Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

AttriChain: Decentralized traceable anonymous identities in privacy-preserving permissioned blockchain

Wei Shao[a,d], Chunfu Jia[a,d,a,*], Yunkai Xu[a], Kefan Qiu[a], Yan Gao[b], Yituo He[c]

[a] College of Cyber Sciences, Nankai University, Tianjin, 300350, China
[b] School of Cyber Security and Computer, Hebei University, Tianjin, China
[c] Bell Honors School, Nanjing University of Posts and Telecommunications, Nanjin, China
[d] Tianjin Key Laboratory of Network and Data Security Technology, Tianjin, 300350, China

ARTICLE INFO

Article history:
Received 8 February 2020
Revised 19 July 2020
Accepted 26 September 2020
Available online 1 October 2020

Keywords:
Blockchain privacy
Attribute-based signatures
Identity management
Anonymity
Traceability

ABSTRACT

In this paper, we propose a framework named AttriChain that supports transactions with anonymous sender identity and threshold traceability in permissioned blockchain. AttriChain realizes on-chain access control and distributed identity governance by allowing users to create transactions anonymously using their transaction keys and their attributes. It provides unlinkability and anonymity of transactions belonging to legitimate users whereas provides accountability for those whom other members in the network regard as questionable or malicious. The design of threshold traceability of anonymous transactions allows network members to trace transactions back to the sender without violating the decentralized property of blockchain. In AttriChain, the spontaneous tracing of misbehaviors in the network distributes the power of identity governance to multiple independent tracing members instead of depending on single-point management. It significantly increases user privacy and autonomy in a permissioned network and facilitates auditing at the same time. We define the functionality and the protocol of AttriChain in the universal composable model. We also provide an implementation of an AttriChain prototype for performance analysis.

1. Introduction

As the interests towards Bitcoin (Nakamoto, 2008) blockchain as a decentralized and cryptographically secured cryptocurrency rise, permissioned or enterprise-leveled blockchain has also drawn people's attention. The permissioned setting of a blockchain is an infrastructure that has an additional access control layer to allow only certain identifiable entities to join and use the network. The layer runs on top of the blockchain to ensure that on-chain members and actions are regulated according to certain policies. In general, the permissions to join or the policies to use the network is decided by the owners of a blockchain (some people or organizations), so that they can take advantage of blockchain without sacrificing all of their authority in the system. Members or users of a permissioned blockchain are a group of pre-determined identifiable entities chosen by the owners who are granted the permissions to access the network and may play different roles. Meanwhile, the running of the system such as node voting and transaction processing is carried out by multiple consensus nodes, who eventually record transactions from members to the ledger. Since members in the network are limited and identifiable,

* Corresponding author.
E-mail address: cfjia@nankai.edu.cn (C. Jia).
https://doi.org/10.1016/j.cose.2020.102069

permissioned blockchain prefers more efficient consensus algorithms that exempt itself from resource-consuming computations to reach a global agreement on the network world states, hence making the system more efficient in comparison to permissionless blockchain. The prevalence of permissioned blockchain in business prompts the appearance of many vastly used permissioned blockchain frameworks, including Hyperledger Hyp, Quorum Quo, Corda Cor etc.

Privacy in blockchain has been a focus of research for a long time. There are mainly two requirements for blockchain privacy: *unlinkability* and *transaction privacy* Feng et al. (2018). *Unlinkability*, or identity privacy, requires that the links between a transaction and the real-world identity of its sender and all the transactions the sender has made in a blockchain network should be kept secret or indiscoverable. Meanwhile, *transaction privacy* requires that only the transaction participators of a transaction should know its content. Many mechanisms targeting the permissionless setting have been proposed to enhance blockchain privacy by design, including the basic self-sovereign pseudo-identities in Bitcoin and Ethereum, mixing services (Bonneau et al., 2014) and zero-knowledge proofs (Sasson et al., 2014). In a permissioned blockchain where some degree of control is allowed, identity privacy is usually enforced as an optional property by the identity management service providers.

While privacy is of importance, the requirement for accountability and auditability is another major concern for blockchain design. The trade-off between unlinkability and accountability is one of the important factors that shape the landscape for blockchain applications. On the one hand, although perfect privacy is expected in some scenarios, it sometimes poses a dilemma in other practice. The reason Bitcoin has high popularity among ransomware developers and money launderers (Conti et al., 2018) is just because of the hidden link between a transaction and its responsibles. On the other hand, although a permissioned blockchain values verifiable transactions and does concern privacy to some extent, there usually lack an additional layer of obscurity within the isolated network. Identity managers perform user registration and authentication to ensure accountability whereas keeping the information protected between users by allowing the use of pseudonyms, access controls or encryptions. The on-chain transaction accountability and auditability are realized at the price of inletting the managers as “super-users”. This mechanism brings some degree of centralization to a permissioned distributed ledger, which increases the risk of manipulation and may become the honeypot for privacy breach.

The dilemma inspires us to find better infrastructural solutions to take identity privacy and decentralized accountability into consideration at the same time. In this paper, we propose a framework named AttriChain that supports decentralized traceable anonymous user identity in a permissioned blockchain applications. Unlike Bitcoin that uses pseudonyms to delink a transaction and its sender, or average permissioned blockchain that relies on membership access controls, AttriChain protects sender anonymity by hiding the links between user on-chain identities and transactions. The feasibility of the idea is determined by the type of blockchain and the application requirements. First, in permissioned blockchain where users are required to be identifiable, real-world user identity registration (Azouvi et al., 2017) is unavoidable, forming the link between user real-world and on-chain identities. Second, in some applications, such as blockchain-based data sharing or contract calling, the link between user on-chain identities and transactions is not necessarily to be instantly verifiable for network security and management, as long as there exists a way to verify transaction ownership when needed.

Therefore, to protect identity privacy and enforce decentralized accountability when needed, in AttriChain, users enclose a signature in a transaction using self-generated transaction keys along with their certified attribute credentials, which is anonymous and only accountable when tracing members in the network collaboratively open it. A transaction remains unforgeable and verifiable while reveals neither the sender identity nor the attributes used to sign. The tracing responsibility of problematic transactions is distributed to multiple members to prevent tracing power abuse. In times of disputes, such as transaction illegal behaviors or compromises, a group of tracing members who possess a tracing key can publish a clue to the network as a piece of the key to open a transaction. The threshold for clue collection is defined based on the fault tolerance capability of the system so that only with enough collection of tracing clues can a signed transaction be traced to its sender. The decision for clue issuance is made spontaneously and independently by each tracing member who holds the right and the criteria to judge the legitimacy of transactions. Once a transaction is opened, the transaction sender on-chain identity is revealed and other peers can take reactive actions towards the sender, such as refusing him/her from registering more transaction public keys or enforcing revocation of his/hers other long-term credentials in the network. Otherwise, no one, including the attribute authorities, the blockchain owners, and the tracing members, can discover the link between a transaction and its sender identity, thus keeping the user anonymous while interacting with the network. To prove its practicality, we have developed a proof-of-concept implementation of the AttriChain smart contract library in a permissioned Ethereum instance and evaluate its performance using criteria including time consumption, gas cost and bandwidth overhead.

In summary, our contributions in this paper are listed as follows:

1. We propose AttriChain, a framework that allows users to interact with the network using anonymous and traceable identities in a permissioned blockchain.
2. We define the functionality of AttriChain in the universal composable (UC) model. We provide the protocol of AttriChain based on a delicate selection of the underlying cryptographic schemes that form an attribute-based signature scheme, including threshold/distributed tag-based encryption (Arita and Tsurudome, 2009) for transaction tracing, signatures for unforgeability and zero-knowledge proofs for anonymity.

3. We run a permissioned instance of Ethereum and present a prototype of AttriChain with Solidity smart contracts, performing the attributes authentication, transaction signing and tracing. We evaluate its performance and costs accordingly.

The paper is organized as follows. In Section 2, we introduce related work including user privacy in normal blockchain and existing anonymity protections. In Section 3, cryptographic primitives used in AttriChain framework are introduced as building blocks. We describe the overview of AttriChain architecture in Section 4. Section 5 and 6 present the ideal functionalities and the protocol of AttriChain. We give a thorough discussion about the anonymous but traceable identity for users in AttriChain in Section 7 and present our implementation and evaluation of AttriChain prototype in Section 8. Finally, we conclude the paper in the last two sections.

2. Related Work

Due to the immutability nature of blockchain, privacy protection should take preventive measures to avoid data leakage after it is recorded on the open ledger. In Bitcoin, the pseudonyms of users are generated by themselves without a link to their real-world identity, ensuring anonymity as long as a pseudonym is not used in multiple transactions. However, the privacy protection it provides is limited, as research finds that Bitcoin de-anonymization is feasible using transaction graph analysis (Androulaki et al., 2013b; Fleder et al., 2015; Meiklejohn et al., 2013; Nakamoto, 2008; Ron and Shamir, 2013), network analysis (Biryukov et al., 2014; Koshy et al., 2014) and data-driven address clustering (Shao et al., 2018). Countermeasures have been proposed for blockchain anonymity in the public setting, such as mixing services (Bonneau et al., 2014; Ruffing et al., 2014),van Saberhagen., ring signatures Mon, Noether et al. (2016) and zero-knowledge proofs (Miers et al., 2013; Sasson et al., 2014). Some solutions provide partial anonymity by allowing a transaction to refer to accounts of other users, which act as elements in an anonymity set (Bünz et al., 2019; Fauzi et al., 2019).

In permissioned blockchain, delicate design of identity management mechanisms is crucial to many existing frameworks. For example, Corda has the Identity Manager Service backed by the *Know-Your-Customer* (KYC) procedure that acts as the gatekeeper to the network Cor. Hyperledger Fabric uses the Membership Service Provider (MSP) based on the *Public Key Infrastructure* (PKI) to establish a trusted membership service for its users Hyp. Privacy is not regarded as an indispensable property, hence only optional tools are provided for network members, such as encryption or secret communication channels. In Hardjono and Pentland (2019), a system named ChainAnchor is proposed that supports anonymous but verifiable user identity and access control on permissioned blockchain. It introduces two additional authorities in the system, namely the permission issuer and the permission verifier, for membership issuing and user authentication respectively. A privacy-preserving token management system designed for permissioned systems is proposed in Androulaki et al. that supports fine-grained auditing. There are some efforts to use attribute-based signatures in permissioned blockchain identity management (Lin et al., 2018; Sun et al., 2018; Zhu et al., 2018). Decentralized traceable attribute-based signatures (DTABS) (El Kaafarani et al., 2014) first add a tracing authority to the scheme in a multi-authority setting to enforce signature traceability. AttriChain is enlightened by the scheme, where the signatures are used to implement on-chain membership proofs and transaction traceability.

3. Building Blocks

This section presents the cryptographic primitives that are used to build the protocol of AttriChain.

