Pistis: Privacy Preserving Compliant Transaction System

Antoine Rondelet and Michal Zajac

Clearmatics, UK
ar@clearmatics.com, michal.zajac@clearmatics.com

Abstract. Recent surge in adoption of cryptoassets, by both retail and institutional investors, has lead to increased scrutiny from law makers and regulators. This increasing development and use of privacy-preserving technologies on blockchain systems raises a "privacy vs compliance" dilemma between designing systems providing strong privacy guarantees – necessary to conduct any type of non-trivial business activity and fundamental to preserve individuals' financial freedom – and the need for transparency to ensure regulatory compliance.
In this paper we introduce *Privacy Preserving Compliant Transaction System* (PPCTS) which assure that blockchain-traded assets can be traded privately, yet, in compliance with their regulatory frameworks. To that end we identify key actors that may take part in regulating such assets, list the key properties of a PPCTS, define a UC-model ideal functionality of such systems, and propose protocols that implement it.

Keywords: zero-knowledge proofs, regulatory compliance, digital cash, Ethereum, Zeth, privacy

Disclaimer: The content of this work does not constitute legal advise.

1 Introduction

The recent global developments in payment systems and financial technologies coupled with the emergence and increased adoption of blockchain and decentralised protocols present a wide set of opportunities as well as numerous challenges.

Market structures, trading technologies and financial instruments have steadily evolved over the centuries, forcing regulators across jurisdictions to continuously adapt existing frameworks to maintain high confidence in the financial system, protect its stability as well as its consumers.

[**Michal 4.10:** Refer to that later – regulators need to keep financial system stable and protect its customers, hence "regulatability" is a must and our solution allows it.]

In the United Kingdom, for instance, the introduction of the Banking Act (1979) [Lee79] – in response to economic turbulence (1973–1975) due to loosely controlled credit allocation [Eff94] – was quickly followed by subsequent regulations, some of which aiming to detect and prevent procedures by which money acquired from criminal activity could be dissimulated and appear as being lawfully acquired (i.e. so-called "money laundering" [Para,20001,Parb][1]). More recently, and in light of the weaknesses revealed by the 2008–2009 financial crisis, the UK regulatory framework has been reshaped, with, notably, the dissolution of the Financial Services Authority (FSA) in 2013[2] and the creation of the Financial Conduct Authority (FCA), Financial Policy Committee (FPC) and the Prudential Regulation Authority (PRA). As of today, and as illustrated in [Aut12, Chapter 1], the Bank of England (BoE) administers monetary policy, influencing interest rates, inflation and employment, and also plays a key role – via its subsidiary, the PRA and the FPC – in the regulation of financial markets, along with the FCA – supervised by HM Treasury.

While sound local financial regulations are key to keep domestic markets robust, stable and trusted, criminal activities are frequently carried out on a large scale. Local actions and systems to *detect and report* suspicious activity to law enforcement – via Suspicious Activity Reports (SARs)[3] – are often criticised as being very costly and offering few results [Edm18]. [**Michal 4.10:** Another thing to refer – in our system policy assigners define what is allowed and what's not. This allows for some degree of automatization of fraud / criminal activity prevention – we allow application to have much stricter rules than they can be defined just for a casual wire transfer. More precisely, we allow users to remain anonymous, transfer funds quickly and privately, but on the other hand, policy assigners assign compliance policies that are at least as strict as compliance policies for classical non-anonymized fund transfer options.]

Moreover, the increasing development of fast cross-border communication technologies and the patchwork of heterogeneous jurisdictions significantly increase the difficulty to bring activities related to money laundering and financing terrorism down. In an attempt to mitigate such issues, an intergovernmental organization, the Financial Action Task Force (FATF), was created in 1989. This organization has developed recommendations and standards in order to foster coordinated global response to prevent organised crime, corruption and terrorism [For]. [**Michal 4.10:** General comment

[1] See [Edm18] for an overview of the evolution of AML regulations in the UK.
[2] Initially founded in 2001
[3] SARs are processed by the National Crime Agency (NCA) in the UK[Age,Age19] and by Financial Crimes Enforcement Network (FinCEN) in the US

– when we provide some background on how financial markets are regulated / oversight, we should also state how our functionalities / PPCTS follows that regulatory approach. That is, we should not only give the reader a general background, but a background that is directly relevant to our solution and show how it is relevant.

Nevertheless, despite the establishment of such organizations, along with the signature of various "Memorandums of Understanding" (MoUs) e.g. [SCA06,CA09] (see also e.g. [Aut19,Com19]) – to ease cross-jurisdiction investigations – gathering evidence for investigating illegal cross-border conduct remains a major challenge, particularly when blocking or secrecy laws are involved, which may be unwelcomed by governmental authorities in other countries [Dot92].

1.1 Regulating cryptoassets

Recent surge in adoption of cryptoassets (by both retail and institutional investors) has led to increased scrutiny from law makers and regulators. Cryptocurrency projects like Diem–formerly known as Libra–present "unprecedented taxation challenges" [CT20], and the increasing development and use of privacy-preserving technologies on blockchain systems (e.g. [RZ19,BAZB19,Wil19,HBHW21]) raises a "privacy vs compliance" dilemma between designing systems providing strong privacy guarantees (necessary to conduct any type of non-trivial business activity and fundamental to preserve individuals' financial freedom which is at the core of democracy), and the need for transparency to ensure regulatory compliance.

Despite the inherent transparency of various blockchains (which is leveraged by several companies and projects e.g. [Ell,Cha,Ree] to understand payment flows within the systems) reports suggest that such systems have "offered exceptional scalability to ransomware operations" [tai20] and are used for illicit activities online [ana20]. Interestingly, empirical evidence suggest that privacy-enabling cryptocurrencies are only used in minority to carry out fraudulent activity on the darknet [SFS+20]. However, it is important to bear in mind that this landscape is evolving astoundingly fast and what holds true today may not remain true tomorrow. In an attempt to conciliate the use of privacy enabling cryptocurrencies with regulatory compliance, some suggested to implement risk-based Anti-Money Laundering (AML), Countering Financing of Terrorism (CFT), and Know-Your-Customers (KYC) processes on Virtual Asset Services Providers (VASPs) [SBL20,For19]. These approaches, articulated around the thesis that VASPs (e.g. cryptocurrency exchanges) act as necessary gateways criminals need to go through to "wash and convert" unlawfully acquired cryptocurrencies to fiat monies (like GBP, EUR or USD) may not hold true indefinitely.

The dramatic expansion of the cryptocurrencies user base[4] leading to broader acceptance of cryptocurrencies as means of payment (see also [Keh20,New21]) represent an impediment for regulators who try to regulate the ever-mutating blockchain industry without altering financial innovation[5]. Even if having "legal tender" in almost no jurisdictions (except El Salvador, at the time of writing [New21]), cryptoassets with large user bases can lead to a large number of (truly) peer-to-peer transfers which associated funds may never be redeemed and converted to fiat money, hence failing to flow through the checks implemented by centralized services such as exchanges. Furthermore, various KYC checks could also be bypassed by carrying out off-chain transactions of cryptoassets via the exchange of "hardware sticks" bearer instruments (e.g. Zeth Bearer Instruments [Ron20b]) or, by using blockchain transaction relays [RT20] as a way to escape on-chain surveillance tools for instance. Finally, the patchwork of heterogeneous cryptocurrency regulations around the world can still be leveraged to escape strict regulations by turning to VASPs located in advantageous jurisdictions, or by designing instruments that may not fall strictly under a regulator's jurisdiction[6].

1.2 Parties involved in regulatory compliance on blockchain systems

Before discussing solutions and designs for the compliance verification, it is crucial to answer the following question: *Who are the parties involved in compliance verification process?* We list below the set of actors that may be obliged to participate in such process. Importantly, the nature of the transacting parties – highly regulated entities like banks, retail traders etc. – may require different compliance checks involving potentially different regulators (see Section 1).

[4] As of Q3 2020, [BPW+20] estimated a total of up to 101 million unique users across 191 million accounts opened at service providers, while two years prior, a similar study estimated the number of identity-verified cryptoasset users at about 35 million globally.

[5] E.g. bills like the Stable Act [TGL20] which intend to regulate stablecoin issuers (bank charters and reserves requirements) are seen as "innovation killers" by opponents.

[6] In the US for instance, the Security and Exchange Commision's (SEC) jurisdiction is over securities, while the Commodity Futures Trading Commission (CFTC) has jurisdiction over derivatives such as futures and swaps (further to the Dodd-Frank Act [Com]). Such separation of concern between the SEC and the CFTC poses a challenge with regard to regulating cryptoassets as it may be unclear which assets (e.g. stablecoins, synthetic assets, blockchain-based derivatives, "utility tokens", NFTs etc.) should be treated as securities or not (see "Howey Test" [SC])

1.2.1 Regulatory bodies.

The role of such regulatory bodies is to act as standard-setters [Dot92], in order to define the set of recommendations and rules that financial actors need to follow in a given (set of) jurisdiction(s). This established set of standards is then used as a basis for further activities, ranging from *supervision* to *enforcement* to *authorisation* to only name a few.[7]

1.2.2 (Financial) Services Providers.

Centralized VASPs. Despite a lack of consensus around cryptocurrency regulations[8], cryptocurrency exchanges allowing consumers to buy and sell cryptoassets tend to implement standard processes that consist in identifying and verifying users, maintaining records, and complying with AML/CFT reporting obligations. Such obligations can vary from one jurisdiction to another and may include, among others, suspicious transaction or activity reporting, enhanced procedures for high transaction amounts etc. Moreover, customized checks may additionally be carried out depending on the risk profile of the cryptoassets traded, as well as the VASPs' jurisdictions. Trading privacy preserving cryptocurrencies, for instance, may require additional customer checks.

Smart contract (DApp) providers. Implementers of decentralized applications (DApps) are providing state transitions allowing network users to transact on a blockchain system (such state transitions may be: "stablecoins functionalities", "decentralized exchanges" etc.). Such applications are hosted on each of the nodes of the decentalized blockchain network, and are thus not executed within a well-defined jurisdiction. Instead, the associated code is hosted and executed on all blockchain nodes worldwide, which poses a challenge for regulation. Nevertheless, such applications *may* be provided by organizations registered within a specific jurisdiction, providing an avenue for local regulators. It is important to note, however, that anonymous groups of developers may still deploy applications to public blockchain networks.

In the next sections, we will argue that DApps deployed or operated by registered organizations may implement privacy-preserving compliance checks at the protocol level; providing a robust complement to regulatory compliance checks implemented on centralized venues (e.g. centralized exchanges). Such compliance checks may be added to the provided state transition on the blockchain, allowing to automate part of the process, preventing fraudulent state transitions and lessening burdensome reconciliations[9].

Blockchain developers and operators. Establishing the regulatory regime of blockchain systems is a tedious task, especially due to the potential lack of central coordinating entity. As proposed in a draft document by the FATF [FAT21a] (further amended [FAT21b]), governance token holders or developers of blockchain systems may be considered as VASPs, and hence subject to their regulatory regimes. While it is not possible to foresee tomorrow's regulations, it seems that if the early FATF recommendations were to be followed, there could be a convergence between the so-called *permissionless* and *permissioned* blockchains. In other words, if there exist a regulated VASP behind the development of every blockchain systems, chances are that KYC and Customer Due Diligence (CDD) processes will be implemented in order to onboard users on the systems, for example, it may be necessary to modify the Ethereum account address creation routine to ensure that new accounts have been created w.r.t. KYC and CDD.

Permissionless blockchain protocols, allowing ad-hoc on/off-boarding on the system, have flourished in the last years and observed steep increase in retail and institutional adoption [Yor,Gou]. It is unclear whether the current situation – relying on centralized VASPs to act as safeguards or gateways to prevent fraudulent conversions of funds between fiat and cryptocurrencies – will persist or mutate. In the rest of this document, we focus exclusively on permissionless blockchains. We further consider blockchains with an expressive execution environment allowing to deploy and execute smart contracts (e.g. EVM chains[10]).

Blockchain services providers. While blockchain protocols act as the backbone on which DApp services are deployed, it is also important to note that a myriad of businesses have been created beyond the strict boundaries of blockchains, forming entire "off-chain" ecosystems. Such services may either be:

custodial, like wallet providers, securing the cryptographic keys of their users; or
non-custodial, like transaction relays, which receive messages from their users and relay them to the blockchain, cf. [RT20]; or blockchain bridges, which connect multiple blockchain systems, e.g. [btc,rai]

It seems fair to assume that custodial services providers must be subject to stricter regulations, since they control the users' funds. On the other hand, loose regulations around non-custodial services may constitute an open door for fraudulent activity, e.g. unregulated flows of liquidity from one blockchain to another etc. For simplicity, we do not consider the regulatory regime for these service providers in the rest of this document.

[7] See e.g. `https://www.fca.org.uk/publications/corporate-documents/our-approach`
[8] See e.g. `https://complyadvantage.com/blog/cryptocurrency-regulations-around-world/` for more details
[9] Since the state of a blockchain is immutable, it is useful to prevent fraudulent state transitions from being executed and added to the chain in order to keep a compliant state at any point in time, reducing reconciliation procedures and associated costs.
[10] i.e. blockchains using the Ethereum Virtual Machine. See [eth] for more details.

Remark 1. We note that, CDD checks are generally carried out by verifying legal documents owned by the service customer (identity check using passports, proof of fund provenance using payslips or bank statements etc.) *which are ill-suited for information disclosure minimization*. For example, the broker probably does not need to know the city of birth of a customer, nor their precise age. Instead it may simply know that the customer is legally authorized to trade. On the other hand, cryptographic tools make it possible to carry out such compliance checks with minimal information disclosure.

1.2.3 Transacting parties.

Beyond financial services providers and regulatory bodies, we now consider two categories of users: those using existing services provided by regulated third parties (e.g. DApps, exchanges etc), and those who do not. While the former user class – *blockchain services users* – is subject to AML/CFT/KYC processes implemented by the service of interest (and must then provide multiple pieces of information), the latter user class – *native blockchain users* – may simply join the permissionless network on an ad-hoc basis, and are thus exempt of such KYC checks. In fact, native blockchain users can either simply transact native assets (e.g. ETH) on the platform without using any DApp services, and/or may leverage the smart-contracts capabilities of the underlying blockchain to seal self-enforcing legally binding agreements with other users as a substitute of legal (paper) contracts. (In such case, the cryptographic signature attached to the blockchain transaction executing the smart-contract acts as the digital equivalent as an "ink signature" on a (paper) contract.)

In the rest of the document, we will treat all smart contracts as DApps, and focus on the relations between service providers, their users and the regulatory bodies.

1.3 State of the art

[**Michal 27.08:** For now we have here only parts moved from other sections. Need to expand]

[**Michal 6.09:** Give an introduction for this section]

Below we discuss some prior approaches to compliance and privacy-preserving transacting systems.

View keys In contexts where an operator must be granted the right to monitor all activity happening on a given service, it is possible to use so-called *view keys*. Regardless of the derivation method employed to generate the keys, this approach aims to keep users' data (e.g. transaction details) private to the rest of the user set, while allowing full disclosure of user's activity to a service operator. Such approach obviously raises issues related to users' privacy and *key custody* – the service operator must ensure that the key material is adequately protected to prevent unintended use of the view keys. This often necessitates to extend the service's infrastructure with Hardware Security Modules (HSMs), employ multi-party computation techniques (e.g. "key splitting"), and set strict access control policies (e.g. to prevent "insider attacks").

On Damgård et al [DGK$^+$20]. Our work relies on Damgård et al [DGK$^+$20] who defines UC-functionality for privacy preserving transacting system. The system allows users to create multiple (yet limited number of) accounts, such that it is infeasible for an external spectator to tell which user posses which accounts. On the other hand, if the user behaves dishonestly then a qualified set of so-called anonymity revokers can reveal its credentials and remove it and all its accounts from the system.

In this paper we rely on this solution. However, as explained in **??**, we extend it considerably. One of the most important improvement being including compliance checks into the system.

1.4 Our contribution

As initially alluded to in [Ron20c], compliance predicates can be used along with modern zero-knowledge proof system in order to conciliate the use of privacy-preserving protocols with regulatory compliance. In fact, arguing about predicate satisfaction in zero-knowledge can lead to efficient design for auditable digital-cash/private transfers on blockchain systems.

In this paper, we propose several considerations and tools for the design of privacy-preserving blockchain-based compliance protocols[11]. We start by considering the various settings by which compliance can be enforced on blockchain systems. Then, we describe a protocol that automates and facilitates privacy-preserving compliance checks on blockchains like Ethereum [Woo19]. We base our solution directly on the work of Dåmgard et al.[DGK$^+$20], however we provide it with the following improvements:

[11] The work [BCG$^+$21], published while writing this manuscript, further proposes to use zero-knowledge proofs as a way to solve so-called "Verification Dilemmas" in multiple legal contexts. What we describe here as a privacy/compliance paradox can be phrased as a Verification Dilemma.

Private policies. Our most important contribution is to allow for private compliance policies both on the accounts as well as on the transactions. More precisely, in [DGK+20] each user has a private list of attributes AL that is signed by an identity provider (to assure that users cannot set these attributes as they like), and each of the user's accounts has a specified publicly-known policy $\mathbf{P}_{comp}$, such that $\mathbf{P}_{comp}(\mathsf{AL}) = \mathsf{true}$, i.e. the policy is fulfilled by the user's attributes. Unfortunately, we note that despite the use of zero-knowledge proof systems to prove that some private data satisfies some properties/constraints, there is always a tradeoff between the information leaked by the system, and the statement that is being proven. For example, consider a trading platform that should only be available to EU citizens (i.e. $\mathbf{P}_{comp}$ here is of the form "is EU citizen?" and returns true or false depending on the input data). The platform operator may, for instance, hold a set of public keys from identity issuers from all 27 countries, and expect users to provide a message signed by one of the authorized identity issuer to create an account. Nevertheless, such an approach leaks (one of) the nationality(ies) of the user via the signature of the identity issuer. So, even if the message signed perfectly blinds the user's sensitive credential data, the protocol uses very restrictive predicates that leak valuable information about the user to the platform (and more importantly, to potential network adversaries!). Such leakages are particularly important since they may accrue as the system is used for prolonged period, and may be correlated with other data sources (e.g. other leakages on the system) to carry out de-anonymization attacks. The first step towards finding a solution to this problem is to acknowledge that leakages still occur even when using cryptographic tools such as zero-knowledge proof systems. Even if $\mathbf{P}_{comp}(\mathsf{AL}) = \mathsf{true}$ is proven in zero-knowledge, the "verification bit" is disclosed to the verifier (i.e. disclosed to the public if a scheme is publicly verifiable). As such, the design of the statements to prove is of particular importance. To stop leaking the citizenship in our example, it may be more appropriate for the trading platform operator to leverage ring signatures with the set of identity issuers. That way the signature on the user message only proves that one EU member has issued valid credentials to this user, without disclosing the particular issuer. That is, system designers must design sufficiently sophisticated predicates to safely provide access to their services to users while empowering them with privacy and minimal information disclosure[12]. This is all the more important in the context of blockchain-based services (e.g. DApps) as user messages (i.e. transactions) are routed on a broadcast network before being included to a public ledger. Poor policy design from the DApp providers exposes their end-users to the whole network.

In this paper we propose a mechanism allowing users to *hide* the compliance policy they prove, effectively minimizing information leakages on the system and better safeguarding users' sensitive personal and transactional data.

Transaction checks. Contrary to [DGK+20], which focuses exclusively on an identity layer, we allow, in our ideal functionality (and protocol), users to show predicate satisfaction on their transaction data. This makes it possible to prove compliance on privacy-preserving state transitions such as ZETH, which brings Zerocash-type [BCG+14] privacy to Ethereum[13].

On the gas model. The gas model introduced in Ethereum [Woo19], and relevant to most smart contract enabled blockchains, is a security mechanism that protects the network against adversaries trying to execute non-halting programs. The mechanism is quite simple and effective as it prices every computational (and memory manipulation) step of a smart contract, effectively requiring the network users to pay for executing state transitions on the system. While it is easy to see how such an approach solves the attack vector abovementioned (i.e. attackers trying to run non-halting programs on the network would see their account balance emptied and the computation would stop afterwards), this mechanism comes with major drawbacks. One such issue is that the gas model constitutes an impediment to user privacy on smart-contract chains. Since executing a smart-contract requires to hold funds to be able to pay for each instruction executed, end-users must call a smart contract from a funded account (either funded via p2p transactions with other peers or via a VASP like an centralized exchange). Inevitably, this means that at least one other party on the network (VASP or another peer) knows additional information about the holder of the account and can use such information to carry out de-anonymization attacks on the network. To mitigate the tension between the need for gas and the need for privacy, we extend our model to introduce a collection of parties called *relays* (denoted R) which append information on the ledger on behalf of a user [RT20].

Remark 2. We note that the need for an incentive structure assumed by Damgard et al. [DGK+20, Section 3.4] to make sure that no account holder can create more accounts than they are allowed, is not necessary if we design a PPCTS for a specific DApp on a smart contract chain. In this setting, no party may be required to report duplicated accounts as the account creation logic may partially be delegated to a smart contract which would simply automatically reject the addition of an existing account entry by malicious account holders.

Our proposed modifications on top of Dåmgard et al.'s *identity-layer* allow us to design a PPCTS, as defined in Section 2.

[12] A general good practice is to design a predicate (to prove in zero-knowledge) as a decision tree that branches with each OR clause in the policy (i.e. see a policy as a logical disjunction). This allows users to use their private data to evaluate the appropriate branch to true, redeeming the whole policy valid, without disclosing the evaluation path in the tree.

[13] As noted in [RZ19] the difference between account-based and UTXO-based blockchains is of particular importance with regard to information leakages. The privacy guarantees in ZETH are weaker than these in Zerocash due to the need to pay for gas. More on that later.

2 PPCTS

[**Michal 26.08:** Should be made a subsection of an introduction?]

Below, and when clear from context, we informally use the term *verifier* to denote the actor that asks pieces of information (e.g. a service) to a *prover* (e.g. a user or customer) and decides to either deem the received pieces of information as admissible (or correct) or not.

2.1 Desired properties

We list below the desired properties of a PPCTS:

Completeness. An honest compliance verifier should always accept a correct proof of compliance from an honest and compliant prover.

Soundness. It should be infeasible for a non-compliant prover to convince the verifier to accept non-complying pieces of information as adequate evidence of compliance, e.g. an invalid transaction.

Efficiency. The compliance process must be efficient in order to preserve the overall system's performances, i.e. proving compliance and verifying it must be efficient.

Reliability. If the system cannot verify the compliance it should fail safe. That is, when a proof of compliance cannot be obtained, the system should behave as if the compliance was not satisfied.

Robustness. The regulatory compliance checks must not *only* rely on trusted third parties and must not be easy to bypass. This property is particularly important on distributed ledgers since KYC checks implemented at specific gateways (e.g. "exchanges/brokers") may easily be bypassed by using decentralized alternatives (decentralized exchanges, buying assets on a purely peer-to-peer basis etc.)

Security. Implementing compliance processes must not undermine the overall security of the system and of its users. For instance, the PPCTS should not represent an attack vector on the system – carrying out compliance checks must not put users' funds at risk – and must be designed around the same (or weaker) security assumptions as the system on which it is built. Past events have demonstrated how (commercial) customer databases and KYC checks carried out by hardware manufacturers and/or service providers may translate into personal information leakages (e.g. as consequences of cyber attacks) about the user base, which may undermine users' personal security/safety, e.g. [Rap20,Zmu20], affecting the breached service's reputation at the same time. [**Michal 18.08:** How it is related to privacy?] **Antoine 25.08:** It is not directly related. Sure a privacy preserving system will have less data to expose in the event of a breach on the service, but here the focus is on the security boundary of the system, which is to make sure that a flaw in the PPCTS cannot be used to break the whole system.

