1AA1` is associated with the attributes `Supplier`, to denote their role, and `43175279`, to specify the process instance number they are involved in (the *case id*). For the sake of brevity, we omit from the attribute name that the former is a role and the latter a process instance identifier (e.g., `Supplier` in place of `RoleIsSupplier` or `43175279` instead of `InvolvedInCase43175279`) as we assume it is understandable from the context. Policies are associated with ciphertexts and expressed as propositional formulae on the attributes (the literals) to determine whether a user is granted access (e.g., `Carrier or Manufacturer`).
As argued in the introduction, one goal of this work is to move away from a single source of trust (or failure); thus, we consider multi-authority methods. To decrypt and access the information in a ciphertext, a user requires a dedicated key. With Multi-Authority Attribute-Based Encryption (MA-ABE), every authority creates a part of that key, henceforth *decryption key (dk)*. A *dk* is a string generated via MA-ABE on the basis of *(i)* the user attributes, and *(ii)* a secret key of the authority. To generate the secret key (coupled with a public key), the authority requires public parameters composed of a sequence of pairing elements that are derived from a pairing group via Elliptic Curve Cryptography (ECC). Due to space restrictions, we cannot delve deeper into the notions of pairing groups and pairing elements. We refer to [6,26] for further details. Once the user has obtained a *dk* from every required authority, it merges them obtaining the *final decryption key (fdk)* to decrypt the message.
In the Cypertext-Policy variant of MA-ABE, a ciphertext for a given message is generated from the public parameters, the public keys of all the authorities, and a policy. In our context, users are process participants, messages are the data artefacts exchanged during process execution, ciphertexts are encrypted versions of these artefacts, policies determine which artefacts can be accessed by which users, and keys are the tools granted to process parties

[4] ipfs.tech. Accessed: 09 June 2023.

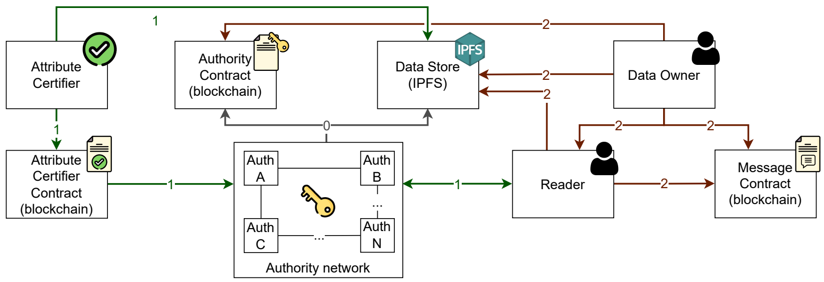


Fig. 2: The key components and their interactions in the MARTSIA approach

to try to access the artefacts. In the following sections, we describe how we combine the use of blockchain and the Cypertext-Policy variant of MA-ABE to create an access control architecture for data exchanges on the blockchain that meets the requirements listed in Table 2.

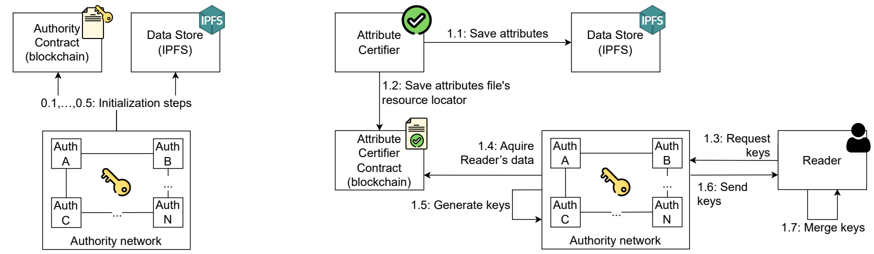
4 The MARTSIA approach

In this section, we describe our approach, named Multi-Authority Approach to Transaction System for Interoperating Applications (MARTSIA). We begin by examining the collaboration among its core software components, and then illustrate the data structures they handle.

4.1 Workflow

Figure 2 illustrates the main components of our architecture and their interactions. The involved parties are: **the Attribute Certifier** specifying the attributes characterizing the potential readers of the information artifacts; we assume the Attribute Certifiers to hold a blockchain account; different Attribute Certifiers may attest to different pieces of information about potential readers; **the Data Owner** encrypting the information artifacts (henceforth also collectively referred to as *plaintext*) with a specific access policy (e.g., the manufacturer who wants to restrict access to the purchase orders to the sole intended parties, i.e., the suppliers); we assume the Data Owner to hold a blockchain account and a Rivest–Shamir–Adleman (RSA) [33] secret/public-key pair; **Readers** interested in some of the information artifacts (e.g., the manufacturer, the joystick supplier, and· the wheelchair ramp supplier); we assume the Readers to hold a blockchain account, a RSA secret-key/public-key pair, and a global identifier (GID) that uniquely identifies them; **the Authorities** that calculate their part of the secret key for the Reader; **the Data Store,** a P2P repository based on IPFS. IPFS saves all exchanged pieces of information in a permanent, tamper-proof manner creating a unique content-based hash as resource locator for each of them; **the Smart Contracts** used to safely store and make available the resource locators to the ciphertext saved on the Data Store (**Message Contract**), the information about potential readers (**Attribute Certifier Contract**), and the data needed by the authorities to generate the public parameters (**Authority Contract**).