3.1. Digital signatures

Digital signatures (DS) (Rivest et al., 1978) have been widely used in many cryptosystems. Formally, a digital signature scheme consists of three polynomial-time algorithms (*keygen, sign, verify*). The key generation algorithm $(sk_{DS}, vk_{DS}) \leftarrow keygen(1^{\lambda})$ generates a pair of secret and public keys (sk_{DS}, vk_{DS}) on the input of the security parameter λ. The signing algorithm $\sigma_{DS} \leftarrow sign(sk_{DS}, m)$ outputs a signature σ_{DS} on the input of a secret signing key sk_{DS} and a message m. On the input of a public verification key vk_{DS}, a signature σ_{DS} and a message m, the verification algorithm $1/0 \leftarrow verify(vk_{DS}, m, \sigma_{DS})$ outputs 1 if σ_{DS} is valid and 0 otherwise. Digital signatures support correctness and existential unforgeability under chosen-message attacks, which are defined in Goldwasser et al. (1988). The ideal functionality of digital signatures $\mathcal{F}_{SIG}$ can be found in Appendix A.

3.2. Non-interactive zero-knowledge proofs

A zero-knowledge proof of knowledge is a case where a prover proves a statement to convey the information that the prover has the knowledge of a secret. Non-interactive zero-knowledge proofs (NIZK) (De Santis and Persiano, 1992) allows a prover to achieve the knowledge proof without revealing any information about it and without rounds or interactions between a prover and a verifier. In a NIZK system, where R denotes a relation for a statement x and a witness w and where L denotes the language of all statements in R, the scheme consists of a tuple of algorithms (*setup, prove, verify, extract, simSetup, simProof*). $(crs, xk) \leftarrow setup(1^{\lambda})$ is to be run by a trusted party to generate a common reference string crs and an extraction key xk. The prove algorithm $\pi \leftarrow prove(crs, x, w)$ outputs a proof π on the input of a reference string crs, a statement x and a witness w. The proof can be verified for validity by the algorithm $1/0 \leftarrow verify(crs, x, \pi)$. The algorithm $w \leftarrow extract(crs, xk, x, \pi)$ extracts the witness w on the input of a common reference string crs, an extraction key xk and a statement x. For the simulation setting, $(c_{sim}, td) \leftarrow simSetup(1^{\lambda})$ generates a simulated reference string c_{sim} and a trap door td to allow proof simulation. The simProof algorithm $\pi_{sim} \leftarrow simProof(c_{sim}, td, x)$ outputs a simulated proof π_{sim} on the input of a simulated reference string c_{sim}, a trap door td and a statement x. A NIZK scheme enjoys three properties: 1) completeness, where honest proofs should be correctly verified; 2) soundness, where

no prover can generate a valid proof for an invalid statement; 3) zero-knowledge, where verifiers learn nothing about the proved secret. The ideal functionality of non-interactive zero-knowledge proofs $\mathcal{F}_{NIZK}$ can be found in Appendix A.

3.3. *(Threshold) tag-based public-key encryption*

To protect a single tracing authority from being compromised or turning malicious, in AttriChain we employ a threshold/distributed transaction tracing mechanism. Let the size of the network be N and the size of the tracing authority group be n, where $n \in [1, N]$. The threshold t is defined according to the fault tolerance ϵ of the system, such that $t \in [\epsilon, n]$.

A non-interactive (threshold) tag-based public key encryption (TPKE) (Arita and Tsurudome, 2009; Ghadafi, 2014) takes a tag τ from a tag space $\mathbb{T}$ as input when encrypting and decrypting a message m from a message space $\mathbb{M}$. For a chosen threshold t, a threshold /distributed tag-based encryption scheme consists of six algorithms (*keygen, encrypt, shareDec, shareVerify, combine, cVerify*). The key generation algorithm $(pk, \{sk\}_{i=1}^{n}, \{vk\}_{i=1}^{n}) \leftarrow keygen(1^{\lambda}, t, n)$ outputs a public key pk, a set of secret keys $\{sk\}_{i=1}^{n}$ and a set of verification keys $\{vk\}_{i=1}^{n}$ on the input of the security parameter λ, a threshold t and the size of the distribution n. The algorithm $c \leftarrow encrypt(pk, \tau, m)$ takes the public key pk, a tag τ and a message m as input and outputs a ciphertext c. For decryption, multiple participators run $\nu_i \leftarrow shareDec(pk, sk_i, \tau, c)$ on the input of a public key pk, a secret key sk_i, a tag τ and a ciphertext c and obtain a decryption share ν_i or the rejection symbol $\perp$. Anyone can run $1/0 \leftarrow shareVerify(pk, vk_i, \tau, c, \nu_i)$ to verify a decryption share providing a public key pk, a verification key vk_i, a tag τ, a ciphertext c and a decryption share ν_i. The algorithm $m \leftarrow combine(pk, \{vk_i\}_{i=1}^{p}, \tau, c, \{\nu_i\}_{i=1}^{q})$ takes a public key pk, p number of verification keys $\{vk_i\}_{i=1}^{p}$, a tag τ, a ciphertext c and q number of decryption shares $\{\nu_i\}_{i=1}^{q}$ as input and outputs the plaintext message m if $p, q \geq t$. The decryption can be verified by the optional algorithm $1/0 \leftarrow cVerify(pk, \tau, m, c)$. TPKE supports three security properties: 1) correctness; 2) selective-tag weak indistinguishability against adaptive chosen-ciphertext attacks (ST-wIND-CCA) (Kiltz, 2006), which informally means that an adversary cannot distinguish which message is encrypted when he has access to a decryption oracle under any tag different from the tag used to encrypt the challenge messages; 3) decryption consistency, which means a ciphertext can not be decrypted into two different messages (El Kaafarani et al., 2014). The ideal functionality $\mathcal{F}_{TPKE}$ is shown in Fig. 2.

4. AttriChain architecture

In an account-based permissioned blockchain framework, the transfer of transactions reflects the interactions between users represented by their long-term anonymous credentials or addresses. The validation of transactions in the design of AttriChain should guarantee that :

1. The owner of a transaction is a rightful person to make transactions in the network.
2. The content of a transaction should only be accessible to members in the network.
3. The transaction should be unlinkable if it is honestly generated.
4. The transaction should be accountable by multiple tracing authorities if it is malicious.

AttriChain protects sender privacy by allowing users to sign their transactions using their attribute credentials instead of just using their public keys. The use of attributes masks the user identities and enables fine-grained attribute-based permissions and access control. Transactions are verified to be issued by a legitimate user whereas revealing only the fact that the transaction satisfies the predetermined access control policy. Neither its linkability to the sender nor the attributes the sender used to sign is disclosed throughout the transaction process.

AttriChain allows a user to remain anonymous to all other peers as long as it behaves according to the network consensus. However, if a tracing authority regards a transaction as problematic or suspicious, it spontaneously votes for tracing the owner by releasing its tracing clue. When a majority of tracing members, i.e. more than a threshold, publish their tracing clue on one transaction, the anonymity of this transaction and its owner is no longer protected and the accountability of the transaction is enforceable by everyone with a sufficient collection of the tracing clues.

4.1. *Entities in AttriChain*

As shown in Fig. 1, there are five different roles in AttriChain:

1. **The chain owner.** As we assume that AttriChain is based on the permissioned blockchain framework, the chain is owned by an individual or an organization, whom we refer to as the chain owner. The access control or the signing policy, i.e. the policy of who can make what transactions in the network, is decided by the owner. The distribution of tracing keys to the volunteered or chosen tracing members is also the responsibility of the owner prior to the initiation of the network.
2. **Attribute authorities.** There can be many authorities that manage and certify network members and their attributes in the multi-authority setting. Multiple attribute authorities manage different attributes of users so that as long as there exists one honest authority, a user can correctly sign a transaction. They receive requests from identifiable users, authenticate their attributes and compute credentials associated to them. They have long term credentials that are linked to their identity, marking down their special responsibility.
3. **Users.** Users or members of the blockchain are entities who join and make interactions with the network anonymously using their attribute credentials issued by some attribute authorities. Some of them can be tracing members who can attribute to the opening of illegal transactions.
4. **Consensus nodes.** They are entities who process and verify transactions from users to ensure that a transaction is from a legitimate user and meets the network

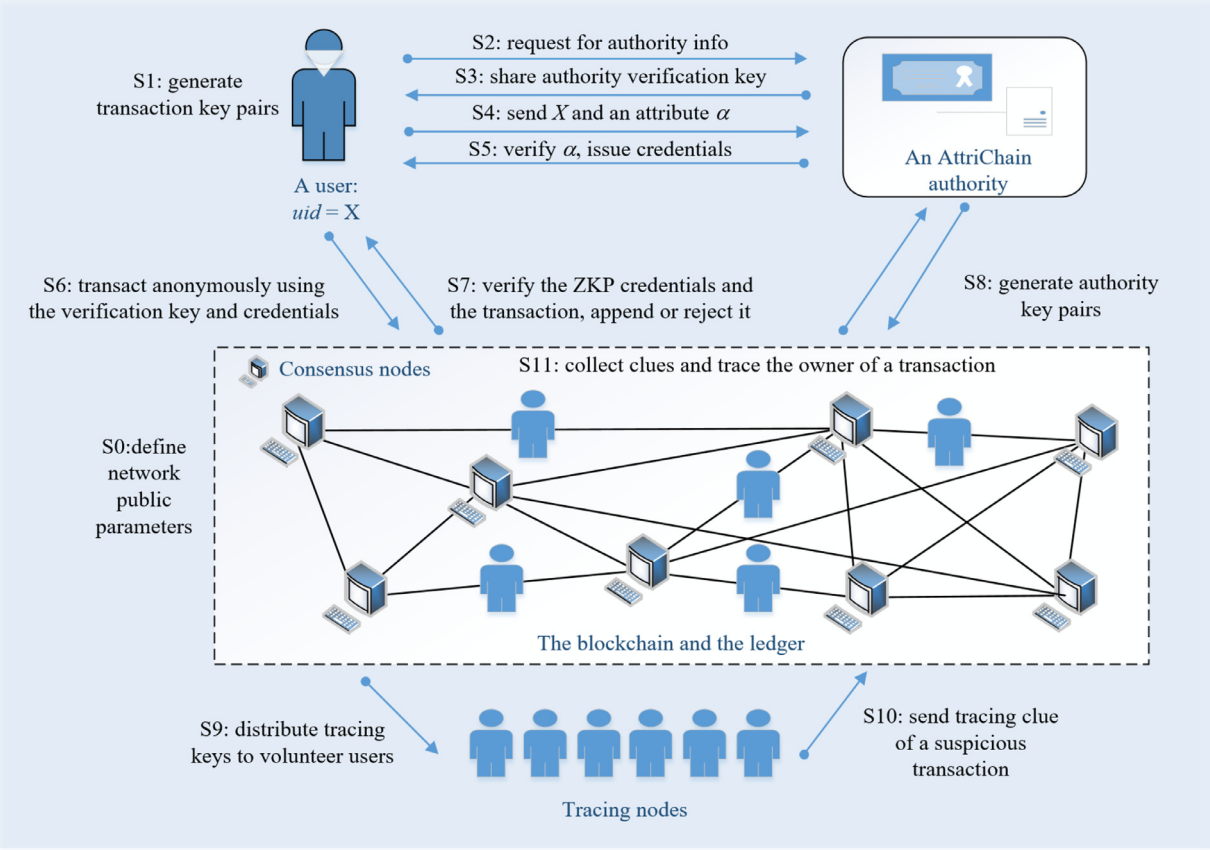


Fig. 1 – The architecture of Attrichain.

consensus requirements. Transactions that are verified by the consensus nodes can be appended to the ledger.