Adaptability. PPCTS should be easy to extend and composable with other systems. In light of the quickly changing (legal and technological) environment due to the fast-paced innovation in the blockchain ecosystem these properties are especially imporant. In fact, regulators may adjust their guidance, or service operators may decide to update their terms and services to collect more data about their users (e.g. transaction-related or personal pieces of information), it should be possible for the parties involved in compliance verification to provide and verify new pieces of information by relying on the infrastructure and techniques already implemented in the PPCTS.

Privacy preservation. Compliance processes should not reveal any additional information except the fact that the proven compliance policy holds.

[**Michal 26.08:** New structure here – statement + description. Used "paragraph" envirionment, but not sure about it (p-graphs allow to easily distinc statement from description)] [**Michal 26.08:** Important and missing – showing that our PPCTS follow these rules] [**Michal 26.08:** Important and missing II – defining the above rules for our PPCTS.]

2.2 Interactions between actors

In light of the diversity in the set of actors, the set of messages exchanged between a service user and a service provider may greatly vary depending on the nature of the service, and depending on the user. While some of the pieces of information related to regulatory compliance may be exchanged directly on the blockchain (e.g. cryptographic commitments, ciphertexts, etc.), other pieces of information need to be exchanged via off-chain interactions. For instance, a regulatory body may publish a set of recommendations and guidelines, that service providers need to follow, on its website (off-chain). A user may want to gain access to a blockchain-hosted service, and thus may exchange pieces of information with the service operator (off-chain) to satisfy the KYC procedures of the service. In response to the user's registration, a service may modify the state the blockchain to grant the user access to the service, and/or may simply carry out off-chain modifications to the service's state and send a confirmation message (off-chain) to the user. While using the service, a user may post data on the blockchain, which may be used by the service operator as input to its monitoring processes.

It is important to remember, however, that blockchains are not "mere distributed databases". In fact, a blockchain acts as "source of truth" to which adding data is generally an expensive operation. Once added to the immutable blockchain state, the cost of keeping a piece of data under consensus (as part of the blockchain state) is paid by the whole network (see, e.g. [Ron21]). As such, not all pieces of information are fit for inclusion on a blockchain system. For instance, adding Personally Identifiable Information (PII) to immutable ledgers is not desired as it violates data regulations in certain jurisdictions (e.g. GDPR). More broadly, appending pieces of information to a blockchain requires to send a message to a public broadcast channel, which constitutes challenges with regard to information leakages minimization in the context of privacy preserving protocols (e.g. as we saw, the mere fact of sending a zk-proof of a publicly verifiable NIZK on a blockchain leaks the "verification bit" which provides information about the instance/witness pair of the user).

3 Protocol overview

3.1 The actors

User U**.** A user is a party using the system. That is, a party that wants to have their transactions (i.e. blockchain virtual machine state transtions) tx to be processed by the system. To that end the user first obtains their application identity from the identity module and (possibly anonymously) registers a number of accounts which are then used to transact. Transactions provided by the user are verified for their their validity and compliance. The latter may require from the user additional data, like a proof that transactions follows the policy assigned by the policy assigner, or interaction with the policy assigner themself. Furthermore, if the user misbehaves they need to take into account possible consequences like their identity reveal and accounts block. This is possible since information required to reveal user's personal data or block their accounts are given to a set of anonymity revokers, who working collectively may disclose them.

Relay R**.** A relay is a party who provides other users with anonymity services. More precisely, a relay acts as a proxy to the blockchain. It receives fees from users in exchange for submitting transactions to the ledger on their behalf. By submitting transactions from its ledger account, the relay pays for the gas associated to the transactions and allows the end-users to stay anonymous (i.e. transact without a funded account on the ledger). Such relayed transactions may, for instance, be used by users to fund a newly created ledger account. Users may stay anonymous to the relay by making sure that no information in the message sent to the relays (e.g. transactions payloads) can be used to de-anonymize them. To that end a number of privacy-preserving techniques may be employed, like encryption, cryptographic commitments, zero-knowledge proofs, and communications via anonymous communication networks. The incentivization structure of relays is outside of the scope of this document. For now, we assume that the relays can receive fees via off-chain or on-chain mechanisms. For discussions on relay mechanisms and incentivization, see e.g. [RT20].

Policy assigner PA**.** The policy assigner is a party which provides the users of the system with compliance policies that have to be followed by transactions the users send. This makes the application owner – VASP – a natural candidate to fulfill that role as well.

Anonymity revoker AR**.** Anonymity revokers make a set of parties which role is to keep encrypted users' credentials and their accounts' details. When a user behaves dishonestly, what has to be agreed by at least some threshold of the anonymity revokers, then these credential and account details are revealed to prevent the misbehaving party from a further use of the system. Alternatively, an honest user should not worry having their credential revealed in case of a dishonest anonymity revoker, as any set that counts less than a threshold anonymity revokers can tell nothing about the stored user's data.

Identity provider I**.** Identity providers are parties who check real-life credential of users and provides them with anonymized credential ready to be used in the system.

Application owner O**.** In the real world, the application owner is a party that sets-up the transacting smart contract, picks which identity providers and policy assigners should be available in the contract. The application owner is a gate-keeper who prevents users, identity providers and policy assigners with bad reputation to join the system. Here we also assume that the application owner sets up the parameters for cryptographic primitives used in the system, like encryption scheme, signature scheme, non-interactive zero-knowledge proofs, where it picks the refence strings as well. (Alternatively, one could set up a collective that jointly sets up the parameters and reference string via an MPC.)

3.2 The protocol briefly explained

From a high level, a user proceeds as follows (see **??** for more information). First, a user U retrieves anonymity revokers' public keys and use them to threshold-encrypt its PRF key k that will be later used to create user's accounts.

Getting new identity. In the next step, the user interacts with a chosen identity provider I to get an identity that will be used in the transacting system. To that end, it provides the identity provider with an attribute list AL which contains all data required to get identity on the system. The identity provider checks AL by either non-cryptographic means (e.g. relying on user's national ID or, in case of institutions, its company register's entry), or using cryptographic credential scheme with credential signed by a trusted entity. Importantly, we require the identity provider to be able to identify the user so legal actions could be imposed on it if it misuses the system. The user also shows I a public key of its choice pcred and shows knowledge of the corresponding secret key scred. If all the checks are accepted, the identity provider signs scred, user's key k that will be used to create new system accounts and attribute list AL. Since we want these values (possibly except of AL) to remain private, the signature is blind. That is, the identity provider does not know what it signs, however, since we also require that the user provides I with a zero-knowledge proof that it picked data to be signed honestly, the identity provider may securely do that without doubting on the validity of the data signed. We denote the signature by σ_{id}.

Getting compliance policy. Before the user starts to transact it also needs a compliance policy. Transactions the user is going to perform will need to fulfill that policy. To that end, the user turns to a policy assigner PA and defines how it wants to use the transacting system. Given that information PA picks a policy for U. This step can be realized in multiple ways – the policy may be assigned based on interaction or there may be a public information what policies are assigned to which set of actions and which type of users. To confirm that the user is allowed to perform the required set of actions the policy assigner picks a predicate $\mathbf{P}_{al}$ which has to be fulfilled by the user's attribute list AL. That is, we require $\mathbf{P}_{al}(\mathsf{AL})$ to hold. To assure that AL remains private a zero-knowledge proof is used to show that predicate. Importantly, PA also checks that AL is the same list of attributes that was signed by the identity provider. Eventually, the policy assigner picks a compliance policy $\mathbf{P}_{comp}$ and sends it to U along with its signature σ_{comp} which certifies authenticity of the policy, i.e. assures that no malicious user can pick its own policy $\mathbf{P}'_{comp}$ and use it instead of the assigned $\mathbf{P}_{comp}$.

If the user wants to use the system in multiple ways ... [**Michal 28.10:** We started a discussion on that. We considered two options here. Either the user gets multiple identities and single policy per identity or we allow only one identity per user and multiple policies. AFAIR, we were leaning towards the latter option. However, now I think that maybe the first option is slightly better. It allows to revoke more user's accounts if it misbehaves. That is, by default we revoke all accounts per identity. If the user has multiple identities then we cannot revoke all its accounts. Obviously (?) the user can set up multiple identities with multiple identity providers. (Maybe there is a way to prevent that, like each of the identity provider could hold a register of credentials hashes h of registered users. Then we could use MPC or PIR (private information retrieval) methods to allow identity providers to privately query other IPs' databases to check whether a concrete hash h was registered there). Another argument for the multiple policies per single identity is more philosophical one. The user wants to get multiple policies, so we should allow that by changing PA specification. Getting multiple policies by getting multiple identities seems like unnecessarily complication. OTOH, if the user wants to perform different activity then a different attribute list may be required – these things are verified by the identity provider. In that case having multiple identities per user makes sense.]

Creating new accounts. Given identity and policy, the user can create accounts that will be used to send and receive transactions. U starts by threshold-encrypting its public credentials pcred and computes the account identifier by running a PRF with a key k on index x. Index x numerates the accounts created by the user, it takes values from 1 to $\mathsf{maxAccounts}$, where the latter is the upperbound for a number of accounts the user can make. Then the user computes a policy digest dig_{comp} which assures that the compliance policy tied with the account is the same as the compliance policy the user got from the policy assigner without revealing the policy itself. Eventually, the user computes zero-knowledge proof to certify that (1) the above computation has been done correctly; (2) it knows a valid scred corresponding to pcred; (3) has a valid signature from the identity provider that assures validity of attribute list AL; (4) it has a valid compliance policy assigned by a policy assigner.

Importantly, the zero-knowledge proof the user is showing assures that it in fact got the compliance policy from the policy assigner and the identity from the identity provider. That is, it excludes the attack where a malicious user U^* obtains an identity from the identity provider, but then gets a compliance policy from another user U (or vice versa) and for such policy creates a new account. Such attack to be successful requires U^* to know U's secret key scred and the signature it got from its identity provider. (If U^* knew U's signature from the identity provider then it could impersonate U.)

Sending transactions. Eventually, the user is allowed to send transactions. To that end, U computes zero-knowledge proofs which show validity of the transaction (regarding the rules of the transacting system, e.g. assuring that funds used in the transaction were not created out-of-thin-air etc.) and transaction compliance with the assigned compliance policy.

Proposed system allows users to send non-compliant transactions if they have been approved by the corresponding policy assigner. Also, to protect privacy of the user, transactions can be send via a Pistis relay who is just another system user. In general, relays can be used whenever user is tasked to publish some information on blockchain.

Revoking user's accounts. If misbehaviour of a user U is detected, its identity can be revealed and accounts blocked. Threshold encryption allows large-enough set of anonymity revokers to decrypt encryptions of user credentials pcred and key k. The former allows the identity provider that provided U with its identity to reveal information that identifies it. The latter allows anonymity revokers to mark and block all accounts created by the user. Importantly, threshold encryption scheme may be set such that no single anonymity revoker may be allowed to reveal user's identity, what is a desired property since an anonymity revoker may be corrupted. On the other hand, no particular anonymity revoker is necessary to decrypt the credentials (that is, it is only required that a big-enough set of anonymity revokers cooperates), what prevents failure of decryption due to anononymity revoker unavailability (e.g. the revoker may be off-line).

3.3 Pistis is a PPCTS

Here we briefly state how we define the properties required for a PPCTS, cf. Section 2.1, for Pistis and argue that they are fulfilled.

Completeness. Completeness of Pistis holds from the completeness of zero-knowledge proof system.

Soundness. Similarly to the above, soundness of Pistis relies on soundness of the zero-knowledge proof system.

Efficiency. Efficiency of the compliance process is assured by using efficient proof systems, in our case – zkSNARKs, which are efficiently provable and verifiable; moreover, the proofs compounds of a constant number of elements despite the size of the statement.

Reliability. This property is assured as PPCTS functionality $\mathbf{F}_{\mathsf{PPCTS}}$ which Pistis implements does not change its state if the provided transaction is non-compliant and the policy assigner does not approved it.

Robustness. Design of Pistis assures that no non-compliant transaction can be processed unless arbitrarily accepted by the policy assigner given that zero-knowledge proof systems that Pistis uses are sound. That holds if the proof systems' structured reference strings (SRS) were generated honestly. This can be provided by using secure multi-party computation (MPC) of the SRS-s, i.e. computing SRS-s be a set of mutually distrusting parties. Depending on the MPC protocols used, it can be guaranteed that the SRS-s are computed honestly if at least one of the parties is honest.

Security. This property is assured as Pistis does not process nor utilize user's secret data from Layer 1, like e.g. a secret key related to a Layer 1 account. Obviously, knowledge of such key may be necessary to execute Pistis. For example, if Pistis is a smart contract on Ethereum, the user uses its secret key to make EVM state transitions related to executing Pistis's commands. However, no system's command takes as input users secret key, nor requires its knowledge.

Adaptability. Pistis is adaptable and customizable in a number of aspects. First of all, Pistis protocol is composed of a number of idealized functionalities, like NIZK, threshold encryption scheme, PRF, functionality of secure parameter generation. Each of these functionalities may be implented independenly, using various cryptographich primitives, etc. Security properties of Pistis guarantees that the protocol will remain secure as long as each of the functionalities' implementations are done securely. Furthermore, the Pistis realizes a UC-functionality of PPCTS, hence it can be securely composed with other UC functionalities.

From the practical point of view, Pistis does not rely on functionalities that are specific to a single blockchain. Hence, although it was created with EVM-compatible blockchains in mind, it is practically blockchain-oblivious.

[**Michal 2.09:** Argument for that point. Pistis is a smart contract and is, to an extend, ledger-neutral. However, I am not sure how we can compose it with other elements. What would be that elements? Oracles? But then, wouldn't we rely security of the system on them? I.e. wouldn't we rely security of the system on some external components we don't control?]

Privacy preservation. This is assured since the only things revealed during transactions are: transaction tx, which in our case hides such elements as transaction's sender, receiver and amount, transaction auxiliary data paux and policy digest dig_{comp}. Furthermore, the sole fact that the transaction is processed does not mean it follows a compliance policy assigned to the sender (as it may be approved by the policy assigner). Also the compliance policy is not revealed at all – the only public information about the policy is that some transaction follows it (shown using a zero knowledge proof) and its digest which is computed using a pseudo-random function.

4 Cryptographic preliminaries and primitives

4.1 Notation

Let $\lambda \in \mathbb{N}$ be the security parameter. We assume that all algorithms described receive as an implicit parameter the security parameter written in unary, 1^λ. An efficient algorithm is a probabilistic Turing machine running in polynomial time, we write PPT to denote a such algorithms. We denote by $\mathsf{Rnd}(\mathcal{A})$ a random variable describing randomness tape of algorithm $\mathcal{A}$. We write $r \leftarrow_{\$} \mathsf{Rnd}(\mathcal{A})$ to denote concrete randomness given to $\mathcal{A}$. Although we usually do not explicitly describe the randomness a PPT algorithm $\mathcal{A}$ is given, we may sometimes do it, in which case we write $\mathcal{A}(x; r)$, which should be interpreted as "algorithm $\mathcal{A}$ runs on input x and randomness r".

For distribution families $\mathcal{A} = \{A(\lambda)\}_{\lambda\in\mathbb{N}}, \mathcal{B} = \{B(\lambda)\}_{\lambda\in\mathbb{N}}$ write $\mathcal{A} \approx_{\varepsilon(\lambda)} \mathcal{B}$ if the statistical distance between $\mathcal{A}$ and $\mathcal{B}$ is upper-bounded by $\varepsilon(\lambda)$. We say that a function $f\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if it vanishes faster than the inverse of any polynomial, i.e. for every $c \in \mathbb{N}$ there exists an integer x' such that for all $x > x'$, we have $|f(x)| < 1/x^c$. We write negl (resp. poly) to denote a negligible (resp. polynomial) function. For a string x of length n we write $\mathsf{len}(x) = n$.

We denote by $\mathbf{R} \in \{0,1\}^* \times \{0,1\}^*$ a polynomial-time decidable binary relation, which associated NP language is defined as $\mathbf{L}_{\mathbf{R}} = \{x \mid \exists w\ s.t.\ (x,w) \in \mathbf{R}\}$. Furthermore, we sometimes abuse notation and write $\mathbf{R}(\mathsf{x},\mathsf{w}) = 0$ if $(\mathsf{x},\mathsf{w}) \notin \mathbf{R}$ and $\mathbf{R}(\mathsf{x},\mathsf{w}) = 1$ if $(\mathsf{x},\mathsf{w}) \in \mathbf{R}$.

In what follows, we refer to a *compliance policy*, as a set of established rules, a financial transaction needs to satisfy and comply with in order to be deemed acceptable. Formally, the compliance policy $\mathbf{P}_{comp}$ is a predicate which can be evaluated to eiher 0 (false) or 1 (true).

For a (finite, abelian, cyclic) group $\mathbb{G}$ we denote by $[1]$ its generator. We use additive notation, that is, $[a]+[b] = [a+b]$ and $a\,[b] = [ab]$. Denote by $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a bilinear pairing, that is for $[a]_1 \in \mathbb{G}_1, [b]_2 \in \mathbb{G}_2, [ab]_T \in \mathbb{G}_T$ holds $e([a]_1, [b]_2) = [ab]_T$. Alternatively, we write $[a]_1 \bullet [b]_2 = [ab]_T$. We specify algorithm GGen that given security parameter 1^λ outputs public parameters p that compounds of description of groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, their generators and bilinear pairing between them. We assume that GGen is run once and for all, that is all of the involved parties, all algorithms, get p as input.

We use UC model by Canetti [Can00]. In our description of ideal functionalities, protocols and simulators we sometimes abuse notation and state that some dataset D contains tuples with party P, or that party P is an input to a procedure. In that case, it should be understand that we do not store, nor pass the whole party P, i.e. all its tapes and description, but only its *unique* identifier $\mathsf{P.id}$.

4.1.1 Signature scheme

A digital signature scheme S consists of 3 algorithms $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ such that:

$\mathsf{KGen}(1^\lambda)$**:** outputs a keypair, made of a public and a secret key $(\mathsf{pk}, \mathsf{sk})$;
$\mathsf{Sig}(\mathsf{sk}, m)$**:** given as input the secret key sk and message m, outputs a signature σ;
$\mathsf{Vf}(\mathsf{pk}, \sigma, m)$**:** verifies whether a signature σ is a valid signature for message m under the key pk.

Furthermore, a signature scheme must be complete and secure (i.e. unforgeable).

Completeness. A valid signature σ for message m should always be accepted by an honest verifier, that is:

$$\Pr\left[\mathsf{Vf}(\mathsf{pk},\sigma,m) = 1 \;\middle|\; (\mathsf{pk},\mathsf{sk}) \leftarrow_{\$} \mathsf{KGen}(1^\lambda), \sigma \leftarrow_{\$} \mathsf{Sig}(\mathsf{sk},m)\right] = 1.$$

Existential unforgeability. It should be intractable for a PPT adversary $\mathcal{A}$ to produce a signature, without knowing the signing key sk, on a message that has not been signed yet. More precisely, $\mathcal{A}$ has negligible probability of winning the game described in Fig. 1.

EUF-CMA

$(\mathsf{pk},\mathsf{sk}) \leftarrow_{\$} \mathsf{KGen}(1^\lambda)$
$(m',\sigma') \leftarrow \mathcal{A}^{O_{\mathsf{sk}}}(\mathsf{pk})$
if $\mathsf{Vf}(\mathsf{pk},m',\sigma') = 1 \wedge (m',\sigma') \notin Q$
  return 1
return 0

Fig. 1. Existential unforgeability of a signature scheme. Oracle O_{sk} takes as input messages m and outputs corresponding signatures σ; pair (m,σ) is then assigned to list Q.

4.1.2 Zero-knowledge proofs

A zero knowledge proof system PS for a family of polynomial-time decidable binary relations $\mathcal{R} = \{\mathbf{R}\}$, is a tuple of 4 algorithms $(\mathsf{KGen}, \mathsf{Pro}, \mathsf{Ver}, \mathsf{Sim})$ that has the following properties.

Completeness. We say that a proof system is *complete* if a proof for a true statement produced by an honest prover is always accepted by an honest verifier, that is for all $\mathbf{R} \in \mathcal{R}$ and $(\mathsf{x},\mathsf{w}) \in \mathbf{R}$

$$\Pr[\mathsf{Ver}(\mathbf{R},\mathsf{srs},\mathsf{x},\pi) = 1 \mid \mathsf{srs} \leftarrow_{\$} \mathsf{KGen}(\mathbf{R}), \pi \leftarrow_{\$} \mathsf{Pro}(\mathbf{R},\mathsf{srs},\mathsf{x},\mathsf{w})] = 1.$$

Zero knowledge. A proof system is *zero-knowledge* if the verifier seeing a proof for a statement x learns nothing besides the veracity of the statement. Formally, we require that there exists a PPT simulator Sim that equipped with a trapdoor td for any $\mathsf{x} \in \mathbf{L}_{\mathbf{R}}$ provides a simulated proof that is indistinguishable from a proof output by an honest user. That is,

$$\{\mathsf{Pro}(\mathbf{R}, \mathsf{srs}, \mathsf{x}, \mathsf{w})\}_{(\mathsf{srs},\mathsf{td}) \leftarrow\!\$\, \mathsf{KGen}(\mathbf{R}), (\mathsf{x},\mathsf{w})} \approx_{\varepsilon} \{\mathsf{Sim}(\mathbf{R}, \mathsf{srs}, \mathsf{td}, \mathsf{x})\}_{(\mathsf{srs},\mathsf{td}) \leftarrow\!\$\, \mathsf{KGen}(\mathbf{R}), (\mathsf{x},\mathsf{w})} .$$

Soundness. A proof system is *sound* if a dishonest prover, who tries to convince a verifier of the veracity of a false statement, has negligible chances to succeed. That is, for all $\mathbf{R} \in \mathcal{R}$ and all PPT adversaries $\mathcal{A}$

$$\Pr[\mathsf{Ver}(\mathbf{R}, \mathsf{srs}, \mathsf{x}, \pi) = 1 \mid (\mathsf{srs}, \mathsf{td}) \leftarrow\!\$\, \mathsf{KGen}(\mathbf{R}), (\mathsf{x}, \mathsf{w}, \pi) \leftarrow\!\$\, \mathcal{A}(\mathbf{R}, \mathsf{srs}), \mathsf{x} \notin \mathbf{L}_{\mathbf{R}}] \leq \mathsf{negl}(\lambda) .$$

Knowledge soundness. In some cases we require from the proof system to be *knowledge sound*. Intuitively, knowledge soundness holds when a PPT is able to produce a valid proof for a statement only when it knows the statement's witness. More precisely, for any PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathsf{Ext}_{\mathcal{A}}$ such that

$$\Pr\left[\begin{array}{l|l} \mathsf{Ver}(\mathbf{R}, \mathsf{srs}, \mathsf{x}, \pi) = 1 & (\mathsf{srs}, \mathsf{td}) \leftarrow\!\$\, \mathsf{KGen}(\mathbf{R}), r \leftarrow\!\$\, \mathsf{Rnd}(\mathcal{A}) \\ \wedge\, \mathbf{R}(\mathsf{x}, \mathsf{w}) \neq 1 & (\mathsf{x}, \mathsf{w}, \pi) \leftarrow\!\$\, \mathcal{A}(\mathbf{R}, \mathsf{srs}; r), \mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathbf{R}, \mathsf{srs}, r) \end{array}\right] \leq \mathsf{negl}(\lambda) .$$

Simulation extractability. Unfortunately, knowledge soundness is often not sufficient to prevent attacks in real-life applications (e.g. a knowledge-sound proof system does not assure that a malicious user cannot get a valid proof from an honest prover, maul it, and present this mauled proof as their own). Simulation extractability is a stronger notion that prevents such attacks. Formally, a proof system is simulation extractable if for any PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathsf{Ext}_{\mathcal{A}}$ such that

$$\Pr\left[\begin{array}{l|l} \mathsf{Ver}(\mathbf{R}, \mathsf{srs}, \mathsf{x}, \pi) = 1 & (\mathsf{srs}, \mathsf{td}) \leftarrow\!\$\, \mathsf{KGen}(\mathbf{R}), (\mathsf{x}, \mathsf{w}, \pi) \leftarrow\!\$\, \mathcal{A}^{O}(\mathbf{R}, \mathsf{srs}; r), \\ \wedge\, \mathbf{R}(\mathsf{x}, \mathsf{w}) \neq 1 & (\mathsf{x}, \pi) \notin Q, \mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathbf{R}, \mathsf{srs}, r) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where, O on $\mathcal{A}$'s input x responds with a simulated proof π for x; Q is a list of all instances x submitted by $\mathcal{A}$ along with the oracle's responses π.