We divide our approach in three main phases, which we discuss in detail next: initialization (Fig. 3(a)), key management (Fig. 3(b)), and data exchange (Fig. 4). In the following, the numbering scheme corresponds to the labels in Figs. 2 to 4.



(a) Authority initialization (b) Key management

Fig. 3: Authority initialization and key management phases in MARTSIA

0: Initialization. Here we focus on the network of authorities, as depicted in Fig. 3(a). The initialization phase consists of the following five steps. (**0.1**) First, each authority creates a separate file with the metadata of all the authorities involved in the process.[5] Authorities are responsible for the setting of public parameters that are crucial to all the algorithms of MA-ABE. Therefore, we have redesigned the public parameter generation program as a Multi-party Computation (MPC) protocol [8,9] to guarantee full decentralization. More specifically, we adapt a commit-then-open coin-tossing protocol [5] as follows to generate a random pairing element, that is, the core piece of data described in [34] for MA-ABE implementation. (**0.2**) Each authority posts on the blockchain the hash of a locally generated random pairing element by invoking the Authority Contract. (**0.3**) After all the hashes are publicly stored, each authority posts the *opening*, namely the previously hashed pairing element in-clear, completing the commit-then-open coin-tossing protocol introduced before. (**0.4**) Then, every authority *(i)* verifies that all the hashes of the pairing elements match the respective openings, *(ii)* independently combines all posted openings via bitwise XOR, and *(iii)* uses the output of this operation (the *final shared pairing element*) to calculate the set of public parameters as illustrated in [34]. (**0.5**) Each authority generates its own public-key/secret-key pair by using the authority key generation algorithm of MA-ABE. To enable full decentralization and notarization, we resort to the Data Store to save the output of all actions (**0.1** to **0.5**) and the Authority Contract to keep track of the corresponding resource locators.

1: Key Management. The key management phase is comprised of the following steps, as illustrated in Fig. 3(b): (**1.1**) The Attribute Certifiers save the attributes and the identifying blockchain account addresses of the Readers on the Data Store and (**1.2**) the corresponding resource locator on the Attribute Certifier Contract so as to make them publicly verifiable on chain. To this end, every Attribute Certifier operates as a push-inbound oracle [2], storing on chain the attributes that determine the role of the Reader and, optionally, the list of process instances in which they are involved. For example, an Attribute Certifier stores on chain that `0x82...1332` is the address of a user that holds the `Manufacturer` role and participates in

[5] Notice that metadata are known to all the authorities and all the actors involved in the process. Therefore, non-malicious authorities are expected to create an identical file. The (same) hash is thus at the basis of the resource locator. As a consequence, anyone can verify whether the authorities behave properly in this step by checking that the resource locators are equal, with no need to load the file from the Data Store.

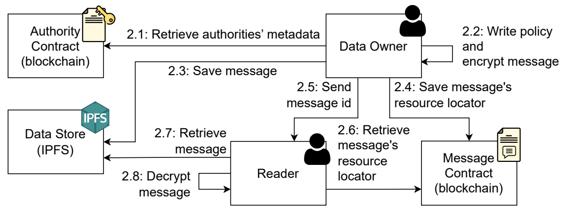


Fig. 4: The data exchange phase in MARTSIA

the process identified by `43175279`. Another Attribute Certifier registers that `0xB0...1AA1` and `0x9E...C885` are Readers both endowed with the `Supplier` and `43175279` attributes, though the former is `National` and the latter `International`. Readers' attributes are stored on a public blockchain for verifiability. However, notice that Readers are referred to by their public addresses, thus keeping pseudonimity. Also, the attribute names are strings that we keep intuitively understandable for the sake of readability in this paper, yet only serve as propositional symbols for the encoding of policies. Obfuscation techniques can thus be seamlessly applied though their discussion goes beyond the scope of this paper. Whenever a Reader (e.g., the international customs) wants to access the data of a message (e.g., the sections of interest in the Document), they operate as follows: (**1.3**) They request a key to all the authorities, passing the identifying GID as input (we enter the detail of this passage later); (**1.4**) Each authority seeks the Reader data (the blockchain address and attributes), and obtains them from the Attribute Certifier Contract; (**1.5**) Equipped with these pieces of information alongside the public parameters, the secret key, and the user's GID, each authority produces a MA-ABE decryption key (*dk*) for the Reader, and (**1.6**) sends it back. Once all *dk*s are gathered from the authorities, (**1.7**) the Reader can merge them to assemble their own *fdk*. Notice that none of the Authorities can create the *fdk* alone (unless specified as such), thus meeting **R5**; no user other than the intended Reader can obtain the key (**R4**). The key management phase can be interleaved with the Data Exchange phase (below).