5. **Tracing nodes.** The tracing nodes or members are a predetermined group of users who volunteer or are randomly chosen to take the tracing responsibility. It is not necessary for them to trust or even know the existence of each other. Their priority is to release their tracing clue to the network when needed. Other than this, they use the network as normal blockchain members.

4.2. *The features*

The design of AttriChain considers the following features:

1. **Anonymous and unforgeable transaction identities.** In AttriChain, users register a unique identifier that can be specifically mapped to themselves in the real world, such as emails or telephone numbers. User privacy is maintained as a decentralized traceable anonymous identity is enclosed in transactions that are unlinkable to this identifier of users. The zero-knowledge membership proofs and signature-based ownership proofs allow transactions to remain sender-anonymous and unforgeable within the blockchain.
2. **On-chain access control.** A member whose attributes can be anonymously verified can comply with attribute-based access control in the blockchain. After the verification of whether a transaction contains proofs satisfying the access control policies, consensus nodes only process transactions from adequate members.
3. **Accountability.** The audibility and accountability allow network members to enforce inner security governance without introducing additional centralized parties. In times of disputes, tracing members can collaboratively enforce the tracing of problematic transactions. It is required that all valid transactions can be correctly traced to its owner and no adversary can forge transactions that trace to another honest member. An adversary should compromise at least t tracing authorities to de-anonymize a transaction, which is very difficult.

4.3. *Security models*

We list the security models and assumptions for each type of entity as follows.

1. **Attribute authorities.** In a multi-authority setting, for a member who possesses an attribute set $\{\mathbb{A}\} = \alpha_1, \alpha_2, ..., \alpha_p$ that satisfies $\phi(some\ \alpha \in \mathbb{A}) = 1$, where ϕ is the signing policy, we assume that at least one attribute authority who manages at least one attribute α_i should be honest for a transaction to remain unforgeable. A member provides its identifier and attributes in exchange for a secret credential that s the attribute to its identity. Considering the traceability of a transaction, we assume all authorities to be honest so that no au-

(Threshold) Tag-based public key Encryption functionality $\mathcal{F}_{TPKE}$

Functionality $\mathcal{F}_{TPKE}$ keeps an initially empty set of $\mathcal{E}$ for encrypted messages, a set $\mathcal{D}$ for decryption shares, a set $\mathcal{K}$ for public keys. It is parametrized by a threshold t and a size n. The functionality requires a group of openers whose sid is to be $((C_1, C_2, ..., C_i), sid')$.

- Upon input $(\texttt{setup}, sid)$ from R^*, check if $sid = (R^*, sid')$ for some sid'. If not, ignore the request. Else, send $(\texttt{setup}, sid, t, n)$ to $\mathcal{A}$. Upon receiving $(\texttt{set}, sid, pk, \{vk_i\}_{i=1}^n)$ from $\mathcal{A}$, send $(\texttt{set}, sid, pk, \{vk_i\}_{i=1}^n)$ to R^* and record (C_i, vk_i) in $\mathcal{K}$.
- Upon input $\texttt{pubkey}$ from a party P, return $(\texttt{pubkey}, \mathcal{K}, pk)$
- Upon input $(\texttt{encrypt}, sid, m, tg, pk)$ from a party P, check if $sid = (P, sid')$ for some $sid' \wedge m \in \mathbb{M} \wedge tg \in \mathbb{T} \wedge pk = pk'$. If so, send $(\texttt{encrypt}, sid, tg, pk)$ to $\mathcal{A}$. Else, send $(\texttt{dummy-encrypt}, sid, tg, m, pk)$ to $\mathcal{A}$. Upon receiving $(\texttt{encrypted}, sid, tg, pk, c)$ from $\mathcal{A}$, output $(\texttt{encrypted}, sid, tg, pk, c)$ to P and record (m, tg, c) in $\mathcal{E}$.
- Upon input $(\texttt{decrypt}, sid, tg, pk, c)$ from C_i, verify $sid = (C_i, sid') \wedge (\cdot, tg, c) \in \mathcal{E} \wedge pk = pk' \wedge R^*$ is not corrupt $\wedge C_i$ is not corrupt. If so, send $(\texttt{decrypt}, sid, tg, pk, c)$ to $\mathcal{A}$. Else, return $(\texttt{decrypt}, \perp)$ to C_i. Upon receiving $(\texttt{decshare}, sid, tg, pk, c, \nu_i)$ from $\mathcal{A}$, output $(\texttt{decshare}, sid, tg, pk, c, \nu_i)$ to C_i and record (m, tg, c, ν_i) in $\mathcal{D}$.
- Upon input $(\texttt{decryptC}, sid, tg, pk, c, \{vk_i\}_{i=1}^p, \{\nu_i\}_{i=1}^q)$ from a party P, If for every vk_i in $\{vk_i\}_{i=1}^p$, $vk_i \in \mathcal{K}$ and for every ν_i in $\{\nu_i\}_{i=1}^q$, $(m, c, tg, \nu_i) \in \mathcal{D}$ and $p, q \geq t$ and $pk = pk'$, do:
  . If $(m, tg, c) \in \mathcal{E}$, send $(\texttt{decrypted}, sid, tg, pk, c, m)$ to P.
  . Else, send $(\texttt{decryptC}, sid, tg, pk, c, \{vk_i\}_{i=1}^p, \{\nu_i\}_{i=1}^q)$ to $\mathcal{A}$. Upon receiving $(\texttt{decrypted}, sid, tg, pk, c, m')$ from $\mathcal{A}$, if (m', tg, c) is not recorded for any $(\bullet, tg, c)$, output $(\texttt{decrypted}, sid, tg, pk, c, m')$ to P and record (m', tg, c) in $\mathcal{E}$.(decryption consistency)

  Else, return $(\texttt{decrypted}, \perp)$ to P.

Fig. 2 – (Threshold) tag-based encryption functionality $\mathcal{F}_{TPKE}$.

thority should issue dummy credentials. However, even if all attribute authority is compromised, the anonymity of a transaction is still protected.

2. **Members of AttriChain.** Users may try to compromise the security of the system, undermine the privacy of honest users by de-anonymizing transactions or try access the network without permissions. We require the anonymity and traceability of AttriChain transaction to hold.
3. **Tracing members.** Tracing members use a piece of a tracing key to unveil the owner of a transaction when needed. However, as long as no more than t tracing secret keys are learned by the attackers, the anonymity of transactions holds.
4. **The consensus nodes and the ledger.** We assume the underlying consensus algorithm is secure for simplicity. Signature verification of a transaction is performed by consensus nodes as an additional step to the consensus algorithm. We assume that the content of a transaction, apart from the enclosed ownership proofs, is unlinkable to the sender. We also assume that the ledger is time-stamped and immutable.

5. The functionalities

In this section, we define the ideal functionality of decentralized traceable anonymous identities for transactions $\mathcal{F}_{DTAI}$ and some functionalities used in the AttriChain protocol, us-

ing the universal composition (UC) and multi-protocol UC framework (Camenisch et al., 2019; Canetti, 2001). In the UC framework, entities including protocols, functionalities, adversaries and environments all play a part in the protocol execution. Ideally, honest parties in environment $\mathcal{E}$ provide inputs to an ideal functionality $\mathcal{F}$ and receive outputs from it. However, in the real world, the protocol execution may be affected by an adversary $\mathcal{A}$ who controls some corrupt parties. Following the simulator paradigm, a simulator $\mathcal{S}$ in the ideal world will launch every equivalent attack on the ideal functionality as the adversary dose in the real world. Informally, we say a protocol Π realizes a functionality $\mathcal{F}$ if there is no polynomial-time environment $\mathcal{E}$ can distinguish $(\Pi, \mathcal{A})$ from $(\mathcal{F}, \mathcal{S})$.

5.1. *Notations*

In this section, we describe the notations used to define the ideal functionalities. We use pseudocode with an obvious semantics to describe each function, where all manipulated values are strings, booleans or the error symbol $\perp$. We use `typewriter` fonts to denote string constants. Public lists are denoted by calligraphic capital letters, such as $\mathcal{V}$ for user registration and $\mathcal{L}$ for the ledger. For better readability, we use subscripts for some symbols to indicate their ownership. For example, v_{tid} means the tracing clue v generated by tracing member tid. The detailed notations are listed below.

U, A, T, O^*, C	the set of users, attribute authorities, tracing members, chain owners, consensus nodes
pp	public parameters
t, n, ϕ, pk	tracing threshold, size, access policy, tracing public key
(P, pid, pvk)	participator, his identifier, his verification key, for $P \in (U, A, T)$. Replace their first letter for a known set, e.g. if $P = U$, the tuple becomes (U, uid, uvk).
msk	master key for transaction encyption key derivation
$\alpha, \mathbb{A}, A_{aid(\alpha)}$	attribute, attribute set, α-managing authority
$tx, \mathbb{P}, \sigma$	transaction content, signing policy, anonymous signature
$\{v_{tid}\}_{i=1}^{q}, \pi$	a set of tracing clues, tracing proof

5.2. *The ideal functionality of $\mathcal{F}_{DTAI}$*

The session identifier of functionality $\mathcal{F}_{DTAI}$ is of the form (U, A, T, O^*, C, sid'), where U, A, T, O^*, C represent users, attribute authorities, tracing members, the chain owner and consensus nodes respectively.

Setup:

1. Upon input $(\texttt{setup}, sid)$ from a party O^*, check if $sid = (O^*, sid')$. If not, ignore the request. Set $pp = (t, n, \phi, pk)$. If (O^*, sid, pp, msk) is recorded, return $(\texttt{set}, sid, pp)$ to O^*. Else send $(\texttt{setup}, sid)$ to $\mathcal{A}$, and receive $(\texttt{set}, sid, pp, msk)$. Record (O^*, sid, pp, msk).
2. Upon input $(\texttt{register}, sid, pid)$ from $P \in (U, A, T)$, check if $sid = (P, sid')$. If not, halt. Send $(\texttt{register}, sid, pid, P)$ to $\mathcal{A}$ and receive $(\texttt{registered}, sid, pid, P, pvk)$. Record (P, pid, pvk) in $\mathcal{V}$ and return $(\texttt{registered}, sid, pid, pvk)$ to P.

Attribute certification:

1. Upon input $(\texttt{request}, sid, uid, \alpha)$ from a party U_i, check if $sid = (U_i, sid')$ and $(U_i, uid, uvk) \in \mathcal{V}$. If not, ignore the request. Send $(\texttt{request}, sid, uid, \alpha, U_i)$ to $A_{aid(\alpha)}$ and mark it as an unprocessed request.
2. Upon input $(\texttt{aCert}, sid, pp', uid, \alpha)$ from a party A_i, check if $sid = (A_i, sid')$ and there is an unprocessed request $(\texttt{request}, sid, uid, \alpha, U_i)$ and $A_i = A_{aid(\alpha)}$ and $pp' = pp$. If not, ignore the request. If A_i is corrupt, return $(\texttt{aCerted}, sid, uid, \alpha, \perp)$ to U_i and record $(uid, \alpha, \perp)$. If there exist a record $(uid, \alpha, \sigma_{uid,\alpha})$, return. Else, check that $\alpha \in \mathbb{A}_{uid}$ for (O^*, sid', pp, msk). If so, send $(\texttt{aCert}, sid, pp', uid, \alpha)$ to $\mathcal{A}$. Upon receiving $(\texttt{aCerted}, sid, uid, \alpha, \sigma_{uid,\alpha})$ from $\mathcal{A}$, record $(uid, \alpha, \sigma_{uid,\alpha})$.