We call a proof system *universal* if KGen depends only on the security parameter λ, not the concrete relation $\mathbf{R}$ itself. That is, a single SRS srs can be used for all relations $\mathbf{R} \in \mathcal{R}$ of a given size. For the sake of efficiency, a universal proof system may have specified additional SRS generating algorithm $\mathsf{KGenSpec}$ which takes srs output by KGen and produces a specialized SRS $\mathsf{srs}^{\mathsf{spec}}$ that allows both the prover and verifier to make and verify proofs more efficiently.

Remark 3 (On transaprent NIZKs). We note that the definitions above focus on SRS-based NIZKs, however other approaches are possible. For example, one could use so-called transparent zero-knowledge proof systems which are made non-interactive in the random-oracle model [BR93] via the Fiat–Shamir transformation [FS87]. In that case, the KGen algorithm may just return an empty string. Also, in that case, the simulator does not learn the trapdoor corresponding to the SRS. Zero-knowledge of Fiat–Shamir based NIZKs relies on programmability of the random oracle. More precisely, the simulator is able to program the oracle to return y, picked by Sim, when it is given x.

It is argued sometimes that random oracle-based NIZKs can be instantiated using a hash function. Such statements have to be taken with a pinch of salt as zero knowledge requires either the hash function programmability or use of much less efficient transformations than Fiat–Shamir's.

4.1.3 Threshold encryption scheme

We give the definition of the threshold encryption scheme following [DGK+20].

A (t, p)-threshold encryption scheme $\mathsf{TE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Comb})$ over a message space M compounds the following algorithms:

$\mathsf{KGen}(1^{\lambda})$**:** takes security parameter and returns a pair of a public and secret keys $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}(\mathbf{pk}, m)$**:** encrypts message $m \in M$ using a vector of public keys $\mathbf{pk} = (\mathsf{pk}_i)_{i \in [1..\mathsf{p}]}$, such that any $(\mathsf{t}+1)$-sized set of pk_i owners should be able to decrypt it. Denote by c_i the share of m encrypted using pk_i.

$\mathsf{Dec}(\mathsf{sk}_i, \mathsf{c}_i)$**:** A party who knows a secret key sk_i corresponding to a public pk_i decrypts c_i and publishes the decrypted share s_i.

$\mathsf{Comb}(\mathbf{pk}, \mathsf{c}, \{\mathsf{s}_i\}_{i \in I'})$**:** Takes a subset $I' \subset \{1, \ldots, \mathsf{p}\}$ of size at least $\mathsf{t}+1$ and outputs either a message m or $\perp$.

We further define below the notion of a secret sharing scheme. A (t, p)-secret sharing scheme SS is made of the following algorithms

$\mathsf{Share}_{\mathsf{t},\mathsf{p}}(m)$**:** Which on input a secret message m produces p shares $\mathbf{s} = (\mathsf{s}_1, \ldots, \mathsf{s}_{\mathsf{p}})$ such that for any subset $I \subset \{1, \ldots, \mathsf{p}\}$ of size at least $\mathsf{t}+1$ is able to reconstruct m from $\{\mathsf{s}_i\}_{i \in I}$. Importantly, m remains random even when a subset of shares of size smaller than $\mathsf{t}+1$ is known.

$\mathsf{Reconstruct}_{\mathsf{t},\mathsf{p}}(\{\mathsf{s}_i\}_{i \in I'})$**:** For $I' \subset \{1, \ldots, \mathsf{p}\}$ of size at least $\mathsf{t}+1$ it reconstructs the shared message m.

Share-and-Encrypt paradigm. A (t, p)-threshold encryption scheme TE can be instantiated using a public key encryption scheme $\mathsf{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ and a secret sharing scheme $\mathsf{SS} = (\mathsf{Share}_{\mathsf{t},\mathsf{p}}, \mathsf{Reconstruct}_{\mathsf{t},\mathsf{p}})$. Let $I = \{1, \ldots, \mathsf{p}\}$, then TE can be instantiated as follows:

$\mathsf{TE.KGen}(1^\lambda)$**:** For each $i \in I$ return $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow_\$ \mathsf{E.KGen}(1^\lambda)$.
$\mathsf{TE.Enc}(\mathbf{pk}, m)$**:** Compute $\mathsf{s} \leftarrow_\$ \mathsf{SS.Share}(\mathsf{t}, \mathsf{p}, m)$ and $\mathsf{c}_i \leftarrow_\$ \mathsf{E.Enc}(\mathsf{pk}_i, \mathsf{s}_i)$. Send c_i to party i.
$\mathsf{TE.Dec}(\mathsf{sk}_i, \mathsf{c}_i)$**:** Decrypt and publish the share $\mathsf{s}_i \leftarrow \mathsf{E.Dec}(\mathsf{sk}_i, \mathsf{c}_i)$.
$\mathsf{TE.Comb}(\mathbf{pk}, \{\mathsf{c}_i\}_{i\in I}, \{\mathsf{s}_i\}_{i\in I'})$: For a set $I' \subset I$ of size at least $\mathsf{t}+1$ combine the decrypted shares by running $m \leftarrow \mathsf{SS.Reconstruct}_{\mathsf{t},\mathsf{p}}(\{\mathsf{s}_i\}_{i\in I'})$.

We additionaly require that the threshold encryption scheme we use is *partialy decryption simulatable* [DGK+20]. More precisely, a (t, p) threshold encryption scheme is (t, p)-partial decryption simulatable if there exists an efficient algorithm $\mathsf{SimPart}$ such that for all PPT adversaries $\mathcal{A}$

$$\left| \Pr\left[\mathsf{Game}^{\mathsf{pds}}_{\mathcal{A}}(\lambda, \mathsf{t}, \mathsf{p}) = 1 \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)\,,$$

for $\mathsf{Game}^{\mathsf{pds}}_{\mathcal{A}}(\lambda, \mathsf{t}, \mathsf{p})$ as defined at Fig. 2.

$\mathsf{Game}^{\mathsf{pds}}_{\mathcal{A}}(\lambda, \mathsf{t}, \mathsf{p})$

$C \leftarrow \mathcal{A}(\lambda, \mathsf{t}, \mathsf{p})$
for $i \in [1..\mathsf{p}]$
  $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow_\$ \mathsf{KGen}(1^\lambda)$
$(R, m_0, m_1) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i\in[\mathsf{p}]}, \{\mathsf{sk}_i\}_{i\in C})$
$b \leftarrow_\$ \{0, 1\}$
$c_0 \leftarrow_\$ \mathsf{Enc}(\mathbf{pk}_R, m_0)$
$c_1 \leftarrow_\$ \mathsf{Enc}(\mathbf{pk}_R, m_1)$
if $b = 0$ **then**
  for $i \in R \setminus C$
    $\mu_i \leftarrow \mathsf{Dec}(\mathbf{sk}_R, c_0)$
else
  for $i \in R \cap C$
    $\mu_i \leftarrow \mathsf{Dec}(\mathbf{sk}_R, c_1)$
  for $i \in R \setminus C$
    $\mu_i \leftarrow \mathsf{SimPart}(\mathsf{t}, \mathsf{p}, \mathbf{pk}_R, c_0, \{\mu_j\}_{j\in R\cap C}, m_0)$
$b' \leftarrow \mathcal{A}(c_b, \{\mu_j\}_{j\in R\setminus C})$
if $b = b' \wedge |m_0| = |m_1| \wedge |R \cap C| \leq \mathsf{t} \wedge R \subset [\mathsf{p}] \wedge C \subset [\mathsf{p}]$
then return 1 **else return** 0

Fig. 2. Partial decryption simulatability game

4.1.4 Pseudorandom functions

A function family $\mathsf{PRF} = \{\mathsf{PRF}_k : A \to B\}_{k\in K}$ is called *pseudorandom* if for a set of keys $K \in \{0,1\}^\lambda$, function h randomly picked from all functions mapping A to B, any PPT adversary $\mathcal{A}$ holds

$$\left|\Pr\left[\mathcal{A}^{\mathsf{PRF}_k}(1^\lambda) = 1 \;\middle|\; k \leftarrow_\$ K\right] - \Pr\left[\mathcal{A}^{h}(1^\lambda) = 1 \;\middle|\; k \leftarrow_\$ K\right]\right| \leq \mathsf{negl}(\lambda)\,.$$

Additionally, in our case we follow [DGK+20] and require from the PRF that it is *weakly robust*. That is, no PPT adversary $\mathcal{A}$, given should be able to come with a key k^* and function input x^* that collides with some other function. More precisely, we call a PRF *weakly robust* if

$$\Pr\left[\exists (x,y) \in Q, \mathsf{PRF}_{\mathsf{k}^*}(x^*) = y \;\middle|\; \mathsf{k} \leftarrow_\$ K, (x^*, \mathsf{k}^*) \leftarrow \mathcal{A}^{\mathsf{PRF}_\mathsf{k}}\right] \leq \mathsf{negl}(\lambda)\,,$$

where Q is a list of all queries and their responses send by the adversary to PRF_k.

5 The protocol

5.1 Primitives' instantiations

Zero-knowledge proof systems [Michal 27.08: Which zkSNARK?]

The choice of zkSNARK used to prove relations is of great importance for efficiency of the system and should be done separately for each system instantiation after considering how the system is supposed to be used. The main trade-off here is between using a universal on non-universal zkSNARK. The latter, like [Gro16] is more efficient, but requires a separate SRS for each of the proven relation. This is especially troublesome if one desires the compliance policy relation $\mathbf{R}_{comp}$ or attribute list relation $\mathbf{R}_{al}$ to be universal relations, i.e. a single relations that may be used to prove virtually any statement, for any compliance policy or attribute list predicate. However, if one designs a system that has a single compliance policy and single attribute list policy, then use of a more flexible proof system is not necessary.

However, when a more general and customizable system is required, one may greatly benefit if it chooses a universal proof system like, e.g. Plonk [GWC19], Sonic [MBKM19], Marlin [CHM+20], Lunar [CFF+20], Basilisk [**?**]. Although less efficient than [Gro16], each of these systems allows to have a single universal SRS for *all* relations (however, efficiently proving or verifying a proof requires a specialized SRS which can be computed from the universal SRS by each of the parties separately), which reduces greatly the number of required reference strings.

[Michal 27.08: Give description of the proof system – here or in the appendix]

$\mathbf{R}_{val}$ which takes $((\mathsf{tx}, \mathsf{paux}_{tx}), (\mathsf{saux}_{tx}))$ and returns 1 if the transaction tx is valid from the transacting system point of view.

$\mathbf{R}_{comp}$ which verifies that the transaction submitted by the user complies with the compliance policy it got assigned **or** the user knows a policy assigner's signature on the transaction. More precisely,

$$\mathbf{R}_{comp} = \left\{ \begin{array}{l} ((\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{dig}_{comp}), \\ (\mathbf{P}_{comp}, \sigma_{comp}, \sigma_{tx}, \mathsf{k}, r)) \end{array} \middle| \begin{array}{l} \mathsf{dig}_{comp} = \tilde{\mathsf{PRF}}_{\mathsf{k}}(\mathbf{P}_{comp}, \sigma_{comp}, r), \mathsf{S.Vf}(\mathsf{pk}_{\mathsf{PA}}, \sigma_{comp}, \mathbf{P}_{comp}) = 1 \\ \mathbf{P}_{comp}(\mathsf{tx}, \mathsf{paux}_{tx}) = 1 \vee \mathsf{S.Vf}(\mathsf{pk}_{\mathsf{PA}}, \sigma_{tx}, (\mathsf{tx}, \mathsf{paux})) = 1 \end{array} \right\}$$

$\mathbf{R}_{acc}$ which pair of public and secret key $(\mathsf{pk}_{acc}, \mathsf{sk}_{acc})$ associated with the account has to fulfil. The relation highly depends on the ledger. Intuitively, the secret key should allow user to sign and send transactions, while the public key should allow others to verify signatures and transactions send by the user.

$\mathbf{R}_{newacc}$ which verifies that the account has been created using correct user's secrets and the party which creates the account has correctly assigned identity and policy. More precisely,

$$\mathbf{R}_{newacc} = \left\{ \begin{array}{l} ((\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{pk}_{\mathsf{I}}, \mathsf{dig}_{comp}, \mathbf{pk}_{\mathbf{AR}}), \\ (\sigma_{id}, r_{tenc}, \mathsf{pcred}, \mathsf{scred}, \mathsf{k}, \\ \mathsf{sk}_{acc}, \mathbf{P}_{comp}, \sigma_{comp}, r_{pd}, \mathsf{AL}, x, \mathbf{P}_{al}, \sigma_{al})) \end{array} \middle| \begin{array}{l} \mathsf{BS.Vf}(\mathsf{pk}_{\mathsf{I}}, \sigma_{id}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL})) = 1 \\ \mathsf{aux}_{acc} = \mathsf{PRF}_{\mathsf{k}}(x), \text{ for some } x \in [1..\mathsf{maxAccounts}] \\ \mathsf{c}_{pcred} = \mathsf{TE.Enc}((\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}}), \mathsf{pcred}; r_{tenc}), \\ \mathbf{R}_{acc}(\mathsf{pk}_{acc}, \mathsf{sk}_{acc}) = 1, \mathbf{R}_{user}(\mathsf{pcred}, \mathsf{scred}) = 1, \\ \mathsf{dig}_{comp} = \tilde{\mathsf{PRF}}_{\mathsf{k}}(\mathbf{P}_{comp}, \sigma_{comp}, r_{pd}), \\ \mathsf{S.Vf}(\mathsf{pk}_{\mathsf{PA}}, \sigma_{comp}, (\mathbf{P}_{comp}, \mathsf{pcred})) = 1, \\ \mathsf{S.Vf}(\mathsf{pk}_{\mathsf{PA}}, \sigma_{al}, (\mathbf{P}_{al}, \mathsf{pcred})) = 1, \mathbf{P}_{al}(\mathsf{AL}) = 1 \end{array} \right\}.$$

$\mathbf{R}_{user}$ which public and secret credentials of the user have to fulfil. That is, $\mathbf{R}_{user} = \{(\mathsf{pcred}, \mathsf{scred})\}$ Following [DGK+20] we require that $\mathbf{R}_{user}$ is a hard relation.

$\mathbf{R}_{sign}$ that user's signature has to fulfil, more precisely

$$\mathbf{R}_{sign} = \left\{ \begin{array}{l} (\mathsf{pk}_{\mathsf{U}}, \mathsf{pk}_{\mathsf{I}}, \mathbf{pk}_{\mathbf{AR}}, \sigma_1, c, \mathsf{AL}, \mathsf{p}), \\ (\mathsf{sk}_{\mathsf{U}}, \mathsf{k}, r', r) \end{array} \middle| \begin{array}{l} \mathbf{R}_{user}(\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}}) = 1 \\ \sigma_1 = \mathsf{PrepSign}(\mathsf{pk}_{\mathsf{I}}, (\mathsf{sk}_{\mathsf{U}}, \mathsf{k}, \mathsf{AL}), r') \\ c = \mathsf{TE.Enc}(\mathbf{pk}_{\mathbf{AR}}, \mathsf{k}; r) \end{array} \right\}$$

$\mathbf{R}_{al}$ which shows that user's attribute list AL satisfies predicate $\mathbf{P}_{al}$ requested by policy assigner PA before the compliance policy can be assigned, that is

$$\mathbf{R}_{al} = \left\{ \begin{array}{l} (\mathsf{pAL}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pcred}, c, \mathsf{pk}_{polas}), \\ (\mathsf{sAL}, \sigma_{id}, \mathsf{scred}, \mathsf{k}, \mathbf{P}_{al}) \end{array} \middle| \begin{array}{l} (\mathsf{pAL}, \mathsf{sAL}) \leftarrow \mathsf{AL}, \mathbf{P}_{al}(\mathsf{AL}) = 1, \\ \mathsf{BS.Vf}(\mathsf{pk}_{id}, \sigma_{id}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL})) = 1, \\ \mathbf{R}_{user}(\mathsf{pcred}, \mathsf{scred}) = 1, c = \mathsf{Enc}(\mathsf{pk}_{polas}, \mathbf{P}_{al}) \end{array} \right\}$$

$\mathbf{R}_{sig}, \mathbf{R}_{bsig}, \mathbf{R}_{enc}, \mathbf{R}_{tenc}$ which determine relations between public and secret key of a signature scheme S, blind signature scheme BS, encryption scheme E and threshold encryption scheme TE.

Remark 4 (Combining statements). We note that some of the relations above could be combined and proven together for the sake of efficiency.

Remark 5 (On universal relations). Depending on application, relations $\mathbf{R}_{al}$ and $\mathbf{R}_{comp}$ may be universal – each of them may be used to show any NP relation. However, they are used to show specific relations – $\mathbf{P}_{al}, \mathbf{P}_{comp}$ that are accounted for when the proven instance is set. That is, for $\mathbf{R}_{al}$ proven relation $\mathbf{P}_{al}$ is set as a part of the instance. For $\mathbf{R}_{comp}$ the relation $\mathbf{P}_{comp}$ remains hidden (as it is private) but the $\mathbf{R}_{comp}$ instance contains uniquely determined $\mathbf{P}_{comp}$'s, so called, policy digest dig_{comp}.

5.1.1 Threshold encryption scheme

Following [DGK+20] we build the threshold encryption scheme using a Shamir's secret sharing scheme with a CLT [CLT18] encryption scheme. The latter relies on a so-called CL framework [CL15] which composes of two algorithms SGen and Solve defined as follows[14]:

$\mathsf{SGen}(1^\lambda, q)$**:** 1. Take a security parameter λ and prime q and produce the following:
- $\hat{\mathbb{G}}$ a finite abelian group of order $\hat{n} \leftarrow q \cdot \hat{s}$. The bitsize of $\hat{s}$ is a function of λ, $\gcd(q, \hat{s}) = 1$. It is required that valid encodings of elements in $\hat{\mathbb{G}}$ can efficiently be recognized.
- $\mathbb{G}$ a cyclic subgroup of $\hat{\mathbb{G}}$ of order $n \leftarrow q \cdot s$ where s divides $\hat{s}$.
- $\mathbb{H}$ the unique cyclic group of order q generated by $\mathfrak{h}$.
- $\mathbb{G}^q$ the subgroup of $\mathbb{G}$ of order s. Since $\mathbb{H} \subset \mathbb{G}$, it holds that $\mathbb{G} \simeq \mathbb{G}^q \times \mathbb{H}$.
- $\tilde{s}$ an upper bound for $\hat{s}$.
- $\mathfrak{g}, \mathfrak{h}, \mathfrak{g}_q$ generators of $\mathbb{G}, \mathbb{H}, \hat{\mathbb{G}}$, $\mathfrak{g} \leftarrow \mathfrak{h} + \mathfrak{g}_q$.

2. Return $\mathsf{p} \leftarrow (\hat{\mathbb{G}}, \mathbb{G}, \mathbb{H}, \tilde{s}, \mathfrak{g}, \mathfrak{h}, \mathfrak{g}_q)$.

$\mathsf{Solve}(q, \mathsf{p}, X)$**:** Solves discrete logarithm problem in $\mathbb{H}$.

Finally the encryption scheme is defined as follows. [**Michal 10.1:** I skip for now description of SSS as it is very standard but describing it takes time]

$\mathsf{SGen}(1^\lambda, 1^\mu)$: 1. sample a μ-bit prime q;
2. $\mathsf{p} \leftarrow_\$ \mathsf{GGen}(1^\lambda, q)$;
3. return p;

$\mathsf{KGen}(\mathsf{p})$**:** 1. sample $\alpha \leftarrow_\$ \mathcal{D}_q$, $y \leftarrow_\$ \alpha \cdot \mathfrak{g}_q$;
2. set $\mathsf{pk} \leftarrow y$, $\mathsf{sk} \leftarrow \alpha$;
3. return $(\mathsf{pk}, \mathsf{sk})$;

$\mathsf{Enc}(\mathsf{pk}, m)$**:** 1. sample $r \leftarrow_\$ \mathcal{D}_q$;
2. return $(r \cdot \mathfrak{g}_q, m \cdot \mathfrak{h} + r \cdot y)$;

$\mathsf{Dec}(\mathsf{sk}, (\mathfrak{c}_1, \mathfrak{c}_2))$**:** 1. compute $\mathfrak{m} \leftarrow \mathfrak{c}_2 - \alpha \cdot \mathfrak{c}_1$;
2. return $\mathsf{Solve}(q, \mathsf{p}, \mathfrak{m})$;

Here $\mathcal{D}_q$ is a distribution that is $\mathsf{negl}(\lambda)$-close (in terms of statistical distance) to the uniform distribution over elements of $\mathbb{G}^q$.

5.1.2 (Blind) signature scheme

Similarly to [DGK+20] we use Poincheval-Sanders [PS16] signature scheme as a blind signature scheme. We also use it when a standard signature scheme is required.

$\mathsf{SGen}(1^\lambda)$**:** Sample $\mathsf{p} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, [1]_1, [1]_2)$.

$\mathsf{KGen}(\mathsf{p})$**:** Given p
1. For $i \in [1..\ell]$, sample $x, y_i \leftarrow_\$ \mathbb{F}_p$.
2. Compute $[x]_1, [x]_2, [y_i]_1, [y_i]_2$ for $i \in [1..\ell]$.
3. Output $\mathsf{pk} = [x]_1, [x]_2, \{[y_i]_1, [y_i]_2\}_{i \in [1..\ell]}$ and $\mathsf{sk} = [x]_1$.

$\mathsf{PrepSign}(\mathsf{pk}, m = (m_1, \ldots, m_\ell))$**:** 1. Sample $r \leftarrow_\$ \mathbb{F}_p$.
2. Compute $\sigma_1 = [r]_1 + \sum_{i=1}^{\ell} m_i [y_i]_1$

$\mathsf{BlindSign}(\mathsf{sk}, \sigma_1)$**:** 1. Sample $\alpha \leftarrow_\$ \mathbb{F}_p$,
2. Output $\sigma_2 = ([\alpha]_1, \alpha \cdot [x + \sigma_1]_1)$.

$\mathsf{Unblind}(\sigma_2, r)$**:** 1. Parse σ_1 as $\mathfrak{a}, \mathfrak{b}$ and compute $\sigma' = (\mathfrak{a}, \mathfrak{b} - r \cdot \mathfrak{a})$.
2. Randomize σ' by choosing $r' \leftarrow_\$ \mathbb{F}_p$ and computing $\sigma = r'\sigma'$.
3. Return σ.

$\mathsf{Vf}(\mathsf{pk}, m = (m_1, \ldots, m_\ell), \sigma)$**:** 1. Parse signature σ as (σ_1, σ_2).
2. Output 1 if:
   - $\sigma_1 \neq [1]_1$, and
   - $\sigma_1 \bullet [x]_2 + \sum_{j=1}^{\ell} [y_j]_2 = \sigma_2 \bullet [1]_2$.