A note on the security of key requests. Maliciously obtaining the *fdk* of another Reader is a high security threat. We decompose this issue in two challenges. First, we want to avoid that information exchanged between a Reader and an Authority is intercepted by other parties. To this end, we convey every communication in step **1.3** through a separate client-server Secure Sockets Layer (SSL) connection between the Reader and each of the authorities. To avoid any false self-identification as a Reader through their GID, we include a handshake preliminary phase in the protocol. It starts with every Authority (server) sending a random value (or *challenge*) to the Reader (client). The latter responds with that value signed with their own RSA private key, so as to let the invoked components verify their identity with the caller's public key.

2: Data Exchange. Figure 4 presents the operations carried out for information storage and access. As a preliminary operation, the Data Owner verifies that the hash links of the files with metadata and public parameters posted by all the authorities are equal to one another to ascertain their authenticity.[5] Then, data is transferred from the Data Owner to the Readers through the following steps. (**2.1**) The Data Owner retrieves the authorities' public keys and public parameters from the blockchain. (**2.2**) Then, they write a policy. Notice that standard MA-ABE sets a maximum length for the input files. In a business process

Table 3: Example of messages stored by MARTSIA upon encryption

Message	Metadata	Header	Body (slices)
Supply purchase order (ramp)	sender: 0x82[...]1332 process_instance_id: 43175279 message_id: 22063028	EncryptedKey: eJytm1[...]eaXV2u Fields: {CV8=[...]6w==, p84W[...]aw==, avNL[...]1w==, AOo=[...]mQ==}	CV8=[...]6w==: Ruk/[...]QQ== p84W[...]aw==: 1UTk[...]mA== avNL[...]1w==: VT+D[...]JQ== AOo=[...]mQ==: ZUE/[...]6w==
Export document	sender: 0x9E[...]C885 process_instance_id: 43175279 message_id: 15469010	SliceId: 62618638 EncryptedKey: eJytWU[...]Bmn4k= Fields: {u7o=[...]iw==, TacL[...]IQ==, EhrB[...]lw==, Tiba[...]Wg==, xJ+6[...]SQ==, RPn7[...]Zz==, T3Wq[...]eK==, QXjN[...]LS==, Jmk8[...]fL==}	u7o=[...]iw==: py8l[...]mA== TacL[...]IQ==: MKNH[...]uQ== EhrB[...]lw==: 5bz0[...]bg== Tiba[...]Wg==: JahD[...]5Q== xJ+6[...]SQ==: 3aIm[...]kg== RPn7[...]Zz==: nRNh[...]GA== T3Wq[...]eK==: PxRk[...]0g== QXjN[...]LS==: Lfhk[...]Ug== Jmk8[...]fL==: Gj43[...]yg==
		SliceId: 19756540 EncryptedKey: eJytm0[...]wtYFo= Fields: {ZOSH[...]9B==, BZ8l[...]KY==, QeUy[...]WT==, rtdZ[...]hf==, 8kuG[...]Ts==}	ZOSH[...]9B==: sctC[...]nQ== BZ8l[...]KY==: 5Gln[...]Pg== QeUy[...]WT==: k49+[...]kA== rtdZ[...]hf==: lRLd[...]tm== 8kuG[...]Ts==: 25E0[...]uc==
		SliceId: 12191034 EncryptedKey: eJytWU[...]sGlU4= Fields: {t8gr[...]QQ==, yuwd[...]vg==, 1K1d[...]zQ==}	t8gr[...]QQ==: ZJ1v[...]5A== yuwd[...]vg==: 7XpN[...]4A== 1K1d[...]zQ==: +QbM[...]Cw==
		SliceId: 98546521 EncryptedKey: eJytk9[...]oIJS6= Fields: {rjY=[...]KQ==, ZdWC[...]xg==, 6aLB[...]iw==, VD2h[...]6w==, 8UmX[...]MQ==}	rjY=[...]KQ==: JPAv[...]LA== ZdWC[...]xg==: w05J[...]Hg== 6aLB[...]iw==: 0wWu[...]vA== VD2h[...]6w==: eZu7[...]QQ== 8UmX[...]MQ==: sXaB[...]kQ==
National customs clearance	sender: 0x5F[...]FFE1 process_instance_id: 43175279 message_id: 64083548	EncryptedKey: eJytWU[...]gGl2Y= Fields: {fdoT[...]kA==, 2AkH[...]Rw==, 0bTn[...]6A==, RZVJ[...]rQ==, 4TXI[...]zw==}	fdoT[...]kA==: fUSZ[...]Bg== 2AkH[...]Rw==: dGp2[...]zA== 0bTn[...]6A==: TuR9[...]bA== RZVJ[...]rQ==: Pq8U[...]dQ== 4TXI[...]zw==: 0Izx[...]Mw==