Transaction:

1. Upon input $(\texttt{transfer}, sid, pp', uid', \mathbb{A}' = \{\alpha\}_{i=1}^{p}, tx, \mathbb{P})$ from a party U_i, check $sid = (U_i, sid')$ and $pp' = pp$. If not, ignore the request. Check that for every $\alpha \in \mathbb{A}'$, $(uid, \alpha, \sigma_{uid,\alpha})$ is recorded where $uid = uid'$ and $\sigma_{uid,\alpha} \neq \perp$, and $\mathbb{P} \subseteq \phi$ and $\mathbb{P}(\mathbb{A}') = 1$. If so, send $(\texttt{sign}, sid, pp', tx, \mathbb{P})$ to $\mathcal{A}$ and receive $(\texttt{signature}, sid, pp', tx, \mathbb{P}, \sigma)$.
   1. If $(sid, uid', \mathbb{A}', tx, \mathbb{P}, \sigma, 0)$ recorded, return error.
   2. Else record $(sid, uid', \mathbb{A}', tx, \mathbb{P}, \sigma, 1)$.

   $TX = Enc_{msk}(tx), \mathbb{P}, \sigma$ $(\texttt{transaction}, sid, uid, TX)$ t
2. Upon input $(\texttt{verify}, sid, pp', TX)$ from a party C, check $sid = (P, sid')$ and $pp' = pp$ and there is an unprocessed request $(\texttt{transaction}, sid, uid, TX)$. If not, ignore the request. Decrypt and parse TX into $(tx', \mathbb{P}', \sigma')$.
   1. If $(sid, uid, \mathbb{A}, tx', \mathbb{P}', \sigma', f)$ is recorded, set $f_{out} = f$.
   2. Else if $(sid, uid, \mathbb{A}, tx', \mathbb{P}', \sigma', 1)$ exists or some A or U is corrupt s.t. $\exists \alpha$ with a record $(uid, \alpha, \cdot)$ s.t. $\mathbb{P}'(\mathbb{A}' \cup \alpha) = 1$, send $(\texttt{verify}, sid, pp, tx', \mathbb{P}', \sigma')$ to $\mathcal{A}$ and obtain b. Set $f_{out} = b$.
   3. Else, set $f_{out} = 0$.

   Output $(\texttt{verified}, sid, pp, TX, f_{out})$ to C and record $(sid, uid, \mathbb{A}, tx', \mathbb{P}', \sigma', f_{out})$. Let $\mathcal{L}$ be an initially empty list and append TX to $\mathcal{L}$. Send $(\texttt{transferred}, \mathcal{L})$ to $\mathcal{A}$. Return $(\texttt{transferred}, sid, uid', pp', \mathbb{A}', tx, \mathbb{P})$ to U_i.

Threshold tracing:

1. Upon input $(\texttt{tTrace}, sid, pp', tid)$ from a party T, check $sid = (T, sid')$ and $pp' = pp$ and $(T, tid, tvk) \in \mathcal{V}$. If not, ignore the request. Query $\mathcal{L}$ for a target element $TX = Enc_{msk}(tx'), \mathbb{P}', \sigma'$. Decrypt and parse it into $(tx', \mathbb{P}', \sigma')$. If T is corrupt, return $(\texttt{tTrace}, sid, pp', \sigma', \perp)$ and record $(sid, tid, tx', \mathbb{P}', \sigma', \perp)$.
   1. If $(sid, tid, tx', \mathbb{P}', \sigma', v_{tid})$ recorded, use it.
   2. Else send $(\texttt{tTrace}, pp, sid, tid, tx', \mathbb{P}', \sigma')$ to $\mathcal{A}$ and receive $(\texttt{clue}, pp, sid, tid, tx', \mathbb{P}', \sigma', v_{tid})$.

   Append $(\texttt{clue}, (TX, v_{tid}))$ to $\mathcal{L}$.
2. Upon input $(\texttt{reveal}, pp, sid, tx', \mathbb{P}', \sigma', \{v_{tid}\}_{i=1}^{q})$ from a party P, check $sid = (P, sid')$ and $pp' = pp$. If not, ignore the request. If for every $v_{tid} \in \{v_{tid}\}_{i=1}^{q}$, $(sid, tid, tx', \mathbb{P}', \sigma', v_{tid})$ and $(sid, uid, \mathbb{A}, tx', \mathbb{P}', \sigma', 1)$ is recorded and $q \geq t$, do:

1. If $(sid, uid, tx', \mathbb{P}', \sigma', \pi)$ is recorded, use it.
2. Else send $(\texttt{reveal}, pp, sid, tx', \mathbb{P}', \sigma', \{v_{tid}\}_{i=1}^{q})$ to $\mathcal{A}$ and receive $(\texttt{proof}, pp, sid, tx', \mathbb{P}', \sigma', \pi)$. Record $(sid, uid, tx', \mathbb{P}', \sigma', \pi)$.

Return $(\texttt{revealed}, pp, sid, TX, uid, \pi)$.

Setup: The `setup` message allows the blockchain owner to initiate its own instance of the functionality. The adversary provides the public parameters including the size and the threshold of the instance, the access control predicate and a membership secret key. The `register` is for the registration of members, attribute authorities and tracing members. The $\mathcal{F}_{DTAI}$ allows the adversary to generate key pairs for them that are used as transactions keys for users and long-term credentials for attribute and tracing authorities.

Attribute certification: A party U_i queries for the certification of its attributes by the message `request`. The attribute authority $A_{aid(\alpha)}$ authenticates the attribute α when it receives an unprocessed request $(\texttt{request}, sid, uid, pp, \alpha, U_i)$ from user U_i. If the party $A_{aid(\alpha)}$ is corrupt, $\mathcal{F}_{DTAI}$ records $\perp$ as an indication of certification failure. For an honest A, the ideal functionality $\mathcal{F}_{DTAI}$ checks if the attribute is valid and belongs to the user under the same setting. If so, it asks the adversary to generate a credential on (uid, α). The attribute credentials are recorded to ensure attribute certification consistency and unforgeability.

Transaction: The `transfer` message is sent by a user U_i to create transactions based on its decentralized traceable anonymous identity. $\mathcal{F}_{DTAI}$ first checks three conditions for the request: 1) the provided attributes have all been certified with a valid credential; 2) the provided signing predicate $\mathbb{P}$ is a sufficient condition for the global access control policy ϕ; 3) the provided attributes satisfies the predicate. After the checks, the activation asks the adversary for a signature under the public parameters on $(tx, \mathbb{P})$. The adversary returns a signature without the knowledge of the identity uid and the attribute set of U_i, which guarantees privacy. The signature is recorded and enclosed as a component of a transaction to anonymously prove the unforgeable ownership of the sender. The content of a transaction tx, which could vary in different scenarios (such as a file in a blockchain-based data sharing system), is encrypted under the membership secret key to allow confidentiality. The verification of a transaction ensures its integrity and compliance by checking if the enclosed signature is correspondent to the user and his attribute set recorded by the `transfer` message, or if there exists corrupted attributes s.t. $\mathbb{P}'(\mathbb{A}') = 1$. When the verification is successful, the transaction TX is appended to the ledger.

Threshold tracing: The `tTrace` message is sent by a tracing member T to enforce accountability of a transaction in ledger $\mathcal{L}$. After checking the session identity and the public parameter, $\mathcal{F}_{DTAI}$ asks the adversary for a share of tracing clue derived from his secret tracing key of tsk. Other parties can trace the sender of a transaction in $\mathcal{L}$ if they have q pieces of valid clues if $q \geq t$ by the message `reveal`.

5.3. *Other functionalities*

Our protocol requires a number of functionalities. Some of them are widely used and therefore are listed in Appendix A in detail. They include:

1. *Digital signatures* (Canetti, 2004). The functionality $\mathcal{F}_{SIG}$ shown in Fig. A.4 guarantees authenticated communication over the unauthenticated blockchain and bind transactions to the identifiable entities in the network.
2. *Non-interactive zero-knowledge proof* (Camenisch et al., 2019). The functionality $\mathcal{F}_{NIZK}$ in Fig. A.5 adopts the most commonly used non-interactive zero-knowledge functionality that allows zero-knowledge proofs of knowledge.
3. *Transaction Ledger* (Androulaki et al., 2013a). The functionality $\mathcal{F}_{Ledger}$, shown in Fig. A.6, is a simplified ledger where transactions are verified, appended to and become available to all parties.
4. *Secure message transmission* (Canetti and Krawczyk, 2002). The functionality $\mathcal{F}_{SMT}$ shown in Fig. A.7 provides secure message transmission channels between two parties.
5. *Public key registration.* The functionality $\mathcal{F}_{PKR}$ in Fig. A.8 allows the publication of public keys and their links to a party under the public-key infrastructure.

Threshold tag-based public-key encryption. The threshold traceability of transactions is resolved by the idea of secret sharing, which informally means separating a secret into shares of secrets and distributing them to a group of entities. Each entity has a secret share, with which t of n can be combined to compute the secret. Based on the idea, we consider a group of tracing members who should collaboratively open a transaction corresponding to a user using the functionality of threshold tag-based public-key encryption $\mathcal{F}_{TPKE}$. The functionality of $\mathcal{F}_{TPKE}$ is shown in Fig. 2. A party R sends the message `setup` to ask for the public key pk and the decryption public key $vk \in \{vk_i\}_{i=1}^{n}$ for every C_i. The message `encrypt` checks that the message m and the tag tg are elements from their space $\mathbb{M}$ and $\mathbb{T}$ and outputs a ciphertext c provided by the adversary without knowledge of m.

When issuing a transaction, the sender generates a fresh key pair (fsk, fvk) and uses it to anonymously sign the transaction for unlinkability. The bond between the fvk and the sender public key uvk, which is linkable to the user identity, is regarded as a secret. To other parties or individual tracing members, the secret is encrypted using $\mathcal{F}_{TPKE}$ where fvk is the tag and uvk is the plaintext. When accountability of a transaction should be enforced, the tracing members, who are in possession of a tracing key pair, can compute their secret share, with which t out of n can be combined to unveil the plaintext.

6. The protocol

This section describes the protocol from the perspectives of different roles in the UC model and provides a security analysis of AttriChain protocol. The overview of AttriChain protocol is shown in Fig. 1. We adopt the notational convention in the previous section for ease of reading. Besides, we use the upright font to indicate boolean status.

6.1. The setup

The initiation of AttriChain protocol mainly includes the setup of global parameters by the blockchain owner, such as the size of the tracing group and the access control policies (S0). Note that the signing policy ϕ, which is expressed as an attribute-based predicate, defines the fine-grained access control for AttriChain.

Formally, the setup protocol keeps a boolean bit *init*, which is initially set as *init* = false to indicate the initialization state of the protocol.