[14] Both CL framework and CLT encryption scheme definitions were taken from [DGK+20]

5.1.3 Pseudorandom function

In our scheme we use two pseudorandom functions, which we denote PRF and $\tilde{\mathsf{PRF}}$. First, similarly to [DGK+20], we use Dodis-Yampolskiy [DY05] PRF. More precisely, let $[1]_1$ be generator of an order-p group $\mathbb{G}_1$. On input x and key $\mathsf{k} \leftarrow_{\$} \mathbb{F}_p$, $\mathsf{PRF} = \left[\frac{1}{\mathsf{k}+x}\right]_1$.

Security of this function holds only for small domains, that is however enough as the maximum number of accounts the user can set up is bounded by maxAccounts. The function is also weakly robust.

Second, we use as $\tilde{\mathsf{PRF}}$ a hash function H assuming it is a weakly robust PRF. Note that we cannot use Dodis-Yampolskiy's PRF here as it is secure only against small domains, what is not the case for $\tilde{\mathsf{PRF}}$. We propose to use here MiMC [AGR+16] hash function as it combines well with (most) zkSNARKs. (That is, proving statements regarding MiMC is usually much more efficient – for the prover – than showing statements for, say, SHA3.)

5.1.4 Data objects

Following [DGK+20] we define the following data objects held by the system users:

User certificate (UC) After registration at an identity provider the user gets a certificate containing
- Public and secret identity credentials pcred and scred;
- PRF key k;
- Attribute list AL; we note the public and secret part of the attribute list by pAL and sAL respectively. Importantly, it may hold that $\mathsf{pAL} = \emptyset$;
- Identity provider's signature on $(\mathsf{scred}, \mathsf{k}, \mathsf{AL})$.

Account creation information (ACI) The user equipped with an identity from the identity provider and UC can create new accounts. The corresponding data object ACI is posted on the ledger and contains:
- aux_{acc}, the (unique) account registration ID;
- c_{pcred}, the threshold encryption of public credentials pcred;
- Identifier I of the identity provider who provided the user with the identity;
- Account public key pk_{acc};
- π, a zero-knowledge proof that certifies that the account has been created by an authorized user.

Identity Provider's Information on User (IPIU) which is a data stored by the identity provider after granting identity to a user. The data set contains:
- Identifier U of the user and its public credentials pcred;
- Set of the anonymity revokers who keeps encryption of the user's PRF key c_{key}.

Additionally, since our protocol also allows users to append transactions to the ledger, we specify the following data object:

Transaction data (TxData) which contains all transaction-related information that are to be posted on the ledger. That is,
- transaction description tx, **Antoine 23.08:** What's that? If we have the originator and "public tx data" in the field below, what do we have in the "tx description"? In practical terms, do you mean the "gasPrice" and other fields making up the tx object? If we don't want to get down the "implementation level", I find this bullet confusing here as its intent is not very clear. Not clear how that relates to the other fields mentioned below.
- auxiliary public transaction data paux_{tx},
- transaction originator policy assigner's PA public key $\mathsf{pk}_{\mathsf{PA}}$ and a compliance policy digest dig_{comp},
- zero-knowledge proofs for transaction validity and transaction compliance.
- Account identifier digest aux_{acc}.

5.2 Functionality and protocol for PPCTS

Application functionality $\mathbf{F}_{\mathsf{PPCTS}}$

We identify the following parties: identity providers $\mathrm{I} = \{\mathsf{I}_1, \ldots, \mathsf{I}_\mathsf{i}\}$, anonymity revokers $\mathrm{AR} = \{\mathsf{AR}_1, \ldots, \mathsf{AR}_{\mathsf{ar}}\}$, users $\mathrm{U} = \{\mathsf{U}_1, \ldots, \mathsf{U}_\mathsf{u}\}$, application owner O, relays $\mathrm{R} = \{\mathsf{R}_1, \ldots, \mathsf{R}_\mathsf{r}\}$, $\mathrm{R} \subseteq \mathrm{U}$, and policy assigners $\mathrm{PA} = \{\mathsf{PA}_1, \ldots, \mathsf{PA}_{\mathsf{pa}}\}$. The functionality is parameterized by

- values ar, i, pa and threshold t;
- relation $\mathbf{R}_{acc}$ that an account private key-public key pair has to fulfill;
- relation $\mathbf{R}_{val}$ that a transaction has to fulfill to be considered valid according to the rules of the transacting system;
- maximal number of accounts maxAccounts a user can open.

Flag ready is initially set to false.

Initialization On $(\texttt{initialize})$ from a party $\mathsf{P} \in \mathrm{I} \cup \mathrm{AR} \cup \mathrm{PA} \cup \mathsf{O} \cup \mathrm{R}$ output to $\mathcal{A}$ $(\texttt{initialized}, \mathsf{P})$. When all parties are initialized set flag $\mathsf{ready} = \mathsf{true}$.

Identity issuance On $(\texttt{issueIdentity}, \mathsf{I}, \mathsf{AL})$ from user U, and $(\texttt{issueIdentity}, \mathsf{U})$ from I:

- if $\mathsf{ready} = \mathsf{false}$, ignore;
- if there is already stored $(\texttt{issuedIdentity}, \mathsf{U}, \mathsf{I}, \mathsf{aux}_{id})$ then ignore; else send $(\texttt{issueIdentity})$ (or $(\texttt{issueIdentity}, \mathsf{U}, \mathsf{I}, \mathsf{AL})$ if I is corrupted) to $\mathcal{A}$.
- Upon receiving $(\texttt{issueIdentity}, \mathsf{aux}_{id})$ from $\mathcal{A}$: if there already is stored tuple $(\texttt{identityIssued}, \mathsf{R}, \mathsf{I}', \mathsf{aux}'_{id})$ for $\mathsf{aux}'_{id} = \mathsf{aux}_{id}$; or $\mathsf{aux}_{id} = \bot$, then abort; otherwise, store $(\texttt{identityIssued}, \mathsf{U}, \mathsf{I}, \mathsf{aux}_{id})$.
- Output $(\texttt{identityIssued}, \mathsf{aux}_{id})$ to $\mathcal{A}$.

Attribute check On $(\texttt{checkAttributes}, \mathsf{PA})$ from the user and $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ from the policy assigner

- if $\mathsf{ready} = \mathsf{false}$, ignore;
- if there is no $(\texttt{issuedIdentity}, \mathsf{U}, \cdot, \cdot)$ stored, then ignore;
- if $\mathbf{P}_{al}(\mathsf{AL}) = 0$ then return to user, policy assigner, and the adversary $(\texttt{attributesNotValid})$ and exit
- else send to the adversary $(\texttt{attributesValid})$ and to the user $(\texttt{attributesValid}, \mathbf{P}_{al})$
- store $(\texttt{attributesValid}, \mathsf{U}, \mathbf{P}_{al})$.

Policy issuance On $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{AL})$ from user U, and $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{al}, \mathbf{P}_{comp})$ from PA:

- if there is no $(\texttt{attributesValid}, \mathsf{U}, \mathbf{P}_{al})$ then ignore; else
- if there is already stored $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{comp}, \ldots)$ then ignore; else send $(\texttt{issuePolicy}, \mathsf{PA})$ to $\mathcal{A}$.
- Store $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al}, \mathbf{P}_{comp})$; if PA or U is corrupted, send $(\texttt{policyAssigned}, \mathsf{pcred}, \mathsf{AL}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al}, \mathbf{P}_{comp})$ to $\mathcal{A}$.

Account creation On $(\texttt{createAccount}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc})$ from user U

- If $\mathsf{ready} = \mathsf{false}$ or $\mathbf{R}_{acc}(\mathsf{pk}_{acc}, \mathsf{sk}_{acc}) \neq 1$ ignore.

Else,

- If $\mathsf{aux}_{id} = \bot$, then abort, else read $(\texttt{identityIssued}, \mathsf{U}, \mathsf{I}, \mathsf{aux}_{id}, \mathsf{PA})$ and abort if there is no such entry;
- Check whether there is $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{comp}, \mathbf{P}_{al})$ entry stored, and abort if there is not the case.
- Output $(\texttt{createAccount}, \mathsf{pk}_{acc}, \mathsf{I}, \mathsf{PA})$ to $\mathcal{A}$;
- On $(\texttt{createAccount}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$ from $\mathcal{A}$, if $\mathsf{aux}_{acc} = \bot$, $\mathsf{dig}_{comp} = \bot$ or there is already an account with the same aux_{acc} or dig_{comp}, then abort, else
  - store $(\texttt{accountCreated}, \mathsf{PA}, \mathbf{P}_{comp}, \mathsf{dig}_{comp}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc})$,
  - set $\mathsf{Count}(\mathsf{aux}_{id}) \leftarrow \mathsf{Count}(\mathsf{aux}_{id}) + 1$,
  - add $(\mathsf{aux}_{acc}, \mathsf{dig}_{comp}, \mathsf{pk}_{acc}, \mathsf{I}, \mathsf{PA})$ to the buffer and the adversary;
- Return $(\texttt{accountCreated}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$ to U.

Shielded account creation On $(\texttt{createAccountShielded}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc}, \mathsf{R})$ from user U and $(\texttt{createAccountShielded}, \mathsf{pk}_{acc})$ from R.

- If $\mathsf{ready} = \mathsf{false}$ or $\mathbf{R}_{acc}(\mathsf{pk}_{acc}, \mathsf{sk}_{acc}) \neq 1$ ignore.

Else,

- If $\mathsf{aux}_{id} = \bot$, then abort, else read $(\texttt{identityIssued}, \mathsf{U}, \mathsf{I}, \mathsf{aux}_{id}, \mathsf{PA})$ and abort if there is no such entry;
- Check whether there is $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{comp}, \mathbf{P}_{al})$ entry stored, and abort if there is not the case.
- Output $(\texttt{createAccountShielded}, \mathsf{pk}_{acc}, \mathsf{I}, \mathsf{PA})$ to $\mathcal{A}$;
- On $(\texttt{createAccountShielded}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$ from $\mathcal{A}$, if $\mathsf{aux}_{acc} = \bot$, $\mathsf{dig}_{comp} = \bot$ or there is already an account with the same aux_{acc} or dig_{comp}, then abort, else
  - store $(\texttt{accountCreated}, \mathsf{PA}, \mathbf{P}_{comp}, \mathsf{dig}_{comp}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc})$,
  - set $\mathsf{Count}(\mathsf{aux}_{id}) \leftarrow \mathsf{Count}(\mathsf{aux}_{id}) + 1$,
  - add $(\mathsf{aux}_{acc}, \mathsf{dig}_{comp}, \mathsf{pk}_{acc}, \mathsf{I}, \mathsf{PA}, \mathsf{R})$ to the buffer and the adversary;
- Return $(\texttt{accountCreatedShielded}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$ to U and R.

Account and transaction retrieval On $(\texttt{retrieve})$ from a party $\mathsf{P} \in \mathrm{U} \cup \mathrm{AR} \cup \mathrm{I} \cup \mathsf{O} \cup \mathrm{PA} \cup \mathrm{R}$ output a list of all accounts and transactions stored in list_{tx}.

Sending transaction On input $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \mathsf{R})$ from user U:

- If there is no $(\texttt{accountCreated}, \mathsf{aux}_{id}, \mathsf{PA}', \mathbf{P}_{comp}, \mathsf{aux}'_{pol}, \mathsf{dig}'_{comp}, \mathsf{aux}'_{acc}, \mathsf{sk}'_{acc}, \mathsf{U})$ such that $\mathsf{aux}_{acc} = \mathsf{aux}'_{acc}$, $\mathsf{sk}_{acc} = \mathsf{sk}'_{acc}$, $\mathsf{PA} = \mathsf{PA}'$, $\mathsf{aux}_{pol} = \mathsf{aux}'_{pol}$ and $\mathsf{dig}_{comp} = \mathsf{dig}'_{comp}$, then abort.
- If $\mathbf{R}_{val}((\mathsf{tx}, \mathsf{paux}_{tx}), \mathsf{saux}_{tx}) \neq 1$ then abort
- Else read $\mathbf{P}_{comp}$ and send $(\texttt{sendTransactionBegins}, \mathsf{aux}_{acc}, (\mathsf{tx}, \mathsf{paux}_{tx}))$ to $\mathcal{A}$.
- Verify that $\mathbf{P}_{comp}(\mathsf{tx}, \mathsf{paux}_{tx}) = 1$ and if that is not the case, send $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx})$ to PA and $\mathcal{A}$. / We allow the adversary to learn when a user tries to send non-complying transaction.
  - On $(\texttt{allowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ from PA send $(\texttt{allowed})$ to $\mathcal{A}$ and U, then proceed;
  - On $(\texttt{notAllowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ from PA send $(\texttt{notAllowed})$ to $\mathcal{A}$ and U and exit.
- Send $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \mathsf{R})$ to $\mathcal{A}$;
- Return $(\texttt{transactionSent})$ to $\mathcal{A}$ and add $(\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp})$ to the buffer.

Shielded sending transaction On input $(\texttt{sendTransactionShielded}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \mathsf{R})$ from user U and $(\texttt{sendTransactionShielded}, \mathsf{tx})$ from R:

- If there is no $(\texttt{accountCreated}, \mathsf{aux}_{id}, \mathsf{PA}', \mathbf{P}_{comp}, \mathsf{aux}'_{pol}, \mathsf{dig}'_{comp}, \mathsf{aux}'_{acc}, \mathsf{sk}'_{acc}, \mathsf{U})$ such that $\mathsf{aux}_{acc} = \mathsf{aux}'_{acc}$, $\mathsf{sk}_{acc} = \mathsf{sk}'_{acc}$, $\mathsf{PA} = \mathsf{PA}'$, $\mathsf{aux}_{pol} = \mathsf{aux}'_{pol}$ and $\mathsf{dig}_{comp} = \mathsf{dig}'_{comp}$, then abort.
- If $\mathbf{R}_{val}((\mathsf{tx}, \mathsf{paux}_{tx}), \mathsf{saux}_{tx}) \neq 1$ then abort
- Else read $\mathbf{P}_{comp}$ and send $(\texttt{sendTransactionShieldedBegins}, \mathsf{aux}_{acc}, (\mathsf{tx}, \mathsf{paux}_{tx}))$ to $\mathcal{A}$.
- Verify that $\mathbf{P}_{comp}(\mathsf{tx}, \mathsf{paux}_{tx}) = 1$ and if that is not the case, send $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{R})$ to PA and $\mathcal{A}$. / We allow the adversary to learn when a user tries to send non-complying transaction.
  - On $(\texttt{allowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ from PA send $(\texttt{allowed})$ to $\mathcal{A}$ and U, then proceed;
  - On $(\texttt{notAllowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ from PA send $(\texttt{notAllowed})$ to $\mathcal{A}$ and U and exit.
- Send $(\texttt{sendTransactionShielded}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \mathsf{R})$ to $\mathcal{A}$ and R;
- Return $(\texttt{shieldedTransactionSent})$ to $\mathcal{A}$ and R and add $(\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp})$ to the buffer.

Buffer release / Publishes the newly created transactions
On $(\texttt{releaseTransBuffer}, f)$ from $\mathcal{A}$:

- If f is not a permutation or its range size does not equal the number of tuples in the buffer, abort;
- remove all tuples from the buffer and add them in the permuted order (according to f) to the list list_{tx}.

Revoking accounts On $(\texttt{revokeAccount}, \mathsf{aux}_{acc})$ from an identity provider I and a set of anonymity revokers $\{\mathsf{AR}_i\}_{i \in I}$:

- If there is no tuple $(\texttt{accountCreated}, \cdot, \cdot, \mathsf{aux}_{acc}, \cdot)$ then abort. Otherwise read $(\texttt{accountCreated}, \mathsf{aux}_{id}, \mathsf{aux}_{pol}, \mathsf{aux}_{acc}, \mathsf{sk}_{acc})$.
- If $|I| \leq \mathsf{t}$ or there is no tuple $(\texttt{identityIssued}, \mathsf{U}, \mathsf{I}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{aux}_{pol})$ abort;
- Otherwise return $(\texttt{accountRevoked}, \mathsf{aux}_{acc}, \mathsf{U})$ to I and $\{\mathsf{AR}_i\}_{i \in I}$.

Tracing malicious accounts On $(\texttt{traceAccounts}, \mathsf{U})$ from I and $\{\mathsf{AR}_i\}_{i \in I}$:

- If U has no identity registered using I, i.e. there is no tuple $\texttt{identityIssued}, \mathsf{U}, \mathsf{I}, \cdot, \cdot, \cdot, \cdot)$, then abort. Otherwise read the value aux_{id} from the tuple.
- If $|I| > \mathsf{t}$, then return a list of all aux_{acc}-s that there is a tuple $(\texttt{accountCreated}, \mathsf{aux}_{id}, \cdot, \mathsf{aux}_{acc}, \cdot)$.

Application protocol $\mathbf{P}_{\mathsf{PPCTS}}$

Used ideal functionalities

$\mathbf{F}_{pp}$ outputs the public parameters for the signature scheme and the threshold encryption scheme.

$\mathbf{F}_{reg}$ which provides the users with public keys of anonymity revokers, policy assigners and other parties that take part in the protocol.

$\mathbf{F}_{ledger}$ implements the following validate predicate: the predicate accepts if a NIZK proof π is valid and if aux_{acc} has not been seen before.

$\mathbf{F}_{idpol}$ is responsible for issuing users' identities. The functionality is parametrized by

- Signature scheme $\mathsf{S} = (\mathsf{SGen}, \mathsf{KGen}, , \mathsf{Vf})$
- Blind signature scheme $\mathsf{BS} = (\mathsf{SGen}, \mathsf{KGen}, \mathsf{BlindSign}, , \mathsf{Vf})$;
- $(\mathsf{t}, \mathsf{ar})$-threshold encryption scheme $\mathsf{TE} = (\mathsf{SGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$
- Encryption scheme $\mathsf{E} = (\mathsf{SGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$
- Relation $\mathbf{R}_{sig}(\mathsf{pk}, \mathsf{sk})$ which verifies whether $(\mathsf{pk}, \mathsf{sk}) \in \mathrm{Im}(\mathsf{S.KGen})$.

$\mathbf{F}_{nizk}$ NIZK functionalities parametrized by relations (each relation parametrizes different instance of the functionality):
- set of relations $\{\mathbf{R}_{al}\}$, each of $\mathbf{R}_{al}$ tells whether an attribute list of the user fulfils a predicate required by the policy assigner;
- $\mathbf{R}_{val}$ which tells whether a transaction is valid according to the transacting system;
- $\mathbf{R}_{comp}$ which tells whether a policy digest has been correctly computed for the compliance policy assigned to the user and that the transaction complies with the assigned compliance policy or the policy assigner approved the transaction by signing it.
- $\mathbf{R}_{newacc}$ which tells whether account has been created using correct user's secrets and the policy digest has been computed correctly.

Setup

- On input $(\texttt{initialize}, \ldots)$ from any party $\mathsf{P} \in \mathrm{I} \cup \mathrm{AR} \cup \mathrm{U} \cup \mathsf{O} \cup \mathrm{R} \cup \mathrm{PA}$
  - Generate public parameters by running $\mathbf{F}_{pp}^{\mathsf{S},\mathsf{BS},\mathsf{E},\mathsf{TE},\lambda}$ and obtain parameters p.

The protocol description for a user U.

Identity issuance On input $(\texttt{issueIdentity}, \mathsf{I}, \mathsf{AL})$
- Retrieve the identity provider's public key pk_I and anonymity revokers' public keys $\{\mathsf{pk}_{\mathsf{AR}_i}\}_{i=1}^{\mathsf{ar}}$ using the $\mathbf{F}_{reg}$ functionality.
- Generate key pair $(\mathsf{pcred}, \mathsf{scred})$ satisfying $\mathbf{R}_{user}$.
- Call $\mathbf{F}_{idpol}$ on $(\texttt{initialize}, (\mathsf{pcred}, \mathsf{scred}), \mathsf{AL})$;
- Choose a random PRF key k and encrypt it $\mathsf{c}_{key} = \mathsf{TE.Enc}((\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_\mathsf{ar}}), \mathsf{k}; r_{key})$.
- Call identity issuance functionality $\mathbf{F}_{idpol}$ on input $(\texttt{issueIdentity}, (\mathsf{c}_{key}, \mathsf{k}, r_{key}, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_\mathsf{ar}}), \mathsf{AL}, (\mathsf{pcred}, \mathsf{scred}), \mathsf{I})$
- After receiving the response σ_{id} from $\mathbf{F}_{idpol}$ set $\mathsf{aux}_{id} = (\mathsf{I}, \mathsf{AL}, \mathsf{scred}, \sigma_{id}, \mathsf{k})$.
- Return $(\texttt{identityIssued}, \mathsf{aux}_{id})$.

Attribute check On input $(\texttt{checkAttributes}, \mathsf{PA})$
- Call $\mathbf{F}_{idpol}$ on $(\texttt{checkAttributes}, \mathsf{PA})$.
- On $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ return the message to U and abort.
- On $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ store $\mathbf{P}_{al}$ and return $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ to U.

Policy issuance On input $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{AL})$
- send $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{AL})$ to the functionality $\mathbf{F}_{idpol}$;
- obtain policy $\mathbf{P}_{comp}$ and its signature σ_{comp};
- store $(\mathsf{pcred}, \mathsf{PA}, \mathbf{P}_{comp}, \sigma_{comp})$;
- return $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al}, \mathbf{P}_{comp})$ to U.

Account creation On input $(\texttt{createAccount}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc})$
- Compute threshold encryption of the public credentials, i.e. compute $\mathsf{c}_{pcred} \leftarrow \mathsf{TE.Enc}((\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_\mathsf{ar}}), \mathsf{pcred}; r_{pcred})$.
- Compute account identifier $\mathsf{aux}_{acc} \leftarrow \mathsf{PRF}_\mathsf{k}(x)$, for x being the index of the newly created account.
- Compute policy digest $\mathsf{dig}_{comp} = \tilde{\mathsf{PRF}}_\mathsf{k}(\mathbf{P}_{comp}, \sigma_{comp}, r_{pd})$ for some random r_{pd}.
- Obtain PA's public key pk_PA using $\mathbf{F}_{reg}$.
- Make a proof that the values above have been computed correctly, i.e. run $\mathbf{F}_{nizk}^{\mathbf{R}_{newacc}}$ on input $(\texttt{proveStatement}, (\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_\mathsf{PA}, \mathsf{pk}_\mathsf{I}\mathsf{dig}_{comp}, \mathbf{pk}_\mathbf{AR}), (\sigma_{id}, r_{tenc}, \mathsf{pcred}, \mathsf{scred}, \mathsf{k}, \mathsf{sk}_{acc}, \mathbf{P}_{comp}, \sigma_{comp}, r_{pd}, \mathsf{AL}, x$ and obtain a proof π_{newacc}.
- Let $\mathsf{ACI} = ((\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}), \pi_{newacc})$, send $(\texttt{createAccount}, \mathsf{ACI})$ to $\mathbf{F}_{ledger}$
- Return $(\texttt{accountCreated}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$.

Shielded account creation On input $(\texttt{createAccountShielded}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc}, \mathsf{R})$
- Compute threshold encryption of the public credentials, i.e. compute $\mathsf{c}_{pcred} \leftarrow \mathsf{TE.Enc}((\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_\mathsf{ar}}), \mathsf{pcred}; r_{pcred})$.
- Compute account identifier $\mathsf{aux}_{acc} \leftarrow \mathsf{PRF}_\mathsf{k}(x)$, for x being the index of the newly created account.
- Compute policy digest $\mathsf{dig}_{comp} = \tilde{\mathsf{PRF}}_\mathsf{k}(\mathbf{P}_{comp}, \sigma_{comp}, r_{pd})$ for some random r_{pd}.
- Obtain PA's public key pk_PA using $\mathbf{F}_{reg}$.
- Make a proof that the values above have been computed correctly, i.e. run $\mathbf{F}_{nizk}^{\mathbf{R}_{newacc}}$ on input $(\texttt{proveStatement}, (\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_\mathsf{PA}, \mathsf{pk}_\mathsf{I}\mathsf{dig}_{comp}, \mathbf{pk}_\mathbf{AR}), (\sigma_{id}, r_{tenc}, \mathsf{pcred}, \mathsf{scred}, \mathsf{k}, \mathsf{sk}_{acc}, \mathbf{P}_{comp}, \sigma_{comp}, r_{pd}, \mathsf{AL}, x$ and obtain a proof π_{newacc}.
- Let $\mathsf{ACI} = ((\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}), \pi_{newacc})$, send $(\texttt{createAccount}, \mathsf{ACI})$ to the relaying party R via $\mathbf{F}_{amt}$.
- Send $(\texttt{retrieve})$ to $\mathbf{F}_{ledger}$ and obtain ledger L.
- If L contains ACI, store $(\mathsf{ACI}, \mathsf{sk}_{acc})$ and return $(\texttt{accountCreatedShielded}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$ else abort.