context, this limitation would undermine practical adoption. To cater for the encryption of arbitrary-size plaintexts, we thus resort to a two-staged hybrid encryption strategy [11]. First, the Data Owner encrypts via MA-ABE a randomly generated symmetric key (of limited size, e.g., `b'3:go+s...x2g='`) with the authorities' public keys and the policy (obtaining, e.g., `eJytm1...eaXV2u`). Afterwards, it encrypts the actual information artifact (of any size) via symmetric key encryption scheme [28] using that symmetric key. In our example scenario, then, the manufacturer does not encrypt via MA-ABE the supply purchase order, but the key through which that document is encrypted (and decryptable). (**2.3**) Thereupon, the Data Owner (e.g., the manufacturer) saves the encrypted symmetric key and information artifact (plus additional metadata we omit here for the sake of clarity and detail in Sect. 4.2) in one file on the Data Store, (**2.4**) sends the file's resource locator to the Message Contract, and (**2.5**) transmits the unique message ID (e.g., `22063028`) assigned to the file to the Reader (e.g., the supplier). As the information artifact is on the Data Store and its resource locator saved on chain, it is written in a permanent, tamper-proof and non-repudiable way, thus meeting requirement **R2**. Equipped with their own *fdk*, the Reader can begin the message decryption procedure. (**2.6**) At first, the Reader retrieves the resource locator of the message from the Message Contract. (**2.7**) Then, once the Reader obtains the ciphertext from the Data Store, they pass it as input alongside the public parameters (see step **0.4** above) and the *fdk* to the MA-ABE decryption algorithm running locally. (**2.8**) Mirroring the operations explained in step **2.1**, MA-ABE decrypts the symmetric key from the retrieved ciphertext. Only with the symmetric key, the Reader can obtain the original information artifact.

4.2 Data Structures

After the analysis of the software components and tasks employed in our approach, we focus on its core data structures: messages and policies.

Messages. Table 1 illustrates the messages we described in our running example in Sect. 2 along with a generated symmetric key for each message. Table 3 shows the messages as saved

Table 4: Message policy examples

Message	Slice	Policy
Supply purchase order (ramp)		`43175279@2+ and (Manufacturer@1+ or (Supplier@1+ and International@1+))`
Export document	a	`Customs@A or (43175279@2+ and ((Supplier@1+ and International@1+) or Manufacturer@1+ or (Carrier@1+ and International@1+)))`
	b	`Customs@A or (43175279@2+ and (Supplier@1+ and International@1+))`
	c	`43175279@2+ and ((Supplier@2+ and International@1+) or Manufacturer@1+)`
	d	`Customs@A or (43175279@2+ and ((Supplier@1+ and International@1+) or Manufacturer@1+))`
National customs clearance		`Customs@A or (43175279@2+ and Manufacturer@1+)`

on the Data Store by the Data Owner after the encryption process explained in Sect. 4.1 (phase 2). Each file stored on the Data Store consists of one or more sections to be accessed by different actors (henceforth, *slices*). Every slice is divided in three parts. **The metadata** contain the *message sender* (e.g., `0x82...1332` in Table 3), the *case id* (e.g., `43175279`), and the *message id* that uniquely identifies the message (e.g., `22063028`). **The body** is the encrypted information saved as key/value entries (*fields*) for ease of indexation. For security, notice that neither the keys nor the values are in clear. **The header** consists of the *encrypted* symmetric *key* generated at step **2.1**, and the list of field keys that the body contains. In case two or more slices form the message (as in the case of the export document), each is marked with a unique *slice id* (e.g., `62618638`). We recall that a message is stored on the Data Store and retrievable through a hash, content-based resource locator. The resource locator can thus be attached to process execution data for monitoring and auditability purposes, in compliance with **R6**.

Policies. We use MA-ABE policies to specify read grants to message slices, thus enabling fine-grained access control as per **R1**. For example, the export document written by the international supplier of process instance `43175279` is partitioned in four slices as illustrated in Table 1. Table 4 shows the encoding of the policies that restrict access to specific classes of Readers, based on the attributes the Attribute Certifiers attested to in step **1.1**.

Henceforth, we will use the following notation to encode a policy P. We shall use $\mathrm{Attr}\texttt{@}X$ as a shorthand notation for a policy indicating that an authority Auth (if X is Auth) or *at least* $n \geqslant 1$ authorities (if X is n+) generate the key based on the verification of attribute Attr. Compound policies can be formed by joining $\mathrm{Attr}\texttt{@}X$ propositions with `or` and `and` logical operators. For instance, `(Customs@A or Supplier@1+)` declares that only authority `A` can authorize customs, whereas *any* authority can generate the *dk* for suppliers, and that only customs or suppliers can read a message. To sum up, we will henceforth use the following grammar for policies P:

$$P ::= \mathrm{Attr}\texttt{@}X \quad | \quad P \ \texttt{and} \ P' \quad | \quad P \ \texttt{or} \ P' \qquad \textit{where } \mathrm{Attr} \textit{ is an attribute;}$$
$$X ::= \mathrm{Auth} \quad | \quad n\texttt{+} \qquad \textit{where } \mathrm{Auth} \textit{ is an authority and } n \textit{ a positive integer.}$$

Notice that we enable the selection of a specific *dk* forger for backward compatibility towards single-authority frameworks. The downsides are that *(i)* no key is generated if that authority is down (if `A` crashed, e.g., a user cannot be recognized as a customs body), and *(ii)* a corrupted authority could take over the generation of an *fdk* if only one attestation is necessary (theirs). Therefore, special attention must be paid in the writing of policies. `43175279@2+` requires that *at least* two authorities attest to the participation of a user in case `43175279`. A user that is not authorized by all the required authorities cannot have the final decryption key as per the policy. Also, whenever multiple authorities are involved in the generation of the *fdk* by contributing to a part of it (the *dk*), only the user can compose the *fdk* and decrypt the ciphertext.