1. Upon input $(\texttt{init}, sid, t, n, \phi)$ from O^*, check if $sid = (O^*, sid')$ for some sid' and $init = false$. If not, ignore the request. Obtain $pk, \{vk_i\}_{i=1}^{n}$ by sending $(\texttt{setup}, sid)$ to $\mathcal{F}_{TPKE}$. Generate a membership key msk and send $(\texttt{Membership}, msk)$ to $\mathcal{A}$. When receiving $\texttt{ok}$ from $\mathcal{A}$, record $(O^*, sid, t, n, \phi, pk, msk)$. Set $init = true$.

6.2. The users

As Fig. 1 shows, a regular user first generates its public key pairs as transaction keys (S1). In a standard blockchain, a user signs their transactions using self-generated transaction keys to protect transaction integrity. Similarly in AttriChain, users can generate as many transaction key pairs as they want. For each key pair, the user registers the public key to a publicly available list maintained on the blockchain recording their mapping with the user identity. Before requesting credential certification, a user should know the attribute authorities, including their public keys and the attributes they manage. The information of all registered attribute authorities should be public to all who can access the ledger (S2,S3). The user can then send requests for attribute certification to attribute authorities who manage some of its attributes α providing its identity *uid* (S4). At this step, the user is not anonymous to the attribute authorities as the identity of users should be known for attribute authentication. If the authentication is successful, the user receives attribute credentials from the attribute authorities through a secret channel (S5). A user creates transactions using its public keys for accountability, the membership secret key for on-chain data protection, and also zero-knowledge attribute credentials for anonymity and access control (S6). Any compilation of attribute credentials can be used for the transaction signature as long as it suffices the access control predicate. To other entities in AttriChain, they are not able to link a transaction to its originator except knowing that the transaction is from someone who conforms with the signing policy ϕ.

In AttriChain, regular users can collect tracing clues from the ledger to unveil a user public key enclosed in a transaction and hence the identity of the sender (S11). At this point, the transaction is no longer anonymous in the network. However, other transactions of the sender are still anonymous and unlinkable.

Formally, the protocol of AttriChain members has parameters including a boolean *reg* for the registration state of a member that is initially set as *false*. It keeps a private and initially empty list $\mathcal{C}$ for certified attribute credentials and a list $\mathcal{T}$ for tracing clues.

1. Upon input $(\texttt{register}, sid)$ from a party U, check if $sid = (U, sid')$ for some sid' and $reg = false$. If not, return. Send $(\texttt{keygen}, sid)$ to $\mathcal{F}_{SIG}$ and retrieve the transaction public key uvk. Register (U, uid, uvk_U) via $\mathcal{F}_{PKR}$. Set $reg = true$.
2. Upon input $(\texttt{request}, sid, uid, \alpha, A_i)$ from a party U, check if $sid = (U, sid')$ for some sid' and $reg = true$. Do:
   1. Send $(\texttt{aCert}, sid, (uid, \alpha))$ to A_i of identity aid via $\mathcal{F}_{SMT}$.
   2. From $\mathcal{F}_{SMT}$, retrieve $(\texttt{aCert}, sid, uid, \alpha, \sigma_{uid,\alpha})$.
   3. Record $(uid, \alpha, \sigma_{uid,\alpha})$ in $\mathcal{C}$.
3. Upon input $(\texttt{transfer}, sid, pp', uid, \mathbb{A} = \{\alpha\}_{i=1}^{p}, tx, \mathbb{P})$ from a party U, check if $sid = (U, sid')$ for some sid' and $reg = true$ and $pp' = pp$. If not, ignore the request. Check that for every $\alpha \in \mathbb{A}$, $(uid, \alpha, \sigma_{uid,\alpha})$ is recorded and $\sigma_{uid,\alpha} \neq \perp$ and $\mathbb{P} \subseteq \phi$ and $\mathbb{P}(\mathbb{A}) = 1$. If so, do:
   1. Generate a fresh pair of keys (fsk, fvk) and compute a credential σ_{fvk} for fvk by sending $(\texttt{sign}, sid, uvk_{uid}, fvk)$ to $\mathcal{F}_{SIG}$ and sign fvk under its secret key.
   2. Compute an encryption c of uvk_{uid} using $\mathcal{F}_{TPKE}$ where fvk is the tag.
   3. Compute a prove π by sending $\mathcal{F}_{NIZK}$ the input $(\texttt{prove}, x, w)$ where $x = (c, pk, \{avk_{aid(\alpha)}\}_{\alpha\in\phi}, \{\alpha_i\}_{i=1}^{\phi})$, and $w = (uvk_{uid}, \{\sigma_{uid,\alpha}\}_{i=1}^{\phi}, \sigma_{fvk})$.
   4. Compute a signature σ_s on (tx, ϕ, π, c, fvk) by the fsk, $\sigma_s \leftarrow sign(fsk, (tx, \phi, \pi, c, fvk))$.

   $(\texttt{transaction}, TX)$ $\Gamma = (c, \pi, \sigma_s, fvk)$ a $TX = Enc_{msk}(tx), \Gamma$. $(\texttt{transferred}, tx)$ to U.
4. Upon input $\texttt{reveal}$ from a party P, obtain tracing clues from the ledger by sending $\texttt{retrieve}$ to $\mathcal{F}_{Ledger}$. Record (tx, v_i) in a list $\mathcal{T}$. For each retrieved transaction, parse $\Gamma = (c, \pi, \sigma_s, fvk)$. Query public keys $\{vk_i\}_{i=1}^{p}$ of tracing members via $(\texttt{pubkey}, sid)$ to $\mathcal{F}_{TPKE}$. Send $(\texttt{decryptC}, sid, c, fvk, pk, \{vk_i\}_{i=1}^{p}, \{v_i\}_{i=1}^{q})$ to $\mathcal{F}_{TPKE}$ and receive uvk_{uid}.

6.3. The attribute authorities

For attribute authorities who have a unique unforgeable identity *aid* that is public to all nodes in the network, they first generate a key pair corresponding to its identifier *aid* (S8). When a user requests for attribute certification, an attribute authority verifies the authenticity of the received information and compute a credential (S5).

Formally, the protocol for attribute authorities in AttriChain keeps parameters including a boolean *reg* for recording the registration state of an authority that is initially set as *false*. It records a private and initially empty list $\mathcal{B}$ for certified attribute credentials.

1. Upon input $(\texttt{register}, sid)$ from a party A_i, check if $sid = (A_i, sid')$ for some sid' and $reg = false$. If not, return. Send $(\texttt{keygen}, sid)$ to $\mathcal{F}_{SIG}$ and retrieve the authority public key avk. Register (A_i, aid, avk_{A_i}) via $\mathcal{F}_{PKR}$ and set $reg = true$.
2. Upon receiving $(\texttt{aCert}, sid, (uid, \alpha))$ from a party U from $\mathcal{F}_{SMT}$, check if (U, uid, uvk_U) is registered and uid is the identifier of U and A_i manages α. If not, return $(\texttt{aCert}, sid, (uid, \alpha), \perp)$. Else:
   1. Send $(\texttt{sign}, sid, (uid, \alpha))$ to $\mathcal{F}_{SIG}$ and obtain an attribute credential $\sigma_{uid,\alpha}$.
   2. Record locally $(uid, \alpha, \sigma_{uid,\alpha})$ in $\mathcal{B}$.

3. Send U the credential $(\texttt{aCert}, sid, (uid, \alpha), \sigma_{uid,\alpha})$ via $\mathcal{F}_{SMT}$.

6.4. The tracing members

The network initiation generates a pair of tracing keys for each tracing nodes (S0, S9). Users in the network can volunteer or be randomly selected to become tracing members. The tracing nodes have the same priority as a normal user except that it has a tracing key to contribute to the opening of a transaction. Tracing nodes do not have to trust or even know the existence of each other. The group can even cover all users to realized fully distributed identity traceability in AttriChain. When a transaction is questionable or has the possibility of been issued by a compromised user, a tracing node can compute and publish its tracing clue to the network (S10).

1. Upon input $(\texttt{register}, sid)$ from a party T_i, check if $sid = (T_i, sid')$ for some sid' and $reg = \text{false}$. If not, return. Query $\texttt{pubkey}$ to $\mathcal{F}_{TPKE}$ and receive its tracing public key tvk_{tid}. Register (T_i, tid, tvk_{tid}) via $\mathcal{F}_{PKR}$ and set $reg = \text{true}$.
2. Query $\mathcal{F}_{Ledger}$ for the entire ledger. Decrypt and parse the transaction TX as $(tx, \Gamma) = (tx, (c, \pi, \sigma_s, fvk))$. Compute the tracing clue ν by sending $(\texttt{decrypt}, sid, c, fvk, pk)$ to $\mathcal{F}_{TPKE}$ and publish it to the ledger $(\texttt{append}, (TX, \nu))$.

6.5. The consensus nodes

For the consensus nodes, apart from the original consensus algorithm, the blockchain framework applies which is omitted for simplicity in this paper, they should verify if the signed transaction is from a user satisfying the access control policy. If the transaction fails the verification, the transaction is suspended (S7).

1. Monitors new input from users for $(\texttt{transaction}, TX)$. Decrypt and parse the transaction as $(tx, \Gamma) = (tx, (c, \pi, \sigma_s, fvk))$. If (sid, tx, Γ, f) is recorded, set $b = f$. Else, do:
   1. Verify the signature σ_s on (tx, ϕ, π, c, fvk) by running $r_1 \leftarrow verify(fvk, \sigma_s, (tx, \phi, \pi, c, fvk))$.
   2. Verify the proof π by sending $(\texttt{verify}, x, \pi)$ to $\mathcal{F}_{NIZK}$, where $x = (c, pk, \{avk_{aid(\alpha)}\}_{\alpha\in\mathbb{P}}, \{\alpha_i\}_{i=1}^{\mathbb{P}})$. Let the result be r_2.

   $b = r_1 \wedge r_2$. (sid, tx, Γ, b). If $b = 1$, $\mathcal{F}_{Ledger}$.

6.6. Security analysis

This section gives the security analysis of AttriChain. We prove that the protocol of AttriChain Π_{DTAI} instantiates functionality $\mathcal{F}_{DTAI}$.

Theorem 1. The protocol Π_{DTAI} securely realizes $\mathcal{F}_{DTAI}$ in the $(\mathcal{F}_{SIG} - \mathcal{F}_{NIZK}\text{-}\mathcal{F}_{TPKE})$ -hybrid model.

We define a simulator $\mathcal{S}$ to interact with $\mathcal{F}_{DTAI}$ to simulate the interaction between an adversary $\mathcal{A}$ and the protocol Π_{DTAI} such that for all environments $\mathcal{E}$, it holds that $(\mathcal{F}_{DTAI}, \mathcal{S})$ and $(\Pi_{DTAI}, \mathcal{A})$ are indistinguishable. The sketch of proof is deferred to Appendix B.