Account and transaction retrieval On input $(\texttt{retrieve})$ call $\mathbf{F}_{ledger}$ on input $(\texttt{retrieve})$. After receiving $(\texttt{retrieve}, L)$ from the functionality, output L.

Sending transactions On input $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{sk}_{acc}, \mathsf{PA}, \sigma_{comp})$:

- Call $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$ on input $(\texttt{proveStatement}, \mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx}), (\mathsf{saux}_{tx}))$ to show validity of the transaction and obtain proof π_{val}.
- For transactions that does not comply with the assigned compliance policy
  - Call $\mathbf{F}_{smt}$ on $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \pi_{val}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp}, \mathsf{PA}, \mathsf{R})$.
  - On $(\texttt{notAllowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ abort.
  - On $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ continue.
- Call $\mathbf{F}_{nizk}^{\mathbf{R}_{comp}}$ on $(\texttt{proveStatement}, \mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{dig}_{comp}), (\mathbf{P}_{comp}, \sigma_{comp}, \sigma_{tx}, \mathsf{k}, r_{pd}))$ to show that the transaction fulfils the compliance policy $\mathbf{P}_{comp}$ and obtain proof π_{comp}.
- Let $\mathsf{TxData} = (\mathsf{tx}, (\mathsf{x}_{val}, \pi_{val}), (\mathsf{x}_{comp}, \pi_{comp}))$,
- Send $(\texttt{append}, \mathsf{TxData})$ to $\mathbf{F}_{ledger}$.
- Return $(\texttt{transactionSent}, \mathsf{TxData})$.

Sending shielded transactions On input $(\texttt{sendTransactionShielded}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{sk}_{acc}, \mathsf{PA}, \sigma_{comp}, \mathsf{R})$:

- Call $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$ on input $(\texttt{proveStatement}, \mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx}), (\mathsf{saux}_{tx}))$ to show validity of the transaction and obtain proof π_{val}.
- For transactions that does not comply with the assigned compliance policy
  - Call $\mathbf{F}_{smt}$ on $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \pi_{val}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp}, \mathsf{PA}, \mathsf{R})$.
  - On $(\texttt{notAllowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ abort.
  - On $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ continue.
- Call $\mathbf{F}_{nizk}^{\mathbf{R}_{comp}}$ on $(\texttt{proveStatement}, \mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{dig}_{comp}), (\mathbf{P}_{comp}, \sigma_{comp}, \sigma_{tx}, \mathsf{k}, r_{pd}))$ to show that the transaction fulfils the compliance policy $\mathbf{P}_{comp}$ and obtain proof π_{comp}.
- Let $\mathsf{TxData} = (\mathsf{tx}, (\mathsf{x}_{val}, \pi_{val}), (\mathsf{x}_{comp}, \pi_{comp}))$,
- Send $(\texttt{append}, \mathsf{TxData})$ to the relay party R via $\mathbf{F}_{amt}$.
- Send $(\texttt{retrieve})$ to $\mathbf{F}_{ledger}$ and obtain ledger L.
- If L contains TxData and return $(\texttt{transactionSent}, \mathsf{TxData})$, else abort.

The protocol description for relays R

Shielded funding accounts On input $(\texttt{createAccount}, \mathsf{pk}_{acc})$ from a relay R,

- append ACI to the ledger by running $(\texttt{append}, \mathsf{ACI})$ at $\mathbf{F}_{ledger}$
- Return $(\texttt{accountCreated}, \mathsf{aux}_{acc}, \mathsf{dig}_{comp})$

Shielded sending transactions On input $(\texttt{sendTransaction}, \mathsf{tx})$ from a relay R

- send $(\texttt{sendTransaction})$ to U by calling $\mathbf{F}_{smt}$/
- get $(\mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{pk}_{\mathsf{PA}}, \mathbf{P}_{comp}, \pi_{comp}, \pi_{val})$ from the user,
- verify the zero knowledge proofs, i.e. call $\mathbf{F}_{nizk}^{\mathbf{R}_{comp}}$ on $(\texttt{verifyProof}, (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{pk}_{\mathsf{PA}}, \mathbf{P}_{comp}), \pi_{comp})$ and $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$ on $(\texttt{verifyProof}, (\mathsf{tx}, \mathsf{paux}_{tx}), \pi_{val})$.
- if some of the proofs do not verify, abort
- else set $\mathsf{TxData} \leftarrow (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{pk}_{\mathsf{PA}}, \mathbf{P}_{comp}, \pi_{comp}, \pi_{val})$
- send $(\texttt{append}, \mathsf{TxData})$ to the ledger $\mathbf{F}_{ledger}$.
- Return $(\texttt{shieldedTransactionSent})$.

The protocol description for identity providers I-s and anonymity revokers AR-s.

Initialization On input $(\texttt{initialize})$ from party $\mathsf{I} \in \{\mathsf{I}_1, \ldots, \mathsf{I}_i\}$ obtain SRS srs from $\mathbf{F}_{keygen}^{\mathsf{S.KGen},\lambda}$ generate key pair $(\mathsf{sk}_\mathsf{I}, \mathsf{pk}_\mathsf{I}) \leftarrow_\$ \mathsf{S.KGen}(1^\lambda)$ and send $(\texttt{initialize}, (\mathsf{sk}_\mathsf{I}, \mathsf{pk}_\mathsf{I}))$ to $\mathbf{F}_{idpol}$.

On input $(\texttt{initialize})$ from party $\mathsf{AR} \in \{\mathsf{AR}_1, \ldots, \mathsf{AR}_{\mathsf{ar}}\}$, get public parameters p from $\mathbf{F}_{pp}^{\mathsf{GGen},1^\lambda}$, generate key pair $(\mathsf{pk}_\mathsf{AR}, \mathsf{sk}_\mathsf{AR}) \leftarrow_\$ \mathsf{TE.KGen}(1^\lambda)$ by calling $\mathbf{F}_{keygen}^{\mathsf{TE.KGen},\lambda}$ and send $(\texttt{register}, (\mathsf{pk}_\mathsf{AR}, \mathsf{sk}_\mathsf{AR}))$ to $\mathbf{F}_{reg}$.

Identity issuance On input $(\texttt{issueIdentity}, \mathsf{U})$ from an identity provider I, call $\mathbf{F}_{idpol}$ with input $(\texttt{issueIdentity}, \mathsf{U}, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}}))$. Given the response $(\mathsf{pcred}, \mathsf{AL}, \mathsf{c}_{pcred})$ from the functionality, set $\mathsf{IPIU} = (\mathsf{U}, \mathsf{pcred}, \mathsf{AL}, \mathsf{c}_{pcred})$.

Retrieving On input $(\texttt{retrieve})$ call $\mathbf{F}_{ledger}$ on input $(\texttt{retrieve})$. After receiving $(\texttt{retrieve}, L)$ from the functionality output L.

Revoking account On input $(\texttt{revokeAccount}, \mathsf{aux}_{acc})$ from an identity provider I and $\{\mathsf{AR}_i\}_{i \in I}$, for $|I| > \mathsf{t}$ the anonymity revokers proceed as follows:

- Call $\mathbf{F}_{ledger}$ on input $(\texttt{retrieve})$;
- After receiving $(\texttt{retrieve}, L)$ from $\mathbf{F}_{ledger}$, search L for user's dataset ACI that contains aux_{acc}.
- Decrypt c_{pcred} jointly by calling TE.Dec.

– Combine the shares to reveal public credentials of the user pcred.

Identity provider I proceeds as follows:

– Read the identity provider identity I' from the decrypted dataset ACI.
– Ignore if $\mathsf{I} \neq \mathsf{I}'$.
– Else, locate $\mathsf{IPIU} = (\mathsf{U}, \mathsf{aux}_{acc}, \mathsf{AL}, \mathsf{c}_{pcred})$ and return U.

Tracing account On input $(\texttt{trace}, \mathsf{U})$ from an identity provider I and $\{\mathsf{AR}_i\}_{i \in I}$, for $|I| > \mathsf{t}$,

– Identity provider locates $\mathsf{IPIU} = (\mathsf{U}, \mathsf{aux}_{acc}, \mathsf{AL}, \mathsf{c}_{pcred})$ containing U and sends c_{key} to anonymity revokers via $\mathbf{F}_{smt}$.
– Each anonymity revoker decrypts its share of user's key k_i and call $\mathbf{F}_{mpcprf}$ on $(\texttt{compute}, \mathsf{k}_i, \mathsf{pcred})$ and receive all account identifiers that can be generated by the user, i.e. $\mathsf{PRF}_{\mathsf{k}}(x)$, for $x \in [1..\mathsf{maxAccounts}]$.

The protocol description for policy assigners PA**.**

Initialization On input $(\texttt{initialize})$ from a policy assigner $\mathsf{PA} \in \{\mathsf{PA}_1, \ldots, \mathsf{PA}_{\mathsf{pa}}\}$ obtain public parameters p from $\mathbf{F}_{pp}^{\mathsf{GGen},\lambda}$, generate key pairs $(\mathsf{S.pk}_{\mathsf{PA}}, \mathsf{S.sk}_{\mathsf{PA}}) \leftarrow\!\!\$\ \mathsf{S.KGen}(1^\lambda)$, $(\mathsf{E.pk}_{\mathsf{PA}}, \mathsf{E.sk}_{\mathsf{PA}}) \leftarrow\!\!\$\ \mathsf{E.KGen}(1^\lambda)$ and send $(\texttt{initialize}, (\mathsf{S.pk}_{\mathsf{PA}}, \mathsf{S.sk}_{\mathsf{PA}}), (\mathsf{E.pk}_{\mathsf{PA}}, \mathsf{E.sk}_{\mathsf{PA}}))$ to $\mathbf{F}_{idpol}$.

Attributes check On input $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ from PA call $\mathbf{F}_{idpol}$ on $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ and return its output.

Policy issuance On input $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp})$ from user policy assigner PA call functionality $\mathbf{F}_{idpol}$ on input $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp})$ and get $(\texttt{policyAssigned}, \mathbf{P}_{comp})$

Conditional transaction approval: On input $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp}, \mathsf{PA})$ from user U, reply via $\mathbf{F}_{smt}$ with either $(\texttt{allowed}, \sigma_{tx}, (\mathsf{tx}, \mathsf{paux}_{tx}))$, where $\sigma_{tx} = \mathsf{S.}(\mathsf{sk}_{\mathsf{PA}}, (\mathsf{tx}, \mathsf{paux}_{tx}))$, or $(\texttt{notAllowed}, (\mathsf{tx}, \mathsf{paux}_{tx}))$.

5.2.1 Security proof and the simulator description

Simulator $\mathsf{Sim}_{\mathsf{PPCTS}}$

The simulator internally emulates all functionalities that are specified in the protocol $\mathbf{P}_{\mathsf{PPCTS}}$.

Initialization

1. The simulator initiates all functionalities it simulates and generates all necessary keys and parameters they use. That is, it runs $\mathbf{F}_{pp}$ to generate public parameters.
2. Honest user U. On $(\texttt{initialized}, \mathsf{U})$ from the functionality:
   – Add U to the set of initialized parties Init.
   – Call $\mathbf{F}_{idpol}$ on $(\texttt{initialize})$.
3. Dishonest user U. On any first message from an uninitialised dishonest user U
   – Call $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{initialize})$ on behalf of U.
   – Add U to Init.
4. Honest policy assigner PA. On $(\texttt{initialized}, \mathsf{PA})$ from the functionality:
   – Pick randomly $(\mathsf{S.pk}, \mathsf{S.sk}) \in \mathbf{R}_{sig}$ and $(\mathsf{E.pk}, \mathsf{E.sk}) \in \mathbf{R}_{enc}$.
   – Call $\mathbf{F}_{idpol}$ on $(\texttt{initialize}, (\mathsf{S.pk}, \mathsf{S.sk}), (\mathsf{E.pk}, \mathsf{E.sk}))$.
   – Add $((\mathsf{S.pk}, \mathsf{S.sk}), (\mathsf{E.pk}, \mathsf{E.sk}))$ to list $\mathsf{list}_{polas,pk}$ and PA to Init
   – Register $((\mathsf{S.pk}, \mathsf{S.sk}), (\mathsf{E.pk}, \mathsf{E.sk}))$ at $\mathbf{F}_{reg}$
5. Dishonest policy assigner PA. On $(\texttt{initialize}, (\mathsf{S.pk}, \mathsf{S.sk}), (\mathsf{E.pk}, \mathsf{E.sk}))$ from a dishonest user PA sent to $\mathbf{F}_{idpol}$
   – Check that $\mathbf{R}_{sig}(\mathsf{S.pk}, \mathsf{S.sk}) = 1$ and $\mathbf{R}_{enc}(\mathsf{E.pk}, \mathsf{E.sk}) = 1$, abort if that is not the case.
   – Output fail if $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{list}_{polas,pk}$. / The adversary broken hardness of relation $\mathbf{R}_{sig}$.
   – Else store $(\mathsf{pk}, \mathsf{sk})$.
   – Sent $(\texttt{initialize})$ to $\mathbf{F}_{\mathsf{PPCTS}}$ on behalf of PA and add the party to Init.
6. Honest identity provider I. On $(\texttt{initialized}, \mathsf{I})$ from the functionality:
   – Pick randomly $(\mathsf{pk}, \mathsf{sk}) \in \mathbf{R}_{sig}$.
   – Call $\mathbf{F}_{idpol}$ on $(\texttt{initialize}, (\mathsf{pk}, \mathsf{sk}))$.
   – Add $(\mathsf{pk}, \mathsf{sk})$ to $\mathsf{list}_{idprov,pk}$ and I to Init.
   – Register $(\mathsf{pk}, \mathsf{sk})$ at $\mathbf{F}_{reg}$
7. Dishonest identity provider I. On $(\texttt{initialize}, (\mathsf{pk}, \mathsf{sk}))$ sent to $\mathbf{F}_{idpol}$ by a dishonest user I:
   – Check that $\mathbf{R}_{sig}(\mathsf{pk}, \mathsf{sk}) = 1$, and abort if that is not the case.
   – If $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{list}_{idprov,pk}$ output fail. / The adversary broken hardness of relation $\mathbf{R}_{sig}$.
   – Send $(\texttt{initialize})$ to $\mathbf{F}_{\mathsf{PPCTS}}$ on behalf of I and add the party to Init
8. Honest anonymity revoker AR. On $(\texttt{initialized}, \mathsf{AR})$ sent to the functionality
   – Pick $(\mathsf{pk}, \mathsf{sk}) \in \mathbf{R}_{tenc}$.

- Send $(\mathsf{pk}, \mathsf{sk})$ to $\mathbf{F}_{reg}$.
- Call $\mathbf{F}_{idpol}$ on $(\texttt{initialize}, (\mathsf{pk}, \mathsf{sk}))$.

9. Dishonest anonymity revoker AR. On $(\texttt{initialize}, (\mathsf{pk}, \mathsf{sk}))$ from the functionality $\mathbf{F}_{idpol}$
   - Check that $\mathbf{R}_{tenc}(\mathsf{pk}, \mathsf{sk}) = 1$ and abort if that is not the case.
   - Call $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{initialize})$ and add the party to the set of initialized parties Init3.

Importantly, if simulator gets a message from a non-initialized party, it ignores it (except the message is sent to initialize the party).

Identity issuance [**Michal 13.12:** Should we add case of both honest parties?] [**Michal 5.1:** No if the functionality doesn't leak anything to the adversary]

1. Passively corrupted I, honest U. On $(\texttt{issueIdentity}, \mathsf{U}, \mathsf{I}, \mathsf{AL})$ from $\mathbf{F}_{\mathsf{PPCTS}}$,
   - Pick randomly $(\mathsf{pcred}, \mathsf{scred})$ accordingly to $\mathbf{R}_{user}$ and compute c_{key} being a dummy encryption of 0.
   - Emulate $\mathbf{F}_{idpol}$ for I and output $(\mathsf{pcred}, \mathsf{c}_{key})$ to I. [**Michal 12.1:** Should we output AL to the identity provider here?]
   - Add c_{key} to list_{ckey}, pcred to $\mathsf{list}_{user,pcred}$, and $((\mathsf{U}, \mathsf{AL}, \mathsf{I}), (\mathsf{c}_{key}, \mathsf{pcred}, \mathsf{AL})$ to list_{issue}.
2. Malicious U, honest I. On corrupted U's call $(\texttt{issueIdentity}, (\mathsf{c}_{key}, \mathsf{k}, r, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})), \mathsf{AL}, (\mathsf{pcred}, \mathsf{scred}), \mathsf{I})$.
   - Abort if c_{key} is not a valid encryption of key k under public keys $(\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})$ and randomness r; or $\mathbf{R}_{user}(\mathsf{pcred}, \mathsf{scred}) \neq 1$.
   - Output fail if the ciphertext or the public key have been already registered, i.e. when $\mathsf{c}_{key} \in \mathsf{list}_{ckey}$ or $\mathsf{pcred} \in \mathsf{list}_{user,pcred}$. / That is, the adversary produced a ciphertext identical to a ciphertext produced by the simulator, breaking security of the encryption scheme; or it produced the same public key pcred breaking hardness of relation $\mathbf{R}_{user}$.
   - Otherwise, call $\mathbf{F}_{\mathsf{PPCTS}}$ with $(\texttt{issueIdentity}, \mathsf{I}, \mathsf{AL})$, compute the signature σ_{id} by internally emulating the honest I. Return $\mathsf{aux}_{id} = (\mathsf{I}, \mathsf{AL}, \mathsf{scred}, \sigma_{id}, \mathsf{k})$ to U.
   - Add $(\sigma_{id}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL}), \mathsf{I})$ to list_{sign}.

Attribute check

1. Honest user U and policy assigner PA. / The simulator does not need to do anything as this procedure does not leak anything to the adversary.
2. Honest user U and malicious policy assigner PA. On $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ sent by PA to $\mathbf{F}_{idpol}$
   - call $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ on behalf of PA;
   - call $\mathbf{F}_{idpol}$ on $(\texttt{checkAttributes}, \mathsf{PA})$ on behalf of the user.
3. Malicious user U and honest policy assigner PA. On $(\texttt{checkAttributes}, \mathsf{PA})$ sent by U to $\mathbf{F}_{idpol}$
   - call $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{checkAttributes}, \mathsf{PA})$ on behalf of U;
   - on $(\texttt{attributesNotValid}, \mathbf{P}_{al})$ or $(\texttt{attributesValid}, \mathsf{U}, \mathbf{P}_{al})$ from $\mathbf{F}_{\mathsf{PPCTS}}$ call $\mathbf{F}_{idpol}$ on $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ on behalf of PA
4. Malicious U and policy assigner PA. We do not allow both the user and policy assigner to be corrupted as that would allow the user to get any policy on any attribute list it wants.

Policy issuance

1. Passively corrupted PA, honest U. On PA's call $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp}, \mathbf{P}_{al})$ to the ideal functionality $\mathbf{F}_{idpol}$
   - Send $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{al}, \mathbf{P}_{comp})$ to $\mathbf{F}_{\mathsf{PPCTS}}$.
   - Simulate the $\mathbf{F}_{idpol}$ functionality for PA and, for the policy $\mathbf{P}_{comp}$ provided by PA, output $(\texttt{policyAssigned})$ to PA.
   - Store $(\mathsf{U}, \mathsf{PA}, \mathbf{P}_{comp})$ to list_{pol}
2. Malicious U, honest PA. On corrupted U's call $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{AL})$ to $\mathbf{F}_{idpol}$
   - Invoke $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{AL})$.
   - Learn the policy $\mathbf{P}_{comp}$ sent to U by the policy assigner.
   - Compute signature σ_{comp} on $(\mathbf{P}_{comp}, \mathsf{pcred})$ using previously picked signature scheme keys.
   - Output $(\texttt{policyAssigned}, \mathbf{P}_{comp}, \sigma_{comp})$ to U.