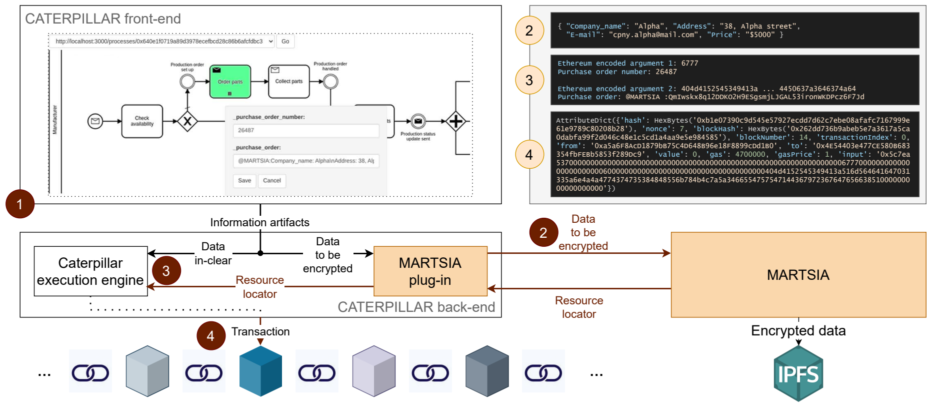


Fig. 5: A run of our integration of MARTSIA with Caterpillar [22]

In our example, the international shipment order is the first slice of the export document. It should be readable by the national and international customs, and by specific actors involved in the process instance: the sender (i.e., the international supplier), the manufacturer, and the international carrier. Additionally, we exert constraints on the authorities providing the dk: Customs are given the dk by Authority `A`, and at least two Authorities must declare that a Reader is involved in the given process instance. The other attributes can be attested to by any Authority. This composite rule translates to the following expression: `Customs@3+ or (43175279@2+ and ((Supplier@1+ and International@1+) or Manufacturer@1+ or (Carrier@1+ and International@1+)))`.

Thus far we have described the architecture of MARTSIA, along with its operations, employed techniques and data structures. Next, we focus on its realization and testing.

5 Implementation and evaluation

MARTSIA is an approach aimed at securing the access to information at a fine-grained level in a distributed fashion. We have hitherto shown its security guarantees by design, using a multi-party process execution as a motivating scenario. In this section, we experimentally evaluate whether MARTSIA can deliver its guarantees and properties in a process context, and at what cost. The code of our prototype alongside the detailed results of our experiments can be found at github.com/apwbs/MARTSIA-Ethereum. We implemented the three contracts described in Sect. 4.1 as a single instance in Solidity, a programming language for the Ethereum Virtual Machine (EVM). We deployed the instance on the Sepolia (Ethereum), Mumbai (Polygon), and Fuji (Avalanche) testnets.[6] We created an IPFS local node[4] to realize the Data Store, and used Python to encode the off-chain modules including the client-server communication channels.

First, to demonstrate the adoption of MARTSIA as a secure data-flow management layer for process execution (thus meeting requirement **R6**), we created an integration module

[6]Goerli: sepolia.etherscan.io; Mumbai: mumbai.polygonscan.com; Avalanche: testnet.snowtrace.io. Accessed: 09 June 2023.

connecting our tool with a state-of-the-art blockchain-based process execution engine tool, i.e., Caterpillar [22]. For our experiments, we used Caterpillar v.1.0.[7] The code to replicate and extend our test is publicly available in our repository.[8] As shown in Fig. 5, we insert a plug-in in the architecture of Caterpillar to use MARTSIA as an intermediate data layer manager, replacing the built-in data store for information securing. We illustrate our experiment with a simplified fragment of the running example (see Sect. 2) focusing on the purchase order sent from the manufacturer to the international supplier. The exchanged data artifact consists of a purchase order number, to be publicly stored on chain, and the confidential purchase order entries listed in the top row of Table 1. The user passes both the entries as input through the Caterpillar panel (see mark ① in Fig. 5), specifying the data they want to be secured by MARTSIA with a special prefix (“`@MARTSIA:`”). Our integration module captures the input and encrypts the indicated entry as explained in Sect. 4 so that only the supplier can read and interpret those pieces of information as per **R1** (②). Once the encryption is concluded, MARTSIA invokes the Caterpillar Smart Contract passing the first argument (the purchase order number) as is, and replaces the second argument with a marked IPFS link in place of the original data (③). The resource locator for the stored information is thus saved on the ledger by the process execution engine for future audits (④), yet not publicly readable (**R2**). Thereupon, the recipient of the confidential information (or the auditor, later on) can retrieve and decode the information with their secret key (**R4**) provided by the authority network (**R5**).