7. System discussion

Anonymous and traceable identities. In AttriChain, the signing process is conducted by the user itself so that the transaction is anonymous to all other parties, including:

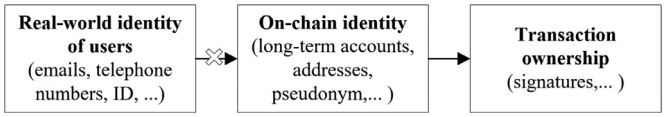
1. To the attribute authorities. The identifier of a user is known to the network during public key registration and to the attribute authorities during attribute certification. However, users send transactions signed by their secret key and attribute credentials that are hidden as commitments by NIZK, so that the other parties learns no information about the sender.
2. To consensus nodes and peer nodes. Other than aforementioned anonymity protection, to prevent the linkability between transactions owned by one owner, every time a transaction is generated, the sender generates a one-time signature key pair and encloses the public key in the output as a one-time ownership proof.
3. To tracing individuals. The tracing individuals have the same access in the blockchain as normal peers, except that they can issue a special transaction on problematic transactions containing a clue ν. Therefore, from the perspective of each individual, the anonymity property holds as it is to a normal network member.

Lemma 1. $L\Pi_{DA\ TI}$ provides full-anonymity and traceability for transactions in AttriChain, giving the assumption that the transaction content is unlinkable.

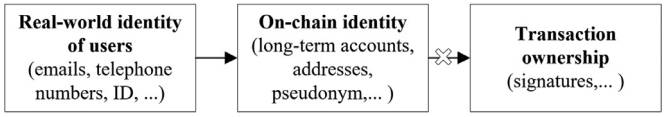
A sketch of proof can be found in Appendix C.

Comparison with other ledgers. As shown in Fig. 3, we define three identities of users in a blockchain system. First, the real-world identity of users is the identity for registration and authentication to enter a blockchain. The identifiers usually include emails, telephones, IDs, and for an inner enterprise system, maybe employee numbers and departments. Since real-world identities contain sensitive information about users, it is not wise to use them directly for on-chain activities (although many permissioned architectures allow so). The second identity is the user on-chain identities, such as long-term credentials and accounts, which act as pseudonyms to protect privacy between users. The third identity is the transaction ownership, meaning the owner identity of general transactions for on-chain interactions, which are represented as signatures. The aim of anonymity protection is to conceal the link between a real-world identity and a transaction ownership.

In public blockchain such as Bitcoin, the real-world identity of a user is not relevant to his on-chain identity since no registration is required. An on-chain identity is used to create transaction ownership and thus is directly linkable to it. It is for clear claiming of coins and double-spending prevention. Anonymity protection relies solely on the disconnected link between a user real-world identity and their on-chain pseudonyms or addresses. However, for permissioned settings, as we emphasized in previous chapters, chain owners and managers have the authority of monitoring the exact identity of whoever using the system and how they use it. Therefore if the privacy protection follows the public meth-



(a) The identity links in other blockchains



(b) The identity links in Attrichain

Fig. 3 – Instead of hiding the link between a real-world identity and an on-chain identity and publishing the link between an on-chain identity and a transaction, AttriChain conceals the relation between the latter but not the former.

ods, we must assume that the owners are trusted and capable of keeping the first link secret. Another solution is to find a trusted third party for both the owners and the users for identity management, like (Hardjono and Pentland, 2019). Yet, it still introduces additional trusted parties to the system.

AttriChain provides an improvement for permissioned blockchain as it exempts the system from entrusting the others that in practice may be dishonest or abuse its power by learning or leaking the first link. Anonymity protection is realized by allowing users to hide the second link, where the first can be known to every blockchain user. The feasibility of the idea is determined by not only the types of blockchain but also the application requirements. For many applications, instant transaction claiming is not the priority concern. For example, in a blockchain-based data sharing system, it is not necessary to instantly uncover who issues a new transaction, as long as the transaction is from a registered and legitimate user. When questionable transactions appear, AttriChain does provide a way to trace the sender, which allows chain governance by chain owners and self-auditing by all members.

Overall, AttriChain leverages privacy and accountability at the same time. Users have total control of transactions for keeping anonymity and for self-governance, whereas chain owners still have the power of knowing who and how their system is used. For achieving so, we substitute the simple digital signatures for transaction ownership for a more complicated attribute-based signature scheme, which inescapably increases the transaction load.

Discussion. In this section, we discuss the possibility of using AttriChain in public blockchain. The major problem that needs to be addressed for using AttriChain in the public setting is the choice of tracing members and the threshold. For public blockchain, users are unidentifiable and only use on-chain identities, which means the member and the size of the user group is variant and unstable. Choosing tracing members and threshold based on on-chain identities is extremely difficult and insecure. It is possible that a user can possess on-chain identities that are more than the threshold to volunteer as tracing members. Even if we only allow identifiable users to become tracing members and enforce strict member verification mechanisms, we must trust some parties to achieve it, which may rise more security concerns. If we can trust some parties for the selection of tracing threshold and the management of tracing members, using identity authentication or possibly incentive mechanisms, AttriChain is applicable to public blockchain, especially when different levels of usage priority and policies are required. The only difference is that, since public chains have no registration of user on-chain identities, the link between user real-world identity and on-chain identity is unregistered. Therefore, the use of AttriChain in public settings hides the two links in Fig. 3 at the same time.

Table 1 – Gas, time consumption and size of AttriChain contract and other functions for a network with the threshold $t = 3$, an average of 100 runs.

Functions	Time	Gas	Size(out/Byte)
$\texttt{Init}_{off}$	450 ms	-	3471
`Request`	3 μs	-	6
`Authenticate`	43 ms	-	404
`Sign`	$\approx$ 3.2 s	$\approx$ 4,68000	7688
$\texttt{Init}_{on}$	2 min 2 s	$\approx$ 35400000	-
`Trace`	4.6 s	$\approx$ 1370000	-
`Collect`	5.4 s	$\approx$ 280000	-

8. Performance analysis

Implementation. We have developed a proof-of-concept implementation of the AttriChain smart contract library in a permissioned Ethereum instance to test the practicality and performance of AttriChain. As shown in Table 1, the contract has five algorithms including `Init`, `Request`, `Authentication`, `Trace` and `Collect`. First, the `Init` calculates computes the public parameters for the network and the tracing key pairs. For attribute certification, a user issues a request of attribute authentication by calling the `Request` algorithm for each attribute providing his identity. When an attribute authority receives the request, it issues a credential to the user by the `Authentication` algorithm. Then, the user locally computes signatures for his transactions using the credentials and send them to the network. A group of tracing members who monitor the blockchain call `Trace` when a suspicious transaction

appears to issue special transactions containing the tracing clue to the public. Any user who collects enough clues can use `Collect` to trace the sender of a transaction.

The library is written in Solidity, a high-level JavaScript-like language that runs on the Ethereum Virtual Machine (EVM) for the integration of smart contracts. We also implement some functionalities with the help of Python, as we move some computations of high-consumption to the local user clients to reduce Ethereum gas consumption. A simple identity management mechanism can be implemented on-chain or off-chain for user real-world identity registration. The list of user real-world identity and on-chain identity (verification keys generated by users using our implemented digital signatures based on RSA) is stored publicly on the blockchain. We have instantiated the hash function with SHA-256 and other collision-resistant hash for mapping two elements spaces. For the instantiation of the threshold tag-based public key encryption (Ghadafi, 2014), we use the pre-compiled contract[1] for pairing operations on the alt_ bn128 curve with lower gas cost and two open-source libraries in Solidity for operations on elliptic curves on bilinear groups $\mathbb{G}_1$[2] and $\mathbb{G}_2$.[3]

To avoid significant gas costs, we implement `Request` and `Authentication` as an off-blockchain process as the two algorithms can be realized by constructing other communications channels between users and authorities, giving the condition that the attribute authorities publish their information on the blockchain. This modification to the design is proved to save a lot of gas consumptions. Meanwhile we regards the function `Init` as an optional library function since the tracing public key distribution can also be accomplished off-chain for the same reason. We provide both the on-chain and off-chain implementation to show the difference of consumption. It is worth mentioning that, although our prototype of AttriChain is implemented in the Ethereum platform, the design of AttriChain theoretically supports all permissioned frameworks.

Testbed. We have simulated the process of attribute certification and transaction issuing of a user, and the process of tracing in a permissioned blockchain with three tracing members and ten consensus nodes. The threshold is set equal as the tracing group size to realized fully-distributed tracing and stronger security. We run the prototype in a truffle 4.1.8 environment with an Intel i7-6300U processor and 8G of RAM.

Performance. Table 1 shows the execution times and gas costs for different procedures in the smart contract.i.e. the `Init`, `Trace` and `Collect` functions (below the double lines). It also shows the execution time of functions and output size realized off-chain (above the double lines). The experiment result is the average value of 100 runs for each function.

The time cost for the `Sign` function is defined as the sum of the time for signature calculation and the verification of the generated transaction. The gas is cost by sending and storing the transactions to the network.

$$T_{Sign} = T_{signature} + T_{VerTX}$$

It is worth noting that gas consumption for `Init` if it is implemented on-chain is significantly higher. That is because the generation of tracing keys involves multiple point multiplication on $\mathbb{G}_2$, which cost around 2800000 gas per operation. Comparing to the off-chain implementation, which only costs 450 ms and few storage overhead, the on-chain realization is slower and needs extra efforts to avoid reaching gas limit, such as function subdividing.

The gas] for `Trace` is cost by the verification of a signature and the calculation of a clue. This is due to the fact that we made a modification to our original design. As described in Section 6, the verification of a signed transaction is conducted by the consensus nodes, where extra conditions are verified before a transaction is recorded on the blockchain. However, the implementation of this process requires modification of the underlying code of the Ethereum consensus algorithm. Considering the work overload and the compatibility of our library, we switch the verification responsibility to the tracing members. That is, before a tracing member computes his tracing clues, he should verify the signature as Section 6.4. Therefore, the `Trace` algorithm includes the verification of NIZK proofs, three pairing checks and two elliptic curve multiplication.

We observe that the total gas consumption `Collect` is linearly related to the number of tracing members since the number of elliptic curve addition is linearly infected by the number of tracing clues q. If q valid tracing clues are collected, the times of EC addition is $2q + 1$.

We release the AttriChain Ethereum smart contract as an open-source library.[4]

Discussion. The implemented prototype proves the practicality and efficiency of AttriChain in a permissioned blockchain. However, the choice of the underlying blockchain framework has a significant impact on the actual realization of the protocol. In our implementation, for the compatibility of Ethereum and gas saving, we made two modifications to our proposed design. As mentioned previously, the realization of attribute authentication is conducted off-chain, where blockchain only provides necessary information for users and attribute authorities to build a communication channel. In addition, in order to avoid changing Ethereum encapsulated codes, the verification of signed transactions is in charge of the tracing members. This inevitably increases the gas cost of the contract.