Shielded account creation [a]

1. Malicious U, honest relay R. On $(\texttt{sendAnonymously}, \mathsf{R}, \mathsf{ACI})$ sent to $\mathbf{F}_{amt}$, where $\mathsf{ACI} = (\mathsf{x}_{newacc} = (\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{pk}_{\mathsf{I}}, \mathsf{dig}_{comp}, \mathbf{pk}_{\mathbf{AR}}), \pi_{newacc})$,
   - Use $\mathbf{F}_{nizk}^{\mathbf{R}_{newacc}}$ to extract the witness $\mathsf{w}_{newacc} = (\sigma_{id}, r_{tenc}, \mathsf{pcred}, \mathsf{scred}, \mathsf{k}, \mathsf{sk}_{acc}, \mathbf{P}_{comp}, \sigma_{comp}, r_{pd}, \mathsf{AL}, x, \mathbf{P}_{al}, \sigma_{al})$ for x_{newacc} or abort if the proof does not verify or there already is registered account $(\mathsf{aux}_{acc}, \mathsf{c}_{pcred}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{pk}_{\mathsf{I}}, \mathsf{dig}_{comp}, \mathbf{pk}_{\mathbf{AR}}, \pi) \in \mathsf{list}_{acc}$.
   - Output fail if at least one of the following holds:
     - $(\sigma_{id}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL}), \mathsf{I}) \notin \mathsf{list}_{sig}$, **/** The adversary broke existential unforgeability of the signature scheme.
     - $\mathsf{pcred} \in \mathsf{list}_{user,pcred}$ **/** The adversary broke hardness of relation $\mathbf{R}_{user}$.
     - $\mathsf{c}_{pcred} \in \mathsf{list}_{cuser}$, **/** The adversary broke semantic security of the encryption scheme.
     - $\mathsf{aux}_{acc} \in \mathsf{list}_{auxacc}$, **/** The adversary broke robustness of the PRF.
     - $x > \mathsf{maxAccounts}$. **/** The adversary broke soundness of NIZK for $\mathbf{R}_{newacc}$.
   - Otherwise, call $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{createAccount}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{aux}_{pol}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc}, \mathsf{R})$ and, when prompted, input $\mathsf{aux}_{acc} = \mathsf{PRF}_{\mathsf{k}}(x)$.
   - Send $(\texttt{append}, \mathsf{ACI})$ to the ledger functionality $\mathbf{F}_{ledger}$.
2. Honest U and relay R. Upon receiving $(\texttt{createAccount}, \mathsf{pk}_{acc}, \mathsf{I}, \mathsf{PA})$ from the ideal functionality:
   - Pick a random aux_{acc} from the PRF domain and forward it to the functionality.
   - Add aux_{acc} into list_{auxacc}, else abort.
   - Prepare a statement $\mathsf{x}_{newacc} = (\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{dig}_{comp}, \mathbf{pk}_{\mathbf{AR}})$ where c_{key} is an encryption of dummy.
   - Get a simulated proof π_{newacc} for x_{newacc} by calling $\mathbf{F}_{nizk}^{\mathbf{R}_{newacc}}$ and append (x, π) to the buffer of $\mathbf{F}_{ledger}$.
   - Add $(\mathsf{x}_{newacc}, \pi_{newacc})$ to list_{acc}.
3. Honest U, malicious relay R. On $(\texttt{createAccount}, \mathsf{pk}_{acc}, \mathsf{I}, \mathsf{PA})$ from the ideal functionality,
   - Pick a random aux_{acc} from the PRF domain and forward it to the functionality.
   - Add aux_{acc} into list_{auxacc}.
   - Prepare statement $\mathsf{x}_{newacc} = (\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{pk}_{\mathsf{I}}, \mathsf{dig}_{comp}, \mathbf{pk}_{\mathbf{AR}})$ where c_{key} is an encryption of dummy.
   - Get a simulated proof π_{newacc} for x_{newacc} by calling $\mathbf{F}_{nizk}^{\mathbf{R}_{newacc}}$ and append (x, π) to the buffer of $\mathbf{F}_{ledger}$.
   - Add $(\mathsf{x}_{newacc}, \pi_{newacc})$ to list_{acc}.
4. Malicious U and relay R. On $(\texttt{sendAnonymously}, \mathsf{R}, \mathsf{ACI})$, where $\mathsf{ACI} = (\mathsf{x}_{newacc} = (\mathsf{c}_{pcred}, \mathsf{aux}_{acc}, \mathsf{I}, \mathsf{pk}_{acc}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{pk}_{\mathsf{I}}, \mathsf{dig}_{comp}, \mathbf{pk}_{\mathbf{AR}}), \pi_{newacc})$, send by U to $\mathbf{F}_{amt}$:
   - Use $\mathbf{F}_{nizk}^{\mathbf{R}_{newacc}}$ to extract the witness $\mathsf{w}_{newacc} = (\sigma_{id}, r_{tenc}, \mathsf{pcred}, \mathsf{scred}, \mathsf{k}, \mathsf{sk}_{acc}, \mathbf{P}_{comp}, \sigma_{comp}, r_{pd}, \mathsf{AL}, x, \mathbf{P}_{al}, \sigma_{al})$ for x_{newacc} or abort if the proof does not verify or there already is registered account $(\mathsf{x}_{newacc}, \pi_{newacc}) \in \mathsf{list}_{acc}$.
   - Output fail if at least one of the following holds:
     - $(\sigma_{id}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL}), \mathsf{I}) \notin \mathsf{list}_{sig}$,
     - $\mathsf{pcred} \in \mathsf{list}_{user,pcred}$,
     - $\mathsf{c}_{pcred} \in \mathsf{list}_{cuser}$,
     - $\mathsf{aux}_{acc} \in \mathsf{list}_{auxacc}$,
     - $\mathsf{aux}_{acc} \in \mathsf{list}_{auxacc}$, or
     - $x > \mathsf{maxAccounts}$.
   - Otherwise, call $\mathbf{F}_{\mathsf{PPCTS}}$ on $(\texttt{createAccount}, \mathsf{aux}_{id}, \mathsf{PA}, \mathsf{aux}_{pol}, \mathsf{pk}_{acc}, \mathsf{sk}_{acc}, \mathsf{R})$ and, when asked, input $\mathsf{aux}_{acc} = \mathsf{PRF}_{\mathsf{k}}(x)$.
   - On $\mathbf{F}_{ledger}$ call $(\texttt{append}, \mathsf{ACI})$ from R

Release Emulate the command by simulating calls to $\mathbf{F}_{ledger}$. That is, when the adversary invokes $(\texttt{releaseTransactionBuffer}, f)$ add the permuted buffer to the list list_{ledger} and then reset the buffer.

Retrieving Simulate the retrieve command in $\mathbf{F}_{ledger}$ and give as output list_{ledger}.

Shielded sending transactions [b]

1. Honest U, relay R and policy assigner PA. Upon receiving $(\texttt{sendTransactionBegins}, \mathsf{aux}_{acc}, (\mathsf{tx}, \mathsf{paux}_{tx}))$
   - On $(\texttt{allowTransaction}, \ldots, \mathsf{PA}, \ldots)$
     - Send to PA message $(\texttt{allowTransaction}, \ldots)$ via $\mathbf{F}_{smt}$ on behalf of U.
     - On $(\texttt{allowed})$,
       - Make a simulated signature σ_{tx} on $(\mathsf{tx}, \mathsf{paux}_{tx})$ on behalf of PA,
       - Send to U on behalf of PA message $(\texttt{allowed}, \mathsf{tx}, \mathsf{paux}_{tx}, \sigma_{tx})$ via $\mathbf{F}_{smt}$. **/** PA allowed transaction.
     - On $(\texttt{notAllowed})$, send $(\texttt{notAllowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ to U via $\mathbf{F}_{smt}$ on behalf of PA and exit.
   - On $(\texttt{sendTransaction}, \ldots)$ from the functionality,

- Prepare simulated zero-knowledge proof for transaction validity, $\mathbf{R}_{val}$ and compliance fulfillment $\mathbf{R}_{comp}$. That is, show $\mathsf{x}_{val} \in \mathbf{L}_{\mathbf{R}_{val}}$ and $\mathsf{x}_{comp} \in \mathbf{L}_{\mathbf{R}_{comp}}$.
- Prepare $\mathsf{TxData} = (\mathsf{tx}, (\mathsf{x}_{val}, \pi_{val}), (\mathsf{x}_{comp}, \pi_{comp}))$.
- Send $(\texttt{sendTransaction}, \mathsf{TxData})$ to R via $\mathbf{F}_{amt}$.
- Send $(\texttt{append}, \mathsf{TxData})$ to $\mathbf{F}_{ledger}$

2. Malicious U, honest relay R and policy assigner PA.
   - On U using $\mathbf{F}_{smt}$ to send to PA message $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \pi_{val}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp}, \mathsf{PA}, \mathsf{R})$ / The user's transaction does not follow the compliance policy and it has to ask the policy assigner for the permission.
     - Abort if π_{val} is not acceptable;
     - Else, extract the witness saux_{tx} from $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$, output fail if this fails.
     - Call the functionality on $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \sigma_{comp}, \mathsf{dig}_{comp}, \mathsf{R})$ on behalf of U.
     - On $(\texttt{notAllowed})$ from the functionality, send to U on behalf of PA message $(\texttt{notAllowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ via $\mathbf{F}_{smt}$. / PA did not allowed the transaction.
     - On $(\texttt{allowed})$ from the functionality,
       * Make a simulated signature σ_{tx} on $(\mathsf{tx}, \mathsf{paux}_{tx})$ on behalf of PA,
       * Send to U on behalf of PA message $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ via $\mathbf{F}_{smt}$. / PA allowed transaction.
   - On input $(\texttt{sendAnonymously}, \mathsf{R}, \mathsf{TxData})$ sent to $\mathbf{F}_{amt}$ by U, read from TxData: aux_{acc}, $\mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx})$, $\mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{PA}, \mathsf{dig}_{comp})$ / User does not ask the policy assigner for a permission but directly goes to the relay
     - Abort if the provided proofs π_{val} or π_{comp} do not verify.
     - Use $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$ and $\mathbf{F}_{nizk}^{\mathbf{R}_{comp}}$ to extract witnesses $\mathsf{w}_{val} = (\mathsf{saux}_{tx})$ and $\mathsf{w}_{comp} = (\mathbf{P}_{comp}, \sigma_{comp}, \sigma_{tx}, \mathsf{k}, r)$ and output fail is this fails.
     - Output fail if σ_{comp} not in the list list_{pol} or σ_{tx} not in the list list_{tx}.
     - Call functionality on $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \sigma_{comp}, \mathsf{R})$ if that has not been already done.
     - Call $\mathbf{F}_{ledger}$ on behalf of the relay with input $(\texttt{append}, \mathsf{TxData})$.
3. Honest U and policy assigner PA, malicious relay R.
   - On $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp}, \mathsf{PA}, \mathsf{R})$ from the functionality,
     - Send $(\texttt{allowTransaction}, \ldots)$ to PA on behalf of the user via $\mathbf{F}_{smt}$.
     - On $(\texttt{notAllowed})$ from the functionality, send $(\texttt{notAllowed}, (\mathsf{tx}, \mathsf{paux}_{tx}))$ to U via $\mathbf{F}_{smt}$ on behalf of PA.
     - On $(\texttt{allowed}, \mathsf{tx}, \mathsf{paux}_{tx})$ from the functionality,
       * Prepare a simulated signature σ_{tx} for $\mathsf{tx}, \mathsf{paux}_{tx}$ on behalf of PA,
       * Send $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ to U via $\mathbf{F}_{smt}$ on behalf of PA.
   - On $(\texttt{sendTransaction}, \ldots)$ from the functionality,
     - Prepare a simulated zero-knowledge proof π_{val} for $\mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx}) \in \mathbf{L}_{\mathbf{R}_{val}}$;
     - Prepare a simulated zero-knowledge proof π_{comp} for $\mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{pk}_{\mathsf{PA}}, \mathsf{aux}_{pol}) \in \mathbf{L}_{\mathbf{R}_{comp}}$;
     - Prepare TxData.
     - Send to R via $\mathbf{F}_{amt}$ message $(\texttt{sendTransaction}, \mathsf{TxData})$.
4. Malicious user U and relay R, honest policy assigner PA.
   - On input $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \pi_{val}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp}, \mathsf{PA}, \mathsf{R})$ sent by U to PA via $\mathbf{F}_{smt}$:
     - Abort if π_{val} is not acceptable.
     - Use functionality $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$ to extract witness saux_{tx} for $(\mathsf{tx}, \mathsf{paux}_{tx})$ and output fail if this fails.
     - Call functionality on $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \sigma_{comp}, \mathsf{R})$.
     - On $(\texttt{notAllowed})$ from the functionality, send $(\texttt{notAllowed}, (\mathsf{tx}, \mathsf{paux}_{tx}))$ to U via $\mathbf{F}_{smt}$ on behalf of PA.
     - On $(\texttt{allowed})$ from the functionality,
       * Prepare a simulated signature σ_{tx} for $\mathsf{tx}, \mathsf{paux}_{tx}$ on behalf of PA,
       * Send $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ to U via $\mathbf{F}_{smt}$ on behalf of PA.
   - On input $(\texttt{sendAnonymously}, \mathsf{R}, \mathsf{TxData})$ sent to $\mathbf{F}_{amt}$ by U, read from TxData: aux_{acc}, $\mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx})$, $\mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{PA}, \mathsf{dig}_{comp})$
     - Abort if the provided proofs π_{val} or π_{comp} do not verify.
     - Use $\mathbf{F}_{nizk}^{\mathbf{R}_{val}}$ and $\mathbf{F}_{nizk}^{\mathbf{R}_{comp}}$ to extract witnesses $\mathsf{w}_{val} = (\mathsf{saux}_{tx})$ and $\mathsf{w}_{comp} = (\mathbf{P}_{comp}, \sigma_{comp}, \sigma_{tx}, \mathsf{k}, r)$, output fail if this fails.
     - Output fail if σ_{comp} not in the list list_{pol} or σ_{tx} not in the list list_{tx}.
     - Otherwise, call functionality $\mathbf{F}_{\mathsf{PPCTS}}$ on input $(\texttt{sendTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{saux}_{tx}, \mathsf{aux}_{acc}, \mathsf{PA}, \mathsf{dig}_{comp}, \sigma_{comp}, \mathsf{R})$ if that has not been done already.
5. Malicious R and policy assigner PA, honest user U.
   - On $(\texttt{allowTransaction}, \ldots)$ from the functionality
     - Send $(\texttt{allowTransaction}, \ldots)$ to PA via $\mathbf{F}_{smt}$;
     - On $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ from PA
       * Verify the signature and if it is correct send $(\texttt{allowed})$ to the functionality;

        * Abort if the signature does not verify correctly;
      - On $(\texttt{notAllowed})$ from PA, send $(\texttt{notAllowed})$ to the functionality and exit.
  - On $(\texttt{sendTransaction}, \ldots)$ from the functionality
    - Prepare a simulated zero-knowledge proof π_{val} for $\mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx}) \in \mathbf{L}_{\mathbf{R}_{val}}$;
    - Prepare a simulated zero-knowledge proof π_{comp} for $\mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{PA}, \mathsf{dig}_{comp}) \in \mathbf{L}_{\mathbf{R}_{comp}}$;
    - Prepare TxData and send $(\texttt{sendTransaction}, \mathsf{TxData})$ via $\mathbf{F}_{smt}$ on behalf of U.
6. Malicious policy assigner PA, honest user U and relay R.
  - On $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp})$ sent by the functionality:
    - Send to PA message $(\texttt{allowTransaction}, \mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{aux}_{acc}, \mathsf{pk}_{acc}, \mathsf{dig}_{comp})$ via $\mathbf{F}_{smt}$ on behalf of U.
    - On $(\texttt{allowed}, \sigma_{tx}, \mathsf{tx}, \mathsf{paux}_{tx})$ sent by PA to U via $\mathbf{F}_{smt}$,
      * Verify the signature and if it is correct send $(\texttt{allowed})$ to the functionality;
      * Abort if the signature does not verify correctly;
    - On $(\texttt{notAllowed}, (\mathsf{tx}, \mathsf{paux}_{tx}))$, send $(\texttt{notAllowed})$ to the functionality and exit.
    - Prepare a simulated zero-knowledge proof π_{val} for $\mathsf{x}_{val} = (\mathsf{tx}, \mathsf{paux}_{tx}) \in \mathbf{L}_{\mathbf{R}_{val}}$;
    - Prepare a simulated zero-knowledge proof π_{comp} for $\mathsf{x}_{comp} = (\mathsf{tx}, \mathsf{paux}_{tx}, \mathsf{PA}, \mathsf{aux}_{pol}) \in \mathbf{L}_{\mathbf{R}_{comp}}$;
    - Collect TxData and send $(\texttt{sendTransaction}, \mathsf{TxData})$ to R via $\mathbf{F}_{amt}$ on behalf of U.

Revoking account

1. Semi-honest I, up to t malicious AR, honest U.
  - When I and a qualified set of anonymity revokers inputs $\texttt{revokeAccount}$, the simulator obtains aux_{acc} and U from the input and output of the functionality $\mathbf{F}_{\mathsf{PPCTS}}$.
  - The simulator, using aux_{acc} searches list_{acc} and retrieves the corresponding c_{key}.
  - Similarly, the simulator uses (U, I), searches list_{issue} and retrieves the corresponding pcred.
  - The simulator equivocates the decryption of c_{key} using pcred.
2. Malicious U and up to t malicious AR-s.
  - The simulator receives U and a list of accounts $\{\mathsf{aux}_{acc}\}$ from the ideal functionality.
  - The simulator looks up the ciphertext c_{pcred} corresponding to the user U and emulates $\mathbf{F}_{mpcprf}$ to output $\{\mathsf{aux}_{acc}\}$.

Tracing account

1. Semi-honest I and up to t semi-honest AR, honest U.
  - When the identity provider and the qualified set of anonymity revokers calls $(\texttt{trace})$, the simulator receives U and a lit list_{auxacc} from the input and output of the main functionality.
  - The simulator recovers c_{pcred} ciphertext.
  - It programs the output of $\mathbf{F}_{mpcprf}$ to be consistent with list_{auxacc}.
2. Malicious user U and up to t malicious anonymity revokers.
  - The simulator receives U and list of accounts list_{auxacc} from the ideal functionality.
  - It looks up the ciphertext c_{pcred} corresponding to the user U and emulates $\mathbf{F}_{mpcprf}$ to output the list list_{auxacc}.

[a] Analogously the simulator proceeds when non-shielded transaction is sent. In that case, interaction with relay is removed.
[b] Analogously the simulator proceeds when non-shielded transaction is sent. In that case, interaction with relay is removed.

Theorem 1. *Let* S *be a existentially unforgeable signature scheme,* PRF *be a PRF,* TE *be a semantically secure threshold encryption scheme which has partial decryption simulatability property. Then the protocol* $\mathbf{P}_{\mathsf{PPCTS}}$ *UC-realizes* $\mathbf{F}_{\mathsf{PPCTS}}$ *functionality in the* $\{\mathbf{F}_{idpol}, \mathbf{F}_{ledger}, \mathbf{F}_{srs}, \mathbf{F}_{pp}, \mathbf{F}_{idpol}, \mathbf{F}_{nizk}, \mathbf{F}_{smt}, \mathbf{F}_{amt}\}$*-hybrid model.*

Proof. The proof goes by game hoping. In the first game, the simulator has access to the ideal functionality $\mathbf{F}_{\mathsf{PPCTS}}$ and provides a view indistinguishable from an execution of a real protocol $\mathbf{P}_{\mathsf{PPCTS}}$. The final game is the real-life protocol.

Hybrid 0. In this hybrid the simulator behaves as described above.
Hybrid 1. In this hybrid the simulator never outputs fail.
Hybrid 0 to Hybrid 1. We argue that the simulator described in the frame above outputs fail with negligible probability only. More precisely, Sim return fail if
- $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{list}_{polas,pk}$ what happens only if the adversary breaks hardness of a hard relation $\mathbf{R}_{sig}$.
- $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{list}_{idprov,pk}$ what happens only if the adversary breaks hardness of a hard relation $\mathbf{R}_{sig}$.
- $\mathsf{pcred} \in \mathsf{list}_{user,pcred}$ what happens only if the adversary breaks hardness of a hard relation $\mathbf{R}_{user}$.
- $\mathsf{c}_{key} \in \mathsf{list}_{ckey}$ what happens only if the adversary breaks semantic security of the encryption scheme.
- $(\sigma_{id}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL}), \mathsf{I}) \notin \mathsf{list}_{sig}$ what happens only if the adversary breaks existential unforgeability of the signature scheme.
- $\mathsf{c}_{pcred} \in \mathsf{list}_{cpcred}$ what happens only if the adversary breaks semantic security of the encryption scheme.

- $\mathsf{aux}_{acc} \in \mathsf{list}_{auxacc}$ what happens only if the adversary breaks robustness of the PRF.
- $x \in \mathsf{maxAccounts}$ what happens only if the adversary breaks soundness of a proof system for $\mathbf{R}_{newacc}$.
- $\sigma_{comp} \in \mathsf{list}_{pol}$ what happens only if the adversary breaks existential unforgeability of the signature scheme.
- $\sigma_{tx} \in \mathsf{list}_{tx}$ what happens only if the adversary breaks existential unforgeability of the signature scheme.
- it fails to extract a witness from an acceptable zero-knowledge proof produced by a malicious party.

All of these events have negligible probability, thus the probability that the hybrids are distinguishable is negligible as well.

Hybrid 2. In this hybrid the simulator picks aux_{acc} in a way compliant with the protocol $\mathbf{P}_{\mathsf{PPCTS}}$. That is, it picks the PRF key k and for a user using credential pcred computes $\mathsf{aux}_{acc} = \mathsf{PRF}_{\mathsf{k}}(x)$ for a counter $x \in [1..\mathsf{maxAccounts}]$.

Hybrid 1 to Hybrid 2. Assume that there is a distinguisher $\mathcal{D}$ who can distinguish both hybrids. That distinguisher can be used to distinguish an output of a PRF PRF and a random bitstring. Hence, it can be used to break pseudorandomness of PRF.

Hybrid 3. In this hybrid the simulator creates ciphertext c_{pcred} as in the real-life protocol, i.e. it encrypts the real user's pcred instead of a dummy.

Hybrid 2 to Hybrid 3. The hybrids are indistinguishable as from the semantic security of the encryption scheme used to produce c_{pcred}, encryption of pcred is indistinguishable from an encryption of a dummy.

Hybrid 4. Similarly as in the previous hybrid, the simulator now changes the way that ciphertext c_{key} is produced. Instead of dummy, it encrypts now user's PRF key k.

Hybrid 2 to Hybrid 4. As previously, semantic security of the threshold encryption scheme assures indistinguishability of the hybrids.

Hybrid 5. In this hybrid the simulator changes its behavior when the $(\texttt{revokeAccount})$ command is executed. That is, it processes threshold decryption of c_{key} which can now succeed as c_{key} contains a real user's key k and not a dummy.

Hybrid 4 to Hybrid 5. Indistingushiability of the hybrids follows from the partial decryption simulatability of the threshold encryption scheme.

Hybrid 6. In this hybrid the simulator changes its behavior when command `traceAccounts` is executed. That is, it outputs the real output of $\mathbf{F}_{mpcprf}$ instead of programming its output. This is possible, since c_{key} consists of the real key not a dummy.

Hybrid 5 to Hybrid 6. The indistinguishability of the hybrids is perfect as simulatability of $\mathbf{F}_{mpcprf}$ is perfect.

Hybrid 7. In the last hybrid the simulator changes its behavior when computing zero knowledge proofs. Now the simulator knows all parts of witnesses corresponding to the proven statements, thus it can compute the proofs honestly (using functionality $\mathbf{F}_{nizk}$).

Hybrid 6 to Hybrid 7. The hybrids are indistinguishable as in the zero-knowledge proof systems simulated proofs are indistinguishable from real ones. Now the simulator presents a view which is exactly the view of the execution of the real protocol.

Since all hops from one hybrid to another leaves the adversary with only negligible probability of distinguishing them, also the first and the final hybrids are distinguishable with negligible probability only. Hence, the simulator succeeds. □

5.3 Joint functionality for identity issuance and policy assignment

Identity issuance functionality $\mathbf{F}_{idpol}^{\mathsf{BS},\mathsf{S},\mathsf{TE},\mathsf{E},\mathsf{t},\mathsf{ar}}$

The functionality is parametrized by

- Blind signature scheme BS,
- Signature scheme S
- (t,ar)-Threshold encryption scheme TE,
- Encryption scheme E.
- Hard relation $\mathbf{R}$.

Setup On $(\texttt{setupStart})$ from any party

- Generate public parameters by running $\mathsf{p}_{idpol} \leftarrow\!\!\$\, (\mathsf{TE.SGen}(1^\lambda), \mathsf{BS.SGen}(1^\lambda), \mathsf{S.SGen}(1^\lambda), \mathsf{E.SGen}(1^\lambda))$;
- Send $(\texttt{setupReady}, \mathsf{p}_{idpol})$ to all parties and set $\mathsf{setupReady} = \mathsf{true}$.

Initialize

Identity provider initialization On input $(\texttt{initialize}, \mathsf{pk}, \mathsf{sk})$ from an identity provider I, ignore if the party has been already initialized, or $\mathsf{setupReady} = \mathsf{false}$.

- If $\mathbf{R}_{bsig}(\mathsf{pk}_{idprov}, \mathsf{sk}_{idprov}) = 1$, then store $(\texttt{idProvInitialized}, \mathsf{I}, (\mathsf{sk}_{idprov}, \mathsf{pk}_{idprov}))$. Send $(\texttt{idProvInitialized}, \mathsf{I}, \mathsf{pk}_{idprov})$ to I and $\mathcal{A}$.
- set $\mathsf{ready} = \mathsf{true}$ if all identity providers were initialized.

Policy assigner initialization Upon input $(\texttt{initialize}, (\mathsf{S.pk_{PA}}, \mathsf{S.sk_{PA}}), (\mathsf{E.sk_{PA}}, \mathsf{E.pk_{PA}}))$ from a policy assigner PA.

- Ignore if
  - PA has already been initialized,
  - $\mathsf{setupReady} = \mathsf{false}$, or
  - $(\mathsf{S.pk_{PA}}, \mathsf{S.sk_{PA}})$ does not fulfill the relation defined by $\mathsf{S.KGen}(\mathsf{p}_{pol})$;
  - $(\mathsf{E.pk_{PA}}, \mathsf{E.sk_{PA}})$ does not fulfill the relation defined by $\mathsf{E.KGen}(\mathsf{p}_{pol})$;
- Else
  - store $(\mathsf{PA}, (\mathsf{S.pk_{PA}}, \mathsf{S.sk_{PA}}), (\mathsf{E.pk_{PA}}, \mathsf{E.sk_{PA}}))$,
  - output $(\texttt{partyInitialized}, \mathsf{S.pk_{PA}}, \mathsf{E.pk_{PA}}, \mathsf{PA})$ to PA and $\mathcal{A}$.
- If all policy assigners have been initialized, set $\mathsf{ready} = \mathsf{true}$.