Aside from empirically showing the suitability of MARTSIA as a secure data-flow manager in an ensemble with a process execution engine, we remark that the cost overhead in terms of transaction fees required by MARTSIA is negligible with respect to the main process execution management. For example, the method running the activity `Order parts` of the BPMN in Fig. 1 on Caterpillar incurs 114494 gas units for execution with our inputs. The on-chain component of MARTSIA, detached from Caterpillar, requires 89772 gas units to store the IPFS link. As we use the second string input field of the Caterpillar's smart contract to save that resource locator, the separate gas consumption for MARTSIA is unnecessary and can be directly included in the overall process execution costs. Notice that the same activity execution saving the purchase order as plaintext (thus renouncing to the fine-grained confidentiality guarantees of MARTSIA) would have entailed a larger cost because the textual content is a longer string than an IPFS link: 116798 gas units. Auditability on process execution *and* secure message exchanges are thus guaranteed with low overhead, as stated in **R3**.

To gauge the gas expenditure and execution time of our system's on-chain components, we called the methods of the deployed Smart Contract daily on Sepolia, Fuji and Mumbai for 14 days (from 16 to 29 May 2023). All the experiment's transactions and gas measurements are available in our code repository.[9] The data we used to run the tests are taken from our running example (see Sect. 2). Table 5 illustrates the results. We divide the measurement in four different phases: *(i)* the deployment of the Smart Contract; *(ii)* the initialization of the Authorities (steps **0.1** to **0.5**); *(iii)* the Reader certification (step **1.2**); *(iv)* the storage of a message by a Data Owner to save a message (step **2.4**). For all the above phases, the table shows the average gas consumed for execution (ranging from 67533 units for step **1.2** to 1692955 units for the Smart Contract deployment) and the cost converted in Gwei (i.e., 10^{-9}

[7] github.com/orlenyslp/Caterpillar. Accessed: 10 June 2023.
[8] github.com/apwbs/MARTSIA-Ethereum/tree/main/caterpillar-interaction
[9] github.com/apwbs/MARTSIA-Ethereum/tree/main/tests. Accessed: 10 June 2023.

Table 5: Execution cost and timing of the steps that require an interaction with the blockchain

	Execution cost [Gwei = ETH × 10^{-9}]				
Platform	Contract deployment (1692955 gas units)	Steps **0.1** to **0.5** (476547 gas units)	Step **1.2** (67533 gas units)	Step **2.4** (89772 gas units)	**Avg. latency** [ms]
Sepolia (ETH)	2539432.514	714820.504	101299.501	134658.001	9288.574
Fuji (AVAX)	340498.771	95873.485	13586.538	18060.662	4278.099
Mumbai (MATIC)	1283.163	354.691	50.311	66.012	4944.807
	Off-chain execution time [ms]				
	0.000	2582.471	38.280	158.447	

Ether). Along with the analysis of costs, we measured the time needed to perform the steps of our approach. Each step involves sending a transaction to the blockchain. Therefore, we separate the execution time between the off-chain data elaboration and the latency induced by the blockchain infrastructure. The average time required to store a transaction in a block ranges from approximately 4.3 sec (Fuji) to 9.3 sec (Sepolia). The off-chain passages require a lower time, from about 0.038 sec for the Reader certification (Step **1.2**) to the circa 2.6 sec needed for the cooperative work carried out by the Authorities during the initilization phase (steps **0.1** to **0.5**). More in-depth comparative analyses and a stress test of the architecture pave the path for future endeavors, as we discuss in Sect. 7 after a summary of the state of the art.

6 Related work

In recent years, numerous approaches have been proposed to automate collaborative processes using blockchain technology [12] beyond the aforementioned Caterpillar [22]. Previous studies in the area have shown the effectiveness of blockchain-based solutions to add a layer of trust among actors in multi-party collaborations [40] even in adversarial settings [23], improve verifiability of workflows with model-driven approaches [22,37], allow for monitoring [13], mining [17], and auditing [10]. Interestingly, a more recent release of Caterpillar [21] enables the dynamic allocation of actors based on a language for policy bindings. MARTSIA has the capability to adjust roles dynamically as well, as access keys are created based on actors' attributes verified at runtime. These studies enhance the integration of blockchain technology with process management, unlocking security and traceability benefits. However, they primarily focus on the control-flow perspective and lack mechanisms for secure access control to data stored on public platforms. In contrast, our work focuses specifically on this aspect in the context of collaborative business processes and, as we demonstrated in Sect. 5, can complement existing blockchain-based process execution engines.