9. Conclusion

We propose AttriChain, a framework that allows users to interact with the network using anonymous and threshold traceable identities in a permissioned blockchain. In the system, no parties directly know who a transaction is from except the sender itself, which makes it difficult for the blockchain owners to govern the system as every transaction is anonymous as long as the sender does not claim the ownership. This is addressed by the design of threshold or distributed traceability which is able to trace a transaction to the enclosed transaction public key and its sender. Identity accountability and

[1] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md.
[2] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-196.md.
[3] https://github.com/musalbas/solidity-BN256G2.
[4] https://github.com/MilkyBoat/AttriChain.

access control are enforceable at the same time but the power is not centralized in the hand of the blockchain owner. The security of blockchain is maintained while the power of identity governance is distributed to multiple tracing members to prevent transaction privacy from being maliciously or unfairly breached. The design of AttriChain adds an obscurity layer in the permissioned blockchain infrastructure. It facilitates the application of blockchain in non-financial scenarios where privacy is a concern, such as blockchain-based e-voting or social networks.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This paper is supported by The National Key Research and Development Program of China (2018YFA0704703); The National Natural Science Foundation of China (61972215, 61702399, 61972073); and The Natural Science Foundation of TianJin (20JCZDJC30500).

Appendix A. Functionalities

A1. *Digital signatures*

The functionality $\mathcal{F}_{SIG}$ guarantees authenticated communications over the unauthenticated blockchain and bind transactions to an identifiable entities in the network (Figs. A.4–A.8).

Functionality $\mathcal{F}_{SIG}$

Functionality $\mathcal{F}_{SIG}$ keeps an initially empty set of $\mathcal{S}$ for signed messages and a set $\mathcal{V}$ for verification keys.

- Upon input $(\texttt{keygen}, sid)$ from a party S, check if $sid = (S, sid')$ for some sid'. If not, ignore the request. Else, send $(\texttt{keygen}, sid)$ to $\mathcal{A}$. Upon receiving $(\texttt{verkey}, sid, vk)$ from $\mathcal{A}$, send $(\texttt{verkey}, sid, vk)$ to S and record (S, vk) in $\mathcal{V}$.
- Upon input $(\texttt{sign}, sid, m)$ from S, check if $sid = (S, sid')$ for some sid'. If not, ignore the request. Else, send $(\texttt{sign}, sid, m)$ to $\mathcal{A}$. Upon receiving $(\texttt{signature}, sid, m, \sigma)$ from $\mathcal{A}$, verify no $(m, \sigma, vk, 0)$ is recorded in $\mathcal{S}$. If it is, send $\perp$ to S, else output $(\texttt{signature}, sid, m, \sigma)$ to S and record $(m, \sigma, vk, 1)$ in $\mathcal{S}$.
- Upon input $(\texttt{verify}, sid, m, \sigma, vk')$ from a party P, send $(\texttt{verify}, sid, m, \sigma, vk')$ to $\mathcal{A}$. Upon receiving $(\texttt{verified}, sid, m, b)$ from $\mathcal{A}$.
  - If $(vk = vk')$ and $(m, \sigma, vk, 1) \in \mathcal{S}$, set $f = 1$.
  - Else, if $(vk = vk')$ and $(m, \sigma, vk, 1) \notin \mathcal{S}$ and the signer S is not corrupt, set $f = 0$ and record $(m, \sigma, vk, 0)$ in $\mathcal{S}$.
  - Else, if (m, σ, vk', f') is recored, set $f = f'$.
  - Else, set $f = b$ and record (m, σ, vk, f) in $\mathcal{S}$.

  Output $(\texttt{verified}, sid, m, f)$ to P.

Fig. A.4 – Digital signature functionality $\mathcal{F}_{SIG}$.

Functionality $\mathcal{F}_{NIZK}^{R}$

Functionality $\mathcal{F}_{NIZK}^{R}$ is parametrized by a relation R for which we can check membership. It keeps an initially empty list L of proven statement.

- Upon input $(\texttt{prove}, x, w)$ from a party P, check if $(s, w) \in R$. If not, ignore the request. Send $(\texttt{prove}, x)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{proved}, \pi)$ from $\mathcal{A}$ where $\pi \in \{0,1\}*$, record (x, π) in L and send $(\texttt{proved}, \pi)$ to P.
- Upon input $(\texttt{verify}, x, \pi)$ from a party P, check if $(x, \pi) \in L$. If not, send $(\texttt{verify}, x, \pi)$ to $\mathcal{A}$. Upon receiving $\texttt{witness}\, w$ from $\mathcal{A}$, if $(x, w) \in R$, store $(x, \pi) \in L$. If (x, w) is valid, then output 1 to P, else return 0.

Fig. A.5 – Non-interactive zero-knowledge functionality $\mathcal{F}_{NIZK}$.

Ledger functionality $\mathcal{F}_{Ledger}$

Functionality $\mathcal{F}_{Ledger}$ stores an initially empty list L of bit string.

- Upon input $(\texttt{append}, tx)$ from a party P, append tx to L. If P is corrupt, then send $(\texttt{append}, tx, P)$ to $\mathcal{A}$. Else return to P.
- Upon input `retrieve` from a party P or $\mathcal{A}$, return L.

Fig. A.6 – Ledger functionality $\mathcal{F}_{Ledger}$.

Secure message transmission functionality $\mathcal{F}_{SMT}$

Functionality $\mathcal{F}_{SMT}$ is for transmitting messages in a private manner.

- Upon input $(\texttt{send}, A, m)$ from a party P:
  . If P and A are honest, then provide a private delayed output $(\texttt{sent}, P, A, m)$ to A.
  . If one of P and A is corrupt, then provide a public delayed output $(\texttt{sent}, P, A, m)$ to $\mathcal{A}$.

Fig. A.7 – Ledger functionality $\mathcal{F}_{SMT}$.

Public key registration functionality $\mathcal{F}_{PKR}$

Functionality $\mathcal{F}_{PKR}$ is for the registration of public keys. It stores an initially empty set T for the party and the registered public key (P, pk).

- Upon input $(\texttt{register}, sid, pk)$ from a party P, check if $sid = (P, sid')$. If not, ignore the request.
  . If (P, pk) is not in L, add (P, pk) to T.
  . Return $(\texttt{registered}, sid, pk)$ to P.
- Upon input $(\texttt{retrieve}, pk')$ from a party P or $\mathcal{A}$, return (P, pk') for $pk = pk'$ in T.
- Upon input $(\texttt{retrieve}, P')$ from a party P or $\mathcal{A}$, return (P, pk') for all $P = P'$ in T.

Fig. A.8 – Public key registration functionality $\mathcal{F}_{\text{PKR}}$.

A2. *Non-interactive zero-knowledge proof*

The functionality $\mathcal{F}_{NIZK}$ adapts the most commonly used non-interactive zero-knowledge functionality that allows zero-knowledge proofs of knowledge.

A3. *Transaction ledger*

The functionality $\mathcal{F}_{Ledger}$ is a simplified global ledger where the transactions are verified, appended to and become available to all parties.

A4. *Secure message transmission*

The functionality $\mathcal{F}_{SMT}$ provides secure message transmission channels between two parties.

A5. *Public key registration*

The functionality $\mathcal{F}_{PKR}$ allows the publication of public keys and their links to a party under the public-key infrastructure.

Appendix B. Proof for Theorem 1

The UC-realization is proven under the assumption that the setup party can only be corrupted after the setup activation was executed once. Let $\mathcal{A}$ be an adversary that interacts with parties running π_{DTAI} in the ($\mathcal{F}_{SIG}$ – $\mathcal{F}_{NIZK}$- $\mathcal{F}_{TPKE}$) -hybrid model. We define a simulator $\mathcal{S}$ to interact with $\mathcal{F}_{DTAI}$ who has an internal copy of $\mathcal{A}$ and involved parties.

Simulating setup. When $\mathcal{S}$ receives $(\texttt{setup}, sid)$ from $\mathcal{F}_{DTAI}$ where $sid = (O^*, sid')$ and O^* is not corrupt, $\mathcal{S}$ simulates $\mathcal{A}$ to process the setup. It first sets $init$ = false. Send $\mathcal{A}$ in the name of $\mathcal{F}_{TPKE}$ the message $(\texttt{setup}, sid, t, n)$ and receive $(\texttt{set}, sid, pk, \{vk\}_{i=1}^{n})$ from $\mathcal{A}$. Query $\mathcal{A}$ for the membership secret key msk. Set $init$ = true. Return $(\texttt{setup}, sid, pp = (t, n, \phi, pk), msk)$ to $\mathcal{F}_{DTAI}$.

$\mathcal{S}$ sets reg = false. When receiving $(\texttt{register}, sid, pid, P)$ from $\mathcal{F}_{DTAI}$, send $\mathcal{A}$ in the name of $\mathcal{F}_{SIG}$ the message $(\texttt{keygen}, sid)$ and receive $(\texttt{verkey}, sid, pvk_{pid})$ from $\mathcal{A}$. Send $\mathcal{A}$ in the name of $\mathcal{F}_{PKR}$ the message $(\texttt{registered}, sid, P, pvk_{pid})$. When $\mathcal{F}_{PKR}$ receives $\texttt{ok}$ from $\mathcal{A}$, mark (P, pid, pvk_{pid}) as recorded. Respond $\mathcal{F}_{DTAI}$ the message $(\texttt{registered}, sid, P, pid, pvk_{pid})$. Set reg = true.

If $\mathcal{A}$ instructs a corrupted P to send $(\texttt{keygen}, sid)$ to $\mathcal{F}_{DTAI}$, then proceed in the natural way. That is, if sid in the two message are different or $sid \neq (P, sid')$ for some sid, then ignore this instruction. Else, send to $\mathcal{A}$ in the name of $\mathcal{F}_{SIG}$ the message $(\texttt{keygen}, sid)$. When $\mathcal{A}$ responds with $(\texttt{verkey}, sid, pvk_{pid})$, send $(\texttt{verkey}, sid, pvk_{pid})$ to P.

Simulating attribute certification: When $\mathcal{F}_{DTAI}$ receives attribute certification request, the simulator proceeds the follows.

If U_i is honest, on seeing output $(\texttt{request}, sid, uid, \alpha)$ for $A_{aid(\alpha)}$ by $\mathcal{F}_{DTAI}$, send $(\texttt{request}, sid, uid, \alpha)$ to $A_{aid(\alpha)}$.

If A_i is honest, on receiving $(\texttt{request}, sid, uid, \alpha)$, simulate the certification process. On receiving $(\texttt{aCert}, sid, uid, \alpha)$ from $\mathcal{F}_{DTAI}$, send $\mathcal{A}$ in the name of $\mathcal{F}_{SIG}$ the message $(\texttt{sign}, sid, (uid, \alpha))$ and forward the response $(\texttt{aCerted}, sid, uid, \alpha, \sigma_{uid,\alpha})$ to $\mathcal{F}_{DTAI}$.

If U_i is corrupt, on receiving $(\texttt{request}, sid, uid, \alpha)$ from U_i, send $(\texttt{request}, sid, uid, \alpha)$ on behalf of U_i to $\mathcal{F}_{DTAI}$.