User initialization On $(\texttt{initialize})$ from a user

- Ignore if
  - U has been initialized;
  - $\mathsf{setupReady} = \mathsf{false}$;
- Else
  - Output $(\texttt{partyInitialized})$ to $\mathcal{A}$ and U.

Identity issuance On input $(\texttt{issueIdentity}, (c, m, r, (\mathsf{pk_{AR_1}}, \ldots, \mathsf{pk_{AR_{ar}}})), \mathsf{aux}_{sig}, (\mathsf{sk_U}, \mathsf{pk_U}), \mathsf{I})$ from a user U and $(\texttt{issueIdentity}, \mathsf{U}, (\mathsf{pk_{AR_1}}, \ldots, \mathsf{pk_{AR_{ar}}}))$ from I:

- If $\mathsf{ready} = \mathsf{false}$, then ignore;
- If $\mathbf{R}_{user}(\mathsf{sk_U}, \mathsf{pk_U}) = 0$ or $c \neq \mathsf{TE.Enc}((\mathsf{pk_{AR_1}}, \ldots, \mathsf{pk_{AR_{ar}}}), m; r)$;
- Else, compute $\sigma_{id} \leftarrow \mathsf{BS.}((\mathsf{sk_U}, m, \mathsf{aux}_{sig}), \mathsf{sk_I})$;
- Output σ_{id} to U and $(\mathsf{pk_U}, c)$ to I.

Attributes check On input $(\texttt{checkAttributes}, \mathsf{PA})$ from user U and $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ from policy assigner PA.

- Read stored aux_{sig}, abort if no aux_{sig} is stored for user U.
- Send to adversary $(\mathsf{len}(\mathbf{P}_{al}), \mathsf{pAL}, \mathsf{pk}_{idprov}, \mathsf{pk}, l_{enc} = \mathsf{len}(\mathsf{E.Enc}(\mathbf{P}_{al})), \mathsf{pk_{PA}})$. **Michal 5.1:** added some adversarial leakage, which the adversary learns in case of the protocol from the fact that the protocol leaks the statement proven in a NIZK way
- If $\mathbf{P}_{al}(\mathsf{aux}_{sig}) = 0$ then send to U $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ and to the adversary $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA})$ exit, else;
- Store $(\texttt{attributesValid}, \mathsf{U}, \mathsf{aux}_{sig}, (\mathsf{pk}, \mathsf{sk}))$ and send $(\texttt{attributesValid}, \mathsf{U}, \mathbf{P}_{al}, \mathsf{PA})$ to the user and $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA})$ to the adversary. **Michal 4.1:** Added $\mathbf{P}_{al}$ to user's output, so the user learns what attribute policy it had assigned

Issue policy On input $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{aux}_{sig})$ from user U and $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp}, \mathbf{P}_{al})$ from PA (or $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp}, \mathbf{P}_{al}, \sigma_{comp})$ if PA is corrupted)

- Check whether $(\texttt{attributesValid}, \mathsf{U}, \mathsf{aux}_{sig}, (\mathsf{pk}, \mathsf{sk}))$ is stored and abort if it is not, in that case output $(\texttt{attributesNotVerified})$.
- Send to adversary $(\mathsf{len}(\mathbf{P}_{comp}))$.
- Store $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{comp})$ and output $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA})$ to the adversary.
- If PA corrupted, verify that $\mathsf{S.Vf}(\mathsf{pk_{PA}}, \sigma_{comp}, (\mathbf{P}_{comp}, \mathbf{P}_{al}, \mathsf{pk})) = 1$ and abort if that is not the case.
- If PA not corrupted, compute $\sigma_{comp} \leftarrow \mathsf{S.}(\mathsf{sk_{PA}}, (\mathbf{P}_{comp}, \mathbf{P}_{al}, \mathsf{pk}))$
- Output $(\texttt{policyAssigned}, \mathsf{paux}, \mathsf{pk}, \mathsf{paux}_{sig}, \mathsf{len}(\mathsf{saux}_{sig}))$ **Michal 17.12:** Do we need to reveal so much about the party that got the policy assigned? Could we do without revealing, say pAL – that is reveal it only to the policy assigner, but not to the adversary? to $\mathcal{A}$ (or $(\texttt{policyAssigned}, \mathsf{aux}, \mathsf{paux}_{sig}, \mathsf{saux}_{sig}, \mathbf{P}_{comp}, \mathbf{P}_{al}, \sigma_{comp})$ if PA or U is corrupted); $(\texttt{policyAssigned})$ to PA; and $(\texttt{policyAssigned}, \mathbf{P}_{comp}, \sigma_{comp})$ to U.

Protocol $\mathbf{P}_{idpol}^{\mathsf{BS},\mathsf{S},\mathsf{TE},\mathsf{E},\mathsf{t},\mathsf{ar}}$

The protocol operates in the $\{\mathbf{F}_{smt}, \mathbf{F}_{nizk}, \mathbf{F}_{reg}\}$-hybrid model and uses the following cryptographic primitives:

- Blind signature scheme BS,
- Signature scheme S
- $(\mathsf{t}, \mathsf{ar})$-Threshold encryption scheme TE,
- Encryption scheme E.
- Hard relation $\mathbf{R}$.

Setup

- On input $(\texttt{setupStart})$ use $\mathbf{F}_{pp}^{\mathsf{TE.SGen},\lambda}$, $\mathbf{F}_{pp}^{\mathsf{E.SGen},\lambda}$, $\mathbf{F}_{pp}^{\mathsf{BS.SGen},\lambda}$, $\mathbf{F}_{pp}^{\mathsf{S.SGen},\lambda}$ to generate public parameters p and publicize it to all parties.
- Send $(\texttt{setupReady}, \mathsf{p}_{idpol})$ to all parties and set $\mathsf{setupReady} = \mathsf{true}$.

Initialize

Identity provider initialization On input $(\texttt{initialize}, (\mathsf{sk}_{idprov}, \mathsf{pk}_{idprov}))$ the identity provide

- check if the key pair has correct distribution with respect to $\mathsf{BS.KGen}(\mathsf{p})$ and ignore if that is not the case,
- else, store $(\mathsf{sk}_{idprov}, \mathsf{pk}_{idprov})$ and send $(\texttt{register}, \mathsf{sk}_{idprov}, \mathsf{pk}_{idprov})$ to $\mathbf{F}_{reg}$,
- output $(\texttt{partyInitialized}, \mathsf{pk}_{idprov}, \mathsf{I})$ to I.

Policy assigner initialization On $(\texttt{initialize}, (\mathsf{S.pk}_{\mathsf{PA}}, \mathsf{S.sk}_{\mathsf{PA}}), (\mathsf{E.pk}_{\mathsf{PA}}, \mathsf{E.sk}_{\mathsf{PA}}))$ the policy assigner checks if the key pairs have correct distribution with respect to $\mathsf{S.KGen}(\mathsf{p}_{pol})$ and $\mathsf{E.KGen}(\mathsf{p}_{pol})$. If it does, store $(\mathsf{S.pk}_{\mathsf{PA}}, \mathsf{S.sk}_{\mathsf{PA}})$, $(\mathsf{E.pk}_{\mathsf{PA}}, \mathsf{E.sk}_{\mathsf{PA}})$ and send to $\mathbf{F}_{reg}$:

- $(\texttt{register}, \mathsf{S.pk}_{\mathsf{PA}}, \mathsf{S.sk}_{\mathsf{PA}})$,
- $(\texttt{register}, \mathsf{E.pk}_{\mathsf{PA}}, \mathsf{E.sk}_{\mathsf{PA}})$

Output $(\texttt{partyInitialized}, \mathsf{S.pk}_{\mathsf{PA}}, \mathsf{E.pk}_{\mathsf{PA}}, \mathsf{PA})$ to PA.

User initialization On $(\texttt{initialize}, (\mathsf{pk}, \mathsf{sk}), \mathsf{aux}_{sig})$

- Check whether $(\mathsf{pk}, \mathsf{sk}) \in \mathbf{R}$ and ignore if that is not the case.
- Register pk at $\mathbf{F}_{reg}$ by sending $(\texttt{register}, (\mathsf{pk}, \mathsf{sk}))$.
- Output $(\texttt{partyInitialized}, \mathsf{pk})$ to U.

Identity issuance

- On input $(\texttt{issueIdentity}, (c, m, r, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})), \mathsf{aux}_{sig}, (\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}}), \mathsf{I})$ to the user U and an input $(\texttt{issueIdentity}, \mathsf{U}, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}}))$ to the identity provider:
  1. The user
     - gets pk_{idprov} from $\mathbf{F}_{reg}$;
     - Computes $\sigma_1 = \mathsf{PrepSign}(\mathsf{pk}_{idprov}, (\mathsf{sk}_{\mathsf{U}}, m, \mathsf{aux}), r')$;
     - Sends $(\texttt{proveStatement}, (\mathsf{pk}_{\mathsf{U}}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p}), (\mathsf{sk}_{\mathsf{U}}, m, r, r'))$ for relation $\mathbf{R}_{sign}$; // The user shows that c contains valid encryption of its credentials and it knows a valid secret key for its public key.
     - Upon receiving the proof π from the functionality, sends $(\mathsf{send}, \mathsf{I}, (\mathsf{pk}_{\mathsf{U}}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p}), \pi)$ to $\mathbf{F}_{smt}$;
  2. Upon receiving $(\mathsf{sent}, \mathsf{U}, ((\mathsf{pk}_{\mathsf{U}}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p}), \pi))$ from $\mathbf{F}_{smt}$, the identity provider I:
     - Inputs $(\texttt{verifyProof}, (\mathsf{pk}_{\mathsf{U}}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p}), \pi)$ to $\mathbf{F}_{nizk}^{\mathbf{R}_{sign}}$.
     - If the NIZK functionality approves the proof, compute $\sigma_2 \leftarrow \mathsf{BlindSign}(\mathsf{sk}_{idprov}, \sigma_1)$;
     - Send σ_2 to U using $\mathbf{F}_{smt}$;
  3. User U unblinds the signature by running $\mathsf{Unblind}(\sigma_2, r')$ and gets the signature σ on $(\mathsf{sk}_{\mathsf{U}}, m, \mathsf{aux})$.

Attributes check

User's actions On $(\texttt{checkAttributes}, \mathsf{PA})$. On $(\mathbf{P}_{al})$ sent by PA via $\mathbf{F}_{smt}$,

- if $\mathbf{P}_{al}(\mathsf{aux}_{sig}) = 0$ then output $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$; **Michal 4.1:** Actually, I am not sure how this message should be handled. Do we need to send anything to the polas / functionality **Michal 4.1:** After consulting DGKOS20 – what is returned in the protocol is what functionality returns to the calling party
- else call $\mathbf{F}_{nizk}^{\mathbf{R}_{al}}$ on $(\texttt{proveStatement}, ((\mathsf{paux}_{sig}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), (\mathsf{saux}_{sig}, \mathbf{P}_{al}, \sigma_{id}, \mathsf{sk})))$ to get proof π_{al};
- send $((\mathsf{paux}_{sig}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), \pi_{al})$ to the policy assigner using $\mathbf{F}_{smt}$.
- get $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ from the policy assigner.

Policy assigner's actions On $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$

- if there is $(\mathbf{P}'_{al}, \mathsf{U})$ stored and $\mathbf{P}_{al} \neq alpolicy$ then abort
- send $(\mathbf{P}_{al})$ to user U via $\mathbf{F}_{smt}$;
- on $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ from U, abort;
- get user's public key pk from $\mathbf{F}_{reg}$;
- on $((\mathsf{paux}_{sig}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), \pi_{al})$ from user U, call $\mathbf{F}_{nizk}^{\mathbf{R}_{al}}(\texttt{verifyProof}, (\mathsf{paux}_{sig}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), \pi_{al}))$ and abort if the proof does not verify;
- abort if pk_{idprov} is not valid identity provider's public key; else
- send $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ to U.
- save $(\mathbf{P}_{al}, \mathsf{U})$.

Policy issuance

User's actions On $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{aux}_{sig})$ **Michal 28.12:** Changed functionality input. Important, seems that we need idprov's signature on AL, otherwise a malicious user could generate it by its own.

- Parse $\mathsf{aux}_{sig} = (\mathsf{paux}_{sig}, \mathsf{saux}_{sig})$, where paux_{sig} is the public part of the input and saux_{sig} the private part.
- Send $(\texttt{issuePolicy})$ to PA using $\mathbf{F}_{smt}$.
- On $(\mathbf{P}_{comp}, \sigma_{comp})$ that PA sent by $\mathbf{F}_{smt}$, store the message.

Policy assigner actions On $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp})$

- If U has no $\mathbf{P}_{al}$ assigned, then abort, else read $\mathbf{P}_{al}$.
- Compute $\sigma_{comp} \leftarrow \mathsf{S}.(\mathsf{sk}_{polas}, (\mathbf{P}_{comp}, \mathbf{P}_{al}, \mathsf{pk}))$;
- Send $(\mathbf{P}_{comp}, \sigma_{comp})$ to the user using $\mathbf{F}_{smt}$.

5.3.1 Security proof for $\mathbf{P}_{idpol}$

Michal 4.1: Need to check whether what protocol outputs matches with what ideal functionality outputs. For example, the protocol uses $\mathbf{F}_{nizk}$ which reveals proven statement to the adversary – we need to make sure that the functionality reveals the same information.

Simulator Sim_{idpol}

Setup

1. Honest party P, on $(\texttt{setupStart}, \mathsf{p}_{pol})$ from the functionality
   - set up all simulated functionalities according to p_{pol}
2. Dishonest party P, on $(\texttt{generatePP})$ to $\mathbf{F}_{pp}^{\mathsf{GGen},\lambda}$.
   - Send $(\texttt{setupStart})$ to the functionality.

Initialize **Michal 30.12:** Need to change that due to changes in the functionality inputs

Identity provider initialization

1. Honest I on $(\texttt{partyInitialized}, (\mathsf{pk}_{idprov}))$

- Pick randomly sk_{idprov} and simulate calling $\mathbf{F}_{reg}$ on $(\texttt{register}, (\mathsf{S.sk}_{idprov}, \mathsf{S.pk}_{idprov}))$.
- Add I to the set of initialized parties.

2. Dishonest I on $(\texttt{register}(\mathsf{pk}_{idprov}, \mathsf{sk}_{idprov}))$ sent by I to $\mathbf{F}_{reg}$
   - Ignore if $(\mathsf{pk}_{idprov}, \mathsf{sk}_{idprov}) \notin \mathrm{Im}(\mathsf{S.KGen})$, the user has already been initialized, or $\mathsf{setupRead} = \mathsf{false}$.
   - Else, input to $\mathbf{F}_{\mathsf{PPCTS}}$ command $(\texttt{initialize}, (\mathsf{pk}_{polas}, \mathsf{sk}_{polas}))$

Policy assigner initialization

1. Honest policy assigner PA, on $(\texttt{partyInitialized}, \mathsf{S.pk_{PA}}, \mathsf{E.pk}_{polas}, \mathsf{PA})$ from the functionality
   - Pick randomly $\mathsf{S.sk_{PA}}$ and simulate calling $\mathbf{F}_{reg}$ on $(\texttt{register}, (\mathsf{S.pk_{PA}}, \mathsf{S.sk_{PA}}))$.
   - Pick randomly $\mathsf{E.sk_{PA}}$ and simulate calling $\mathbf{F}_{reg}$ on $(\texttt{register}, (\mathsf{E.pk_{PA}}, \mathsf{E.sk_{PA}}))$.
   - Add PA to the set of initialized parties Init. **Michal 17.12:** Do we need to keep the party set Init?
2. Dishonest policy assigner PA, on $(\texttt{register}, (\mathsf{S.pk}_{polas}, \mathsf{S.sk}_{polas})$ sent by U to $\mathbf{F}_{reg}$,
   - Ignore if $(\mathsf{S.pk}_{polas}, \mathsf{S.sk}_{polas}) \notin \mathrm{Im}(\mathsf{S.KGen})$, the user has already been initialized, or $\mathsf{setupRead} = \mathsf{false}$.

   On $(\texttt{register}, (\mathsf{E.pk}_{polas}, \mathsf{E.sk}_{polas})$ sent by U to $\mathbf{F}_{reg}$
   - Ignore if $(\mathsf{E.pk}_{polas}, \mathsf{E.sk}_{polas}) \notin \mathrm{Im}(\mathsf{E.KGen})$
   - Else, input to $\mathbf{F}_{\mathsf{PPCTS}}$ command $(\texttt{initialize}, (\mathsf{S.pk}_{polas}, \mathsf{S.sk}_{polas}), (\mathsf{E.pk}_{polas}, \mathsf{E.sk}_{polas}))$ and add U to the set of initialized parties.

User initialization

1. Honest user U, on $(\texttt{partyInitialized}, \mathsf{pk})$
   - Pick randomly sk and simulate calling $\mathbf{F}_{reg}$ on $(\texttt{register}, (\mathsf{pk}, \mathsf{sk}))$. / Note that with overwhelming probability $(\mathbf{R}(\mathsf{pk}, \mathsf{sk}) = 0$ hence the simulator has to simulate $\mathbf{F}_{reg}$ instead of running it honestly
2. Dishonest user U, on $(\texttt{register}, (\mathsf{pk}, \mathsf{sk}))$ sent by U to $\mathbf{F}_{reg}$,
   - Ignore if $(\mathsf{pk}, \mathsf{sk}) \in \mathbf{R}$, the user has already been initialized, or $\mathsf{setupRead} = \mathsf{false}$.
   - Else, input to $\mathbf{F}_{\mathsf{PPCTS}}$ command $(\texttt{initialize}, (\mathsf{pk}, \mathsf{sk}))$

Identity issuance

1. Honest user U and identity provider I. **Michal 4.1:** Both in our functionality and in DGKOS20 the adversary learns nothing while honest user talks with an honest identity provider. OTOH DGKOS20 also doesn't consider a case of both parties honest. **Michal 5.1:** To my understanding, it is fine to not write a proof for honest-honest case as long as the adversary doesn't learn anything from the functionality
2. Honest user U and malicious identity provider I.
   - on behalf of I send to $\mathbf{F}_{idpol}$ input $(\texttt{issueIdentity}, \mathsf{U}, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}}))$;
   - make a simulated signature σ_1 and then a simulated proof π for $(\mathsf{pk_U}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p})$ using $\mathbf{F}_{nizk}^{\mathbf{R}_{sign}}$;
   - on σ_2 sent by the identity provider, unblind the signature and obtain σ_{id}.
   - Output σ_{id} to U.
3. Malicious user U and honest identity provider I.
   - On $(\texttt{proveStatement}, (\mathsf{pk_U}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p}), (\mathsf{sk_U}, m, r, r'))$ sent by the user to $\mathbf{F}_{nizk}^{\mathbf{R}_{sign}}$, pass it to the functionality and return the proof π.
   - Run extractor on $\pi, (\mathsf{pk_U}, \sigma_1, c, \mathsf{aux}_{sig}, \mathsf{p})$ and reveal the witness. If the witness is invalid output fail.
   - Else, send to $\mathbf{F}_{idpol}$ $(\texttt{issueIdentity}, (c, m, r, (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})), \mathsf{aux}_{sig}, (\mathsf{sk_U}, \mathsf{pk_U}), \mathsf{I})$ on behalf of U.
   - Get from $\mathbf{F}_{idpol}$ signature σ_{id} and compute simulated signature σ_2 which unblinded gives σ_{id}. / Security of this step follows from the simulatability of the signature scheme **Michal 4.1:** Check this step – does the simulator needs to know σ_{id} to compute σ_2?
4. Malicious user U and identity provider I. We do not consider this case. If user and identity provider collude then the user could have assigned any identity it wants.

Attributes check

1. Honest user U and policy assigner PA. On $(l_{alpolicy}, \mathsf{pAL}, \mathsf{pk}_{idprov}, \mathsf{pk}, l_{enc}, \mathsf{pk}_{\mathsf{PA}})$
   (a) On $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA})$ from the functionality [**Michal 5.1:** Do I need to reveal U here?]
   - Pick random policy $\mathbf{P}'_{al}$ of length $l_{alpolicy}$.
   - Using $\mathbf{F}_{smt}$ send $(\mathbf{P}'_{al})$ to U.
   - Output $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA})$

   (b) On $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA})$
   - Pick random policy $\mathbf{P}'_{al}$ of length $l_{alpolicy}$.
   - Using $\mathbf{F}_{smt}$ send $(\mathbf{P}'_{al})$ to U.
   - Compute simulated proof π_{al} for statement $\mathbf{R}_{al}((\mathsf{pAL}, \mathsf{pk}_{idprov}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), (\mathsf{sAL}, \mathbf{P}'_{al}, \sigma_{id}, \mathsf{sk}, \mathsf{k}))$, for some randomly picked $\mathsf{sAL}, \sigma_{id}, \mathsf{sk}, \mathsf{k}$ and c being an encryption of $\mathbf{P}'_{al}$, [**Michal 29.12:** Problem, the simulator picks randomly $\mathbf{P}'_{al}$ but later has to make a zkproof that reveals $\mathbf{P}_{al}$.] [**Michal 30.12:** Solution – put in the instance only encryption (under PA's public key) of the alpolicy]
   - Send the proof to the policy assigner PA.
   - Simulate proof verification by PA.
   - If the proof verifies correctly, send $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA})$ to user U.
2. Honest user U and malicious policy assigner PA. On $(\mathbf{P}_{al})$ sent by the policy assigner via $\mathbf{F}_{smt}$.
   - Input $(\texttt{checkAttributes}, \mathsf{U}, \mathbf{P}_{al})$ to the functionality $\mathbf{F}_{\mathsf{PPCTS}}$ on behalf of PA.
   - On $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA})$ from the functionality
     - compute simulated proof π_{al} for statement $\mathbf{R}_{al}((\mathsf{pAL}, \mathsf{pk}_{idprov}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), (\mathsf{sAL}, \mathbf{P}_{al}, \sigma_{id}, \mathsf{sk}, \mathsf{k}))$, for some randomly picked $(\mathsf{sAL}, \mathbf{P}_{al}, \sigma_{id}, \mathsf{sk}, \mathsf{k})$ and $c = \mathsf{E.Enc}(\mathbf{P}_{al})$
     - Send the proof to the policy assigner PA.
     - If the proof verifies correctly, send $(\texttt{attributesValid}, \mathsf{U}, \mathbf{P}_{al})$ to user U.
   - On $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA})$, output $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA})$ to the user.
3. Malicious user U and honest policy assigner PA.
   (a) On $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$
   - Input to $\mathbf{F}_{\mathsf{PPCTS}}$ $(\texttt{checkAttributes}, \mathsf{PA}, \mathsf{AL}, (\mathsf{pk}, \mathsf{sk}), (\mathsf{pk}_{id}, \sigma_{id}, \mathsf{k}))$ on behalf of user.
   - Pick a random policy $\mathbf{P}'_{al}$ and send it via $\mathbf{F}_{smt}$ to U.
   - On $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA})$ from the functionality, read $(\texttt{attributesNotValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ sent by the functionality to U and exit.
   - Else, on $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA})$ from the functionality, read $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ sent by the functionality to U. [**Michal 4.1:** That's how the simulator can learn $\mathbf{P}_{al}$ sent to a malicious party.]
   - Make simulated proof π_{al} for instance $(\mathsf{paux}_{sig}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}})$ using simulated functionality $\mathbf{F}_{nizk}^{\mathbf{R}_{al}}$, here c is an encryption of $\mathbf{P}_{al}$ under PA public key.