Another area of research related to our investigation is the protection of privacy and integrity of data stored on the blockchain. Several papers in the literature explore the use of encryption for this purpose. Next, we provide an overview of techniques. Hawk [18] is a decentralized system that utilizes user-defined private Smart Contracts to automatically implement cryptographic methods. Our approach does not require the encoding of custom smart contracts, as it is based on policies stored on chain to encrypt messages. Bin Li et al. [19] introduce RZKPB, a privacy protection system for shared economy based on blockchain technology. Similarly to MARTSIA, this approach does not involve third parties and resorts to external data stores. Differently from their approach, we link data on chain with the data stores so as to permanently store the resource locators. Henry et al. [16] employ smart contracts that handle payment tokens. Banks operate as trustworthy intermediaries to preserve privacy. MARTSIA pursues confidentiality of exchanged information too, although it does not resort to central authorities (the banks) to this end. Rahulamathavan et al. [32] propose a new blockchain architecture for IoT applications that preserves privacy through the use of ABE. We also utilize ABE in our approach, but MARTSIA integrates with existing technologies, whilst their model aims to change the blockchain protocol. Benhamouda et al. [3] introduce a solution that enables a public blockchain to serve as a storage place for confidential data. As in our approach, they utilize shared secrets among components. However, their approach discloses the secret when determined conditions are fulfilled, whereas MARTSIA does not reveal secret data on the blockchain. In the healthcare domain, Wang et al. [39] create a secure electronic health records system that combines Attribute-Based Encryption, Identity-Based Encryption, and Identity-Based Signature with blockchain technology. Their architecture is different from ours as the hospital has control over the patient's data and sets the policies, whereas solely the data owners manage data in MARTSIA. Tran et al. [38] and Pournaghi et al. [31] propose approaches for decentralized storage and sharing based on private blockchains. We operate in the context of public blockchains to leverage the higher degree of security given by the general validation of transactions. Athanere et al. [1] present an approach where the data owner encrypts the file, and then a hashed version of it is stored on a cloud server. The data owner encrypts the data with the public key of the message reader, and the necessary public parameters are generated by an administrator. MARTSIA differs from this solution because it uses MA-ABE and symmetric key encryption to encrypt the data instead of a public key. Pham et al. [30] propose an idea for a decentralized storage system named B-Box, based on IPFS, MA-ABE and blockchains. Though we resort to those building blocks too, we include mechanisms for secure initialization of the authority network, allow for fine-grained access control on message parts, and impede by design any actor from accessing keys.

7 Conclusion and future remarks

In this work, we introduce MARTSIA, a technique that merges blockchain technology with Multi-Authority Attribute-Based Encryption (MA-ABE) to regulate data access in the scenario of multi-party business operations. Additionally, our method employs IPFS for preserving information artifacts, access regulations, and metadata. We utilize smart contracts to keep the user attributes, establish the access grants to the process participants, and save the connection to IPFS files. MARTSIA allows for a detailed specification of access permissions, ensuring data reliability, persistence, and irrefutability, thus enabling auditability with minimal added costs.

Our approach exhibits limitations we aim to overcome in future work. If a Data Owner wants to revoke access to data for a particular Reader, e.g., they can change the policy and encrypt the messages again. However, the old data on IPFS would still be accessible. Therefore, we are considering the usage of InterPlanetary Name System (IPNS), as it allows for the replacement of existing files. With it, a message can be replaced with a new encryption thereof that impedes Readers whose grant was revoked to access it. More generally, the life-cycle of data artifacts, policies and smart contracts constitutes a management aspect worth investigating. From a technological perspective, we are working on the implementation of MARTSIA on other public blockchain platforms such as Algorand[10] [7] to analyze the benefits and challenges stemming from different DLTs, including costs. Also, we are developing an alternative key request protocol for readers that adopts the blockchain as a communication layer so as to avoid direct channels between readers and authorities. In light of the considerable impact that a correct expression of policies has on the overall approach, we envision automated verification and simulation of policies for future work to properly assist the users in their policy specification task. Future endeavors also include the integration of Zero Knowledge Proofs [15] with ABE to yield better confidentiality and privacy guarantees, and of decentralized identifiers [29] and oracles [2] to verify data ownership. We plan to conduct a formal threat analysis to prove the security of our approach, and run field tests for the empirical evaluation of its robustness.

Acknowledgements. The work of E. Marangone and C. Di Ciccio was partially funded by the Cyber 4.0 project BRIE, the Sapienza project DRONES, by project SERICS (PE00000014) under the NRRP MUR program funded by the EU-NGEU, and by project PINPOINT (B87G22000450001) under the PRIN MUR program.

References

1. Athanere, S., Thakur, R.: Blockchain based hierarchical semi-decentralized approach using IPFS for secure and efficient data sharing. J. King Saud Univ. - Comput. Inf. **34**(4), 1523–1534 (2022)
2. Basile, D., Goretti, V., Di Ciccio, C., Kirrane, S.: Enhancing blockchain-based processes with decentralized oracles. In: BPM Blockchain and RPA Forum. pp. 102–118 (2021)
3. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: TCC (2020)
4. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: SP. pp. 321–334 (2007)
5. Blum, M.: Coin flipping by telephone. In: CRYPTO. pp. 11–15 (1981)
6. Chase, M.: Multi-authority attribute based encryption. In: TCC. pp. 515–534 (2007)
7. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theor. Comput. Sci. **777**, 155–183 (2019)
8. Chi, A., Yao, C.: Protocols for secure computations (ext. abstract). In: FOCS. pp. 160–164 (1982)
9. Chi, A., Yao, C.: How to generate and exchange secrets (ext. abstract). In: FOCS. pp. 162–167 (1986)
10. Corradini, F., Marcelletti, A., Morichetta, A., et al.: Engineering trustable and auditable choreography-based systems using blockchain. ACM Trans. Manage. Inf. Syst. **13**(3) (2022)
11. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. **33**(1), 167–226 (2003)
12. Di Ciccio, C., Cecconi, A., Dumas, M., García-Bañuelos, L., et al.: Blockchain support for collaborative business processes. Inform. Spektrum **42**, 182–190 (2019)
13. Di Ciccio, C., Meroni, G., Plebani, P.: On the adoption of blockchain for business process monitoring. SoSyM **21**(3), 915–937 (2022)
14. Feng, Q., He, D., Zeadally, S., Khan, M.K., Kumar, N.: A survey on privacy protection in blockchain system. Journal of Network and Computer Applications **126**, 45–58 (2019)
15. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Providing Sound Foundations for Cryptography, pp. 203–225 (2019)
16. Henry, T., Hatin, J., Kazmierczak, L., Laga, N., Gaaloul, W., Bertin, E.: Random-value payment tokens for on-chain privacy-preserving payments. In: CoopIS. pp. 223–241 (2022)