Simulating transaction: On receiving $(\texttt{sign}, sid, pp', tx, \mathbb{P})$ from $\mathcal{F}_{DTAI}$, send $\mathcal{A}$ in the name of $\mathcal{F}_{SIG}$ the message $(\texttt{sign}, sid, uvk_{uid}, fvk)$ and receive a signature on fvk, namely σ_{fvk}. Send $\mathcal{A}$ in the name of $\mathcal{F}_{TPKE}$ the message $(\texttt{encrypt}, sid, fvk, pk)$ and receive a ciphertext $(\texttt{encrypted}, sid, fvk, pk, c)$. Send $\mathcal{A}$ in the name of $\mathcal{F}_{NIZK}$ the

message $(\texttt{prove}, x, w)$ where $x = (c, pk, \{avk_{aid(\alpha)}\}_{\alpha\in\phi}, \{\alpha_i\}_{i=1}^{\phi})$ and receive a proof $(\texttt{proved}, x, \pi)$. Send $\mathcal{A}$ in the name of $\mathcal{F}_{SIG}$ the message $(\texttt{sign}, sid, (tx, \phi, \pi, c, fvk))$ and receive a signature σ_s. Forward the response $(\texttt{signature}, sid, pp', tx, \mathbb{P}, \sigma_s)$ to $\mathcal{F}_{DTAI}$.

The simulator $\mathcal{S}$ manages an initially empty list to emulate the $\mathcal{F}_{Ledger}$.

On receiving $(\texttt{verify}, sid, pp, tx', \mathbb{P}', \sigma')$ from $\mathcal{F}_{DTAI}$, send $\mathcal{A}$ in the name of $\mathcal{F}_{SIG}$ the message $(\texttt{verify}, sid, fvk, (tx, \phi, \pi, c, fvk), \sigma_s)$ and receive a response r_1. Send $\mathcal{A}$ in the name of $\mathcal{F}_{NIZK}$ the message $(\texttt{verify}, x, \pi)$ and receive a response r_2. Forward the response $r = r_1 \wedge r_2$ to $\mathcal{F}_{DTAI}$.

Simulating threshold tracing: On receiving $(\texttt{tTrace}, pp, sid, tid, tx', \mathbb{P}', \sigma')$ from $\mathcal{F}_{DTAI}$, send $\mathcal{A}$ in the name of $\mathcal{F}_{TPKE}$ the message $(\texttt{decrypt}, sid, fvk, pk, c)$ and forward the decryption share $(\texttt{clue}, sid, pp, tid, tx, \mathbb{P}, \sigma, \nu)$ to F_{DTAI}.

On receiving $(\texttt{reveal}, pp, sid, tx', \mathbb{P}', \sigma', \{\nu_{tid}\}_{i=1}^{q})$ from $\mathcal{F}_{DTAI}$, send $\mathcal{A}$ in the name of $\mathcal{F}_{TPKE}$ the message $(\texttt{decryptC}, sid, c, fvk, pk, \{vk_i\}_{i=1}^{p}, \{\nu_i\}_{i=1}^{q})$ and forward the received prove of user identity $(\texttt{proof}, pp, sid, tx', \mathbb{P}', \sigma', \pi)$ to F_{DTAI}.

It is straightforward to verify that the simulation is perfect. That is, for any (even computationally unbounded) environment $\mathcal{E}$ and $\mathcal{A}$, it holds that $\mathcal{E}$s view of an interaction with $\mathcal{S}$ and $\mathcal{F}_{DTAI}$ is distributed identically to its view of an interaction with parties running protocol π_{DTAI} in the $(F_{SIG} - F_{NIZK} - F_{TPKE})$-hybrid model.

Appendix C. Proof for Lemma 1

Anonymity: Anonymity requires that an adversary can not distinguish two transactions from two users, i.e. the two signature enclosed should be indistinghuishable and the signatures should be unlinkable to its sender. There are three possible ways for an adversary to break anonymity of the construction: 1) an adversary without a legitimate extraction key learns the secret behind a proof π; 2) Knowing the threshold tag-based encryption scheme and the tag, an adversary without enough tracing keys learns the encrypted message, i.e, the user public key uvk; 3) an adversary can successfully forge a signature σ_s signed by the fresh secret key fsk, making a signature distinguishable from others. For the three attacks, we have the zero-knowledge property of $\mathcal{F}_{NIZK}$, the property of selective-tag weak indistinguishability under chosen-ciphertext attack of $\mathcal{F}_{TPKE}$ and the unforgeability property of the signature scheme. Therefore, we prove that the scheme achieves full anonymity, on the threshold condition.

Traceability: The lemma states that AttriChain transactions should be traceable on the base of two independent conditions: the enclosed signature σ_s and zero-knowledge proof π can be correctly verified. To break traceability of the construction, an adversary can be constructed against the conditions, that is, to either break the soundness of $\mathcal{F}_{NIZK}$ or the unforgeability of the tagged digital signature scheme. The possibility for the adversary to succeed is negligible.

REFERENCES

Androulaki, E., Karame,J., De Caro, A., Dubovitskaya, M., Elkhiyaoui, K., Tackmann, B., 2013. Privacy-preserving auditable token payments in a permissioned blockchain system.

Androulaki E, Karame GO, Roeschlin M, Scherer T, Capkun S. Evaluating user privacy in bitcoin. In: International Conference on Financial Cryptography and Data Security. Springer; 2013. p. 34–51.

Arita S, Tsurudome K. Construction of threshold public-key encryptions through tag-based encryptions. In: International Conference on Applied Cryptography and Network Security. Springer; 2009. p. 186–200.

Azouvi S, Al-Bassam M, Meiklejohn S. Who am i? Secure identity registration on distributed ledgers. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology.. Springer; 2017. p. 373–89.

Biryukov A, Khovratovich D, Pustogarov I. Deanonymisation of clients in bitcoin p2p network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM; 2014. p. 15–29.

Bonneau J, Narayanan A, Miller A, Clark J, Kroll JA, Felten EW. Mixcoin: anonymity for bitcoin with accountable mixes. In: International Conference on Financial Cryptography and Data Security. Springer; 2014. p. 486–504.

Bünz B, Agrawal S, Zamani M, Boneh D. Zether: towards privacy in a smart contract world. IACR Cryptol. ePrint Arch. 2019;2019:191.

Corda, White book[online] https://docs.cenm.r3.com/identity-manager.html

Camenisch J, Drijvers M, Tackmann B. Multi-protocol UC and its use for building modular and efficient protocols. IACR Cryptol. ePrint Arch. 2019;2019:65.

Canetti R. Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. IEEE; 2001. p. 136–45.

Canetti R. Universally composable signature, certification, and authentication. In: Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.. IEEE; 2004. p. 219–33.

Canetti R, Krawczyk H. Universally composable key exchange and secure channels. In: Eurocrypt 2002; 2002. p. 337–51.

Conti M, Gangwal A, Ruj S. On the economic significance of ransomware campaigns: a bitcoin transactions perspective. Comput. Secur. 2018;79:162–89.

De Santis A, Persiano G. Zero-knowledge proofs of knowledge without interaction. In: Proceedings, 33rd Annual Symposium on Foundations of Computer Science. IEEE; 1992. p. 427–36.

El Kaafarani A, Ghadafi E, Khader D. Decentralized traceable attribute-based signatures. In: Cryptographers Track at the RSA Conference. Springer; 2014. p. 327–48.

Fauzi P, Meiklejohn S, Mercer R, Orlandi C. Quisquis: a new design for anonymous cryptocurrencies. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer; 2019. p. 649–78.

Feng Q, He D, Zeadally S, Khan MK, Kumar N. A survey on privacy protection in blockchain system. J. Netw. Comput. Appl. 2018.

Fleder, M., Kester, M. S., Pillai, S., 2015. Bitcoin transaction graph analysis. ArXiv Preprint ArXiv:1502.01657.

Ghadafi E. Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability. In: International Conference on Cryptology and Information Security in Latin America. Springer; 2014. p. 327–47.

Goldwasser S, Micali S, Rivest RL. A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 1988;17(2):281–308.
Hyperledger, Fabric white book[online]. https://hyperledger-fabric.readthedocs.io/en/release-1.4/membership/membership.html.
Hardjono, T., Pentland, A., 2019. Verifiable anonymous identities and access control in permissioned blockchains. arXiv preprint arXiv:1903.04584.
Kiltz E. Chosen-ciphertext security from tag-based encryption. In: Theory of Cryptography Conference. Springer; 2006. p. 581–600.
Koshy P, Koshy D, McDaniel P. An analysis of anonymity in bitcoin using p2p network traffic. In: International Conference on Financial Cryptography and Data Security. Springer; 2014. p. 469–85.
Lin C, He D, Huang X, Choo K-KR, Vasilakos AV. Bsein: a blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0. J. Netw. Comput. Appl. 2018;116:42–52.
Monero Projecthttps://getmonero.org/.
Meiklejohn S, Pomarole M, Jordan G, Levchenko K, McCoy D, Voelker GM, Savage S. A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 Conference on Internet Measurement Conference. ACM; 2013. p. 127–40.
Miers I, Garman C, Green M, Rubin AD. Zerocoin: anonymous distributed e-cash from bitcoin. In: 2013 IEEE Symposium on Security and Privacy. IEEE; 2013. p. 397–411.
Nakamoto S. In: Consulted. Bitcoin: A Peer-to-Peer Electronic Cash System; 2008.
Noether S, Mackenzie A, et al. Ring confidential transactions. Ledger 2016;1:1–18.
Quorum Whitepaper[online]. https://github.com/jpmorganchase/quorum/blob/master/docs/QuorumWhitepaperv0.2.pdf.
Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 1978;21(2):120–6.
Ruffing T, Moreno-Sanchez P, Kate A. Coinshuffle: Practical decentralized coin mixing for bitcoin. In: European Symposium on Research in Computer Security. Springer; 2014. p. 345–64.
van Saberhagen., N., Cryptonote v 2.0. [online].
Sasson EB, Chiesa A, Garman C, Green M, Miers I, Tromer E, Virza M. Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. IEEE; 2014. p. 459–74.
Shao W, Li H, Chen M, Jia C, Liu C, Wang Z. Identifying bitcoin users using deep neural network. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer; 2018. p. 178–92.
Sun Y, Zhang R, Wang X, Gao K, Liu L. A decentralizing attribute-based signature for healthcare blockchain. In: 2018 27th International Conference on Computer Communication and Networks (ICCCN). IEEE; 2018. p. 1–9.
Zhu Y, Qin Y, Zhou Z, Song X, Liu G, Chu WC-C. Digital asset management with distributed permission over blockchain and attribute-based access control. In: 2018 IEEE International Conference on Services Computing (SCC). IEEE; 2018. p. 193–200.

Wei Shao Born in 1994, received the B.S. degree in computer science in 2012–2015 from South China University of Technology and the M.Sc. degree in information security in 2016 from University College London, UK. She is now a Ph.D. student in the department of cyber sciences at Nankai University. Her research interests include information security, cryptography and blockchain security.

Chunfu Jia A Ph.D. supervisor, a professor and the Head of Department in the Department of cyber Sciences, Nankai University. His main research interests include network and system security, cryptography application and malware analysis He has lead or participated in the National Natural Science Foundation of China, 9 provincial scientific research projects.

Yunkai Xu An undergraduate student at the college of cyber Sciences, Nankai University. He is good at programming with multiple languages including Python etc.

Kefan Qiu A graduate student at the college of cyber Sciences, Nankai University. He is good at programming with multiple languages including Solidity etc. His major research interests include malware and applied cryptography.

Yan Gao An undergraduate student at the school of cyber security and computer, Hebei University. He is good at programming with multiple languages including Solidity, Go etc.

Yituo He An undergraduate student at the Bell Honors School, Nanjing University of Posts and Telecommunications. He is good at cryptography and number theory.