   (b) On $((\mathsf{paux}_{sig}, \mathsf{pk}_{\mathsf{I}}, \mathsf{pk}, c, \mathsf{pk}_{\mathsf{PA}}), \pi_{al})$ sent to PA
   - Abort if the proof does not verify.
   - Abort if pk_{idprov} is not a valid public key of the identity provider.
   - Else, extract witness $(\mathsf{saux}_{sig}, \mathbf{P}_{al}, \sigma_{id}, \mathsf{sk})$ if this fails output fail.
   - Extract $\mathbf{P}_{al}$ from c.
   - Send to $\mathbf{F}_{idpol}$ input $(\texttt{checkAttributes}, \mathsf{PA})$.
   - Send $(\texttt{attributesValid}, \mathsf{U}, \mathsf{PA}, \mathbf{P}_{al})$ to U.
4. Malicious user U and policy assigner PA. We do not allow both U and PA to be corrupted at the same time. This would allow the adversary to have accepted attribute list that was not certified by the identity provider.

Issue policy

1. Honest user U and policy assigner PA.
   - Check if there is stored $(\mathbf{P}_{al}, \mathsf{U})$; if not, abort; else read $\mathbf{P}_{al}$.
   - Get $(l_{comppolicy})$ from $\mathbf{F}_{idpol}$.
   - Pick a random policy $\mathbf{P}'_{comp}$ of size $l_{comppolicy}$.
   - On $(\texttt{policyAssigned}, \mathsf{U}, \mathsf{PA})$ from $\mathbf{F}_{idpol}$
     - send $(\texttt{issuePolicy})$ to PA on behalf of U;
     - on behalf of PA compute $\sigma_{comp} \leftarrow \mathsf{S.}(\mathsf{sk}_{polas}, (\mathbf{P}_{comp}, \mathbf{P}_{al}, \mathsf{pk}))$
2. Honest U and malicious policy assigner PA. On $(\mathbf{P}_{comp}, \sigma_{comp})$ sent by PA to U using $\mathbf{F}_{smt}$;
   - Call $\mathbf{F}_{idpol}$ on $(\texttt{issuePolicy}, \mathsf{U}, \mathbf{P}_{comp})$ on behalf of PA.
   - Send to PA via $\mathbf{F}_{smt}$ $(\texttt{issuePolicy})$,
3. Malicious U and honest policy assigner PA. On $(\texttt{send}, \mathsf{PA}, \texttt{issuePolicy})$ sent by U to PA by $\mathbf{F}_{smt}$
   - Call the functionality $\mathbf{F}_{idpol}$ on $(\texttt{issuePolicy}, \mathsf{PA}, \mathsf{AL})$ for AL stored by the simulator.
   - Abort if there is no stored $(\mathbf{P}_{al}, \mathsf{U})$.
   - Pick a random compliance policy $\mathbf{P}'_{comp}$ and send $(\mathbf{P}'_{comp}, \sigma_{comp} \leftarrow \mathsf{S.}(\mathsf{sk}_{polas}, (\mathbf{P}'_{comp}, \mathbf{P}_{al}, \mathsf{pk})))$ via $\mathbf{F}_{smt}$ to U.
4. *Malicious U and honest policy assigner PA.* We do not allow both U and PA to be corrupted at the same time as it only verify given signature (what is also done in $\mathbf{F}_{idpol}$) and send input messages using $\mathbf{F}_{smt}$. Hence, the adversary could only

Theorem 2 ($\mathbf{F}_{idpol}$ securely realizes $\mathbf{F}_{idpol}$). *Let S be existentially unforgeable signature scheme, then $\mathbf{P}_{pol}$ securely realizes $\mathbf{F}_{idpol}$ in the $\{\mathbf{F}_{pp}, \mathbf{F}_{srs}, \mathbf{F}_{reg}, \mathbf{F}_{smt}\}$-hybrid model.*

distinguish the real and ideal world in case of honest user and policy assigner. However, a adversary that tells whether $\mathbf{F}_{smt}$ transfers a random message m (resp. m') or user's credentials (resp. assigned policy) breaks security of $\mathbf{F}_{smt}$ what happens with negligible probability only. □

6 Conclusion

Antoine 24.08: TODO

References

20001. Anti-terrorism, Crime and Security Act 2001, 2001 c. 24, 2001.

ABLZ17. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Heidelberg, December 2017.

Age. National Crime Agency. Suspicious activity reports. `https://www.nationalcrimeagency.gov.uk/what-we-do/crime-threats/money-laundering-and-illicit-finance/suspicious-activity-reports`. Last accessed: 2021-03-08.

Age19. National Crime Agency. Introduction to suspicious activity reports. `https://www.nationalcrimeagency.gov.uk/who-we-are/publications/158-introduction-to-suspicious-activity-reports-sars/file`, 2019.

AGH+18. Joe Andrieu, Nathan George, Andrew Hughes, Christophe MacIntosh, and Antoine Rondelet. Five mental models of identity. Rebooting the Web of Trust VII, 2018. `https://github.com/WebOfTrustInfo/rwot7-toronto/blob/master/final-documents/mental-models.pdf`.

AGR+16. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219. Springer, Heidelberg, December 2016.

ana20. *Analysis of Darknet Market Activity as a Country-Specific, Socio-Economic and Technological Phenomenon*, 2020. `https://docs.apwg.org/ecrimeresearch/2020/29_Analysis_of_Darknet_Sales-eCrime_2020-Preconference_version.pdf`.

Aut12. Financial Services Authority. Journey to the fca. `https://www.fca.org.uk/publication/corporate/fsa-journey-to-the-fca.pdf`, 2012.

Aut19. Financial Conduct Authority. The uk financial conduct authority and the us securities and exchange commission sign updated supervisory cooperation arrangements. `https://www.fca.org.uk/news/press-releases/uk-financial-conduct-authority-and-us-securities-and-exchange-commission-sign-updated-supervisory`, 2019.

BAZB19. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Report 2019/191, 2019. `https://eprint.iacr.org/2019/191`.

BCG+14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

BCG+21. Kenneth A. Bamberger, Ran Canetti, Shafi Goldwasser, Rebecca Wexler, and Evan Zimmerman. Verification dilemmas, law, and the promise of zero-knowledge proofs. `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3781082`, 2021.

BFS16. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 777–804. Springer, Heidelberg, December 2016.

BPW+20. Apolline Blandin, Dr. Gina Pieters, Yue Wu, Thomas Eisermann, Anton Dek, Sean Taylor, and Damaris Njoki. 3rd global cryptoasset benchmarking study. Technical report, University of Cambridge (Cambridge Center for Alternative Finance), 2020. `https://www.jbs.cam.ac.uk/wp-content/uploads/2021/01/2021-ccaf-3rd-global-cryptoasset-benchmarking-study.pdf`.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993.

btc. `http://btcrelay.org/`. Last accessed: 2021-09-13.

CA09. US Commodity Futures Trading Commission and UK Financial Services Authority. Memorandum of understanding concerning cooperation and the exchange of information related to the supervision of cross-border clearing organizations. `https://www.cftc.gov/sites/default/files/idc/groups/public/@internationalaffairs/documents/file/ukfsa09.pdf`, 2009.

Can00. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

CFF+20. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. Cryptology ePrint Archive, Report 2020/1069, 2020. `https://eprint.iacr.org/2020/1069`.

Cha. Chainalysis. `https://www.chainalysis.com/`. Last accessed: 2021-09-13.

Cha83. David Chaum. Blind signature system. In David Chaum, editor, *Advances in Cryptology – CRYPTO'83*, page 153. Plenum Press, New York, USA, 1983.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, Heidelberg, May 2020.

CL15. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 487–505. Springer, Heidelberg, April 2015.

CLT18. Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 733–764. Springer, Heidelberg, December 2018.

Com. U.S. Commodity Futures Trading Commission. Dodd-frank act. `https://www.cftc.gov/LawRegulation/DoddFrankAct/index.htm`.

Com19. U.S. Commodity Futures Trading Commission. Joint statement by uk and us authorities on continuity of derivatives trading and clearing post-brexit. `https://www.cftc.gov/PressRoom/PressReleases/7876-19`, 2019.

CT20. Allison Christians and Mahwish Tazeem. Facebook's libra: The next tax challenge for the digital economy. *Stanford Journal of Blockchain Law & Policy*, 6 2020. https://stanford-jblp.pubpub.org/pub/libra-tax-challenge.

DGK+20. Ivan Damgård, Chaya Ganesh, Hamidreza Khoshakhlagh, Claudio Orlandi, and Luisa Siniscalchi. Balancing privacy and accountability in blockchain transactions. Cryptology ePrint Archive, Report 2020/1511, 2020. `https://eprint.iacr.org/2020/1511`.

Dot92. James R. Doty. The role of securities and exchange commission in an internationalized marketplace. 60 Fordham L. Rev. S77, 1992. `https://ir.lawnet.fordham.edu/flr/vol60/iss6/5`.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, Heidelberg, January 2005.

Edm18. Tim Edmonds. Briefing paper number 2592, money laundering law. `https://researchbriefings.files.parliament.uk/documents/SN02592/SN02592.pdf`, 2 2018.

Eff94. Robert Effros. *Chapter 10 Regulatory and Other Changes in the U.K. Banking Market*. INTERNATIONAL MONETARY FUND, USA, 1994.

Ell. Elliptic. `https://www.elliptic.co/`. Last accessed: 2021-09-13.

eth. `https://ethereum.org/en/developers/docs/evm/`. Last accessed: 2021-09-13.

FAT21a. FATF. Public consultation on fatf draft guidance on a risk-based approach to virtual assets and virtual asset service providers. https://www.fatf-gafi.org/publications/fatfrecommendations/documents/public-consultation-guidance-vasp.html, 2021.

FAT21b. FATF. Updated guidance for a risk-based approach to virtual assets and virtual asset service providers. https://www.fatf-gafi.org/publications/fatfrecommendations/documents/guidance-rba-virtual-assets-2021.html, 2021.

For. Financial Action Task Force. `https://www.fatf-gafi.org/about/`. Last accessed: 2021-03-08.

For19. Financial Action Task Force. Guidance for a risk-based approach to virtual assets and virtual asset service providers. `www.fatf-gafi.org/publications/fatfrecommendations/documents/Guidance-RBA-virtual-assets.html`, 2019.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.

Fuc18. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 315–347. Springer, Heidelberg, March 2018.

Gou. Jonathan V. Gould. Occ chief counsel's interpretation on national bank and federal savings association authority to use independent node verification networks and stablecoins for payment activities. `https://www.occ.gov/news-issuances/news-releases/2021/nr-occ-2021-2a.pdf`.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, Heidelberg, May 2016.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

HBHW21. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`, 2021. Version 2021.2.14 [NU5 proposal].

Keh20. Leonard Kehnscherper. Swiss canton takes taxes in bitcoin as crypto gains traction. `https://www.bloomberg.com/news/articles/2020-09-03/swiss-canton-takes-taxes-in-bitcoin-as-crypto-goes-mainstream`, 09 2020. Last accessed: 2021-03-08.

Lee79. T. Peter Lee. Significant Developments in the United Kingdom in 1979. *Journal of Comparative Corporate Law and Securities Regulation 2*, pages 335–337, 1979.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2111–2128. ACM Press, November 2019.

New21. BBC News. Bitcoin: El salvador makes cryptocurrency legal tender. https://www.bbc.co.uk/news/world-latin-america-57398274, 2021.
Para. UK Parliament. The money laundering regulations 1993, si 1993/1933. https://www.legislation.gov.uk/uksi/1993/1933/made.
Parb. UK Parliament. The money laundering regulations 2003, si 2003/3075. https://www.legislation.gov.uk/uksi/2003/3075/introduction/made.
PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 111–126. Springer, Heidelberg, February / March 2016.
rai. https://github.com/aurora-is-near/rainbow-bridge. Last accessed: 2021-09-13.
Rap20. Kenneth Rapoza. After ledger hack, who can you trust for bitcoin storage? https://www.forbes.com/sites/kenrapoza/2021/12/28/after-ledger-hack-who-can-you-trust-for-bitcoin-storage/, 2020.
Ree. Alan Reed. https://www.bitcoinabuse.com/. Last accessed: 2021-09-13.
Ron20a. Antoine Rondelet. A note on anonymous credentials using BLS signatures. *CoRR*, abs/2006.05201, 2020.
Ron20b. Antoine Rondelet. Thoughts on the design of zeth bearer instruments (zbis). https://medium.com/clearmatics/thoughts-on-the-design-of-zeth-bearer-instruments-zbis-725c1b32d64c, 2020.
Ron20c. Antoine Rondelet. Zecale: Reconciling privacy and scalability on ethereum. *CoRR*, abs/2008.05958, 2020.
Ron21. Antoine Rondelet. Zeth cadcad simulations. https://github.com/clearmatics/simodelation/blob/master/zeth/zeth-cadCAD-simulation.ipynb, 2021.
RT20. Antoine Rondelet and Duncan Tebbs. On the design of zeth transaction relays. https://github.com/clearmatics/zeth-specifications/blob/master/relay/zeth_relay.pdf, 2020.
RZ19. Antoine Rondelet and Michal Zajac. ZETH: on integrating zerocash on ethereum. *CoRR*, abs/1904.00905, 2019.
SBL20. Dana V. Syracuse, Joshua L. Boehm, and Nick Lundgren. Anti-Money Laundering Regulation of Privacy-Enabling Cryptocurrencies. https://www.perkinscoie.com/images/content/2/3/v7/237411/Perkins-Coie-LLP-White-Paper-AML-Regulation-of-Privacy-enablin.pdf, 2020.
SC. U.S. Securities and Exchange Commission. Framework for "investment contract" analysis of digital assets. https://www.sec.gov/corpfin/framework-investment-contract-analysis-digital-assets.
SCA06. U.S. Securities, Exchange Commission, and U.K. Financial Services Authority. Memorandum of understanding. https://www.sec.gov/about/offices/oia/oia_multilateral/ukfsa_mou.pdf, 2006.
SFS+20. Erik Silfversten, Marina Favaro, Linda Slapakova, Sascha Ishikawa, James Liu, and Adrian Salas. *Exploring the use of Zcash cryptocurrency for illicit or criminal purposes*. RAND Corporation, Santa Monica, CA, 2020.
tai20. *Averages don't characterise the heavy tails of ransoms*, 2020. https://docs.apwg.org/ecrimeresearch/2020/22_Averages_don_t_characterise_the_heavy_tails_of_ransoms.pdf.
TGL20. Rashida Tlaib, Jesus "Chuy" García, and Stephen Lynch. A bill to amend the federal deposit insurance act to provide for the classification and regulation of stablecoins, and for other purposes. https://tlaib.house.gov/sites/tlaib.house.gov/files/STABLEAct.pdf, 2020.
Val76. Leslie G. Valiant. Universal circuits (preliminary report). In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 196–203. ACM, 1976.
Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, March 2008.
Wil19. Zachary J. Williamson. The aztec protocol. https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf, 2019.
Woo19. Dr Gavin Wood. ETHEREUM: A Secure Decentralised Generalised Transaction Ledger Byzantium. https://ethereum.github.io/yellowpaper/paper.pdf, 2019. [VERSION 7e819ec - 2019-10-20].
Yor. David Yormesor. Eib issues its first ever digital bond on a public blockchain. https://www.eib.org/en/press/all/2021-141-european-investment-bank-eib-issues-its-first-ever-digital-bond-on-a-public-blockchain.
Zmu20. Adrian Zmudzinski. Hacked ledger customers fear for safety. https://modernconsensus.com/cryptocurrencies/hacked-ledger-customers-fear-for-safety/, 2020.

A Additional functionalities

NIZK functionality The functionality describes an ideal execution of a NIZK system. That is, a valid statement is always accepted by the functionality, no party can make the functionality accept an invalid statement, no information about the statement's witness are revealed.

NIZK functionality $\mathbf{F}_{nizk}^{\mathbf{R}}$

Functionality is parametrised by a relation $\mathbf{R}$.

Proving statements On $(\texttt{proveStatement}, \mathsf{x}, \mathsf{w})$ from party P:
- Check whether $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$ and, if that is not the case, ignore.
- Else send $(\texttt{proveStatement}, \mathsf{x})$ to $\mathcal{A}$ and wait for answer $(\texttt{statementProven}, \pi)$.
- Store (x, π), send $(\texttt{statementProven}, \pi)$ to P.

Verifying statements On $(\texttt{verifyProof}, \mathsf{x}, \pi)$ from party P:

- Check whether (x, π) is stored. If not, send $(\texttt{giveWitness}, \mathsf{x})$ to $\mathcal{A}$.
- On answer w, check whether $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$. If that is the case store (x, π).
- If there is (x, π) stored, send $(\texttt{statementVerified}, 1)$. Otherwise send $(\texttt{statementVerified}, 0)$.

SRS functionality The functionality assures secure generation of the SRS.

SRS functionality $\mathbf{F}_{srs}^{\mathsf{KGen},\mathsf{KGenSpec},\lambda}$

1. On input $(\texttt{generateMasterSRS})$,
   - generate $\mathsf{srs} \leftarrow_{\$} \mathsf{KGen}(\lambda)$,
   - broadcast srs to all parties.
2. On input $(\texttt{generateSpecializedSRS}, \mathbf{R})$
   - generate $\mathsf{srs}^{\mathsf{spec}} \leftarrow_{\$} \mathsf{KGenSpec}(\mathbf{R})$
   - broadcast $\mathsf{srs}^{\mathsf{spec}}$ to all parties.

[**Michal 12.08:** Check! changed functionality to cover two part SRS generation]

Public parameter functionality The functionality assures secure generation of the public parameters. For a list of protocols $p_1, p_2, \ldots$ the functionality runs $p_i.\mathsf{SGen}(1^\lambda)$ and outputs the resulting public parameters.

Public parameters functionality $\mathbf{F}_{pp}^{(p_1,p_2,\ldots,p_n),\lambda}$

On input $(\texttt{generatePP})$, if that is the first time this input has been given, generate $\mathsf{p}_i \leftarrow_{\$} p_i.\mathsf{SGen}(1^\lambda)$, for $i \in [1..n]$ and broadcast p_i to all parties. Ignore further $(\texttt{generatePP})$ queries.

Register functionality The functionality is responsible for receiving and storing users public and secret keys. It assures that no party can change its key-pair and all the parties retrieves the public keys they request.

Register functionality $\mathbf{F}_{reg}$

Registration Upon receiving $(\texttt{register}, \mathsf{sk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})$ from party P, keep $(\mathsf{P}, \mathsf{sk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})$ and return $(\texttt{initialized}, \mathsf{P})$ if that is the first request from P. Otherwise ignore.

Retrieval Upon receiving $(\texttt{retrieve}, \mathsf{P}')$ from some party P or the adversary, output $(\texttt{retrieve}, \mathsf{P}', \mathsf{pk}_{\mathsf{P}'})$ to P. $\mathsf{pk}_{\mathsf{P}'}$ is set to $\perp$ if no record $(\mathsf{P}', \mathsf{sk}_{\mathsf{P}'}, \mathsf{pk}_{\mathsf{P}'})$ exists.

Secure messaging functionality This functionality assures that two parties can securely – confidentially and robustly – exchange messages.

Secure message transfer functionality $\mathbf{F}_{smt}$

On input $(\texttt{send}, \mathsf{P}', m)$ from a party P:

- If both parties P and P' are honest, output $(\texttt{sent}, \mathsf{P}, m)$ to P' and $(\texttt{sent}, \mathsf{P}, \mathsf{P}', |m|)$ to $\mathcal{A}$.
- Otherwise, output $(\texttt{sent}, \mathsf{P}, \mathsf{P}', m)$ to $\mathcal{A}$.

Anonymous messaging functionality The functionality behaves similarly to $\mathbf{F}_{smt}$ however it offers extra privacy as the recipient does need to learn who is the message sender. Importantly, the receiver is able to send a response to the sender using the functionality.

Anonymous messaging functionality $\mathbf{F}_{amt}$

On input $(\texttt{sendAnonymously}, \mathsf{P}', mid, m)$ from a party P:

- If both parties are honest, output $(\texttt{sentAnonymously}, mid, m)$ to P' and $(\texttt{sentAnonymously}, \mathsf{P}', |m|)$ to the adversary. Store $(\mathsf{P}, \mathsf{P}', mid)$
- If the receiving party is corrupted, output $(\texttt{sentAnonymously}, \mathsf{P}', mid, m)$ to $\mathcal{A}$.
- If the sending party is corrupted, output $(\texttt{sentAnonymously}, \mathsf{P}, \mathsf{P}', m, mid)$ to $\mathcal{A}$.

On input $(\texttt{return}, mid, m')$ from party P',

– Check whether there is $(\mathsf{P}, \mathsf{P}', mid)$ stored and abort if that is not the case;
– Else, send $(\texttt{returned}, mid, m')$ to P.

MPC PRF functionality The functionality allows the parties who hold shares of a PRF's secret key to evaluate the function on given input.

PRF functionality $\mathbf{F}_{mpcprf}$

The function interacts with parties $\mathsf{P}_1, \ldots, \mathsf{P}_\mathsf{p}$ and is parametrized by

– Constants t, p, $\mathsf{maxAccounts}$;
– (t, p)-secret sharing scheme SS,
– PRF function PRF;

Upon input $(\texttt{compute}, \mathsf{k}_i, \mathsf{aux})$ from at least $\mathsf{t}+1$ parties P_i:

– Compute $\mathsf{k} = \mathsf{Reconstruct}(\mathsf{k}_{i_1}, \ldots, \mathsf{k}_{i_{t+1}})$;
– Evaluate $\mathsf{PRF}_\mathsf{k}(x)$ on all $x \in [1..\mathsf{maxAccounts}]$;
– Return the list of outputs to all requesters P_i, $i \in [1..\mathsf{p}]$.

Ledger functionality The functionality describes how ideal execution of a ledger looks like. It allows the parties to add content to the ledger and gives the adversary some control on how the things waiting to be included into the ledger are ordered.

Ledger functionality $\mathbf{F}_{ledger}$

The functionality is parametrized by a predicate valid and starts with empty list L and variable buffer initially set to empty string ε.

Append On input $(\texttt{append}, x)$ from a party P, if $\mathsf{valid}(\mathsf{state}, (\mathsf{buffer}, x)) = 1$ then set $\mathsf{buffer} \leftarrow (\mathsf{buffer}, x)$ and broadcast (x, P).
[**Michal 10.05:** The broadcast is used to model the gas model which reveals the party that makes changes to the ledger.]

Retrieve On input $(\texttt{retrieve})$ from a party P or $\mathcal{A}$, return $(\texttt{retrieve}, L)$ to the requestor if it is an honest party. In case the requestor is a corrupted party, return $(\texttt{retrieve}, L, \mathsf{buffer})$.

Buffer release On input $(\texttt{releaseTransactionBuffer}, f)$ from $\mathcal{A}$, ignore if f is not a permutation. Else, permute the buffer accordingly to f, i.e. set $\mathsf{buffer} \leftarrow f(\mathsf{buffer})$, set $L \leftarrow (L, \mathsf{buffer})$ and $\mathsf{buffer} \leftarrow \varepsilon$.

B Assuring secure SRS generation

In some scenarios one may assume that the numbr of regulators is limited. In that case, the problem of trusted SRS generation seems feasible to handle using the following approach: Assume that the proof system that the regulator $\mathcal{R}$ and the regulated party $\mathcal{P}$ use is Sub-ZK (subversion zero-knowledge, i.e. the zero-knowledge property holds even if the SRS is generated maliciously, cf. [BFS16,ABLZ17,Fuc18]). Each regulator $\mathcal{R}$ creates its own SRS which is then given to $\mathcal{P}$ who verifies whether the SRS provides zero knowledge, and if that is the case, it uses it to produce the proof.

We note that in such scenario both soundness and zero knowledge are preserved. First, soundness holds since the regulator *is interested* in getting valid proofs for valid statements. Thus if it does not want to undermine its efforts