[10] A preliminary version is available at github.com/apwbs/MARTSIA-Algorand

17. Klinkmüller, C., Ponomarev, A., Tran, A.B., Weber, I., van der Aalst, W.M.P.: Mining blockchain processes: Extracting process mining data from blockchain applications. In: BPM Blockchain and CEE forum. pp. 71–86 (2019)
18. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: SP. pp. 839–858 (2016)
19. Li, B., Wang, Y.: RZKPB: A privacy-preserving blockchain-based fair transaction method for sharing economy. In: TrustCom/BigDataSE. pp. 1164–1169 (2018)
20. Liu, Z., Jiang, Z.L., Wang, X., et al.: Multi-authority ciphertext policy attribute-based encryption scheme on ideal lattices. ISPA/IUCC/BDCloud/SocialCom/SustainCom pp. 1003–1008 (2018)
21. López-Pintado, O., Dumas, M., García-Bañuelos, L., Weber, I.: Controlled flexibility in blockchain-based collaborative business processes. Inf. Syst. **104**, 101622 (2022)
22. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: A business process execution engine on the Ethereum blockchain. Softw., Pract. Exper. **49**(7), 1162–1193 (2019)
23. Madsen, M.F., Gaub, M., Høgnason, T., et al.: Collaboration among adversaries: Distributed workflow execution on a blockchain. In: FAB. pp. 8–15 (2018)
24. Marangone, E., Di Ciccio, C., Weber, I.: Fine-grained data access control for collaborative process execution on blockchain. In: BPM Blockchain and RPA Forum. pp. 51–67 (2022)
25. Mendling, J., Weber, I., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Manage. Inf. Syst. **9**(1), 4:1–4:16 (2018)
26. Miller, V.S.: Use of elliptic curves in cryptography. Springer (1986)
27. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
28. National Institute of Standards and Technology: Announcing the Advanced Encryption Standard (AES). Tech. rep., Federal Information Processing Standards Publication 197 (2001)
29. Norta, A., Matulevicius, R., Leiding, B.: Safeguarding a formalized blockchain-enabled identity-authentication protocol by applying security risk-oriented patterns. Comput. Secur. **86**, 253–269 (2019)
30. Pham, V.D., Tran, C.T., Nguyen, T., Nguyen, T.T., et al.: B-box - a decentralized storage system using IPFS, attributed-based encryption, and blockchain. In: RIVF. pp. 1–6 (2020)
31. Pournaghi, S., Bayat, M., Farjami, Y.: Medsba: a novel and secure scheme to share medical data based on blockchain technology and attribute-based encryption. JAIHC **11** (11 2020)
32. Rahulamathavan, Y., Phan, R.C.W., Rajarajan, M., Misra, S., Kondoz, A.: Privacy-preserving blockchain based IoT ecosystem using attribute-based encryption. In: ANTS. pp. 1–6 (2017)
33. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems (reprint). Commun. ACM **26**(1), 96–99 (1983)
34. Rouselakis, Y., Waters, B.: Efficient statically-secure large-universe multi-authority attribute-based encryption. Cryptology ePrint Archive, Paper 2015/016 (2015)
35. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT. p. 457–473 (2005)
36. Stiehle, F., Weber, I.: Blockchain for business process enactment: A taxonomy and systematic literature review. In: BPM Blockchain and RPA Forum. pp. 5–20 (2022)
37. Tran, A.B., Lu, Q., Weber, I.: Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In: BPM Demos. pp. 56–60 (2018)
38. Tran, C.T., Pham, V.D., Nguyen, T., Dinh, H.H.Q., et al.: A novel approach for developing decentralized storage and sharing systems. In: BSCI. p. 85–90 (2021)
39. Wang, H., Song, Y.: Secure cloud-based ehr system using attribute-based cryptosystem and blockchain. J. Med. Syst. **42**(8), 152 (2018)
40. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: BPM. pp. 329–347 (2016)
41. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger (2014)
42. Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications. Springer (2019)
43. Zhang, R., Xue, R., Liu, L.: Security and privacy on blockchain. ACM Comput. Surv. **52**(3) (2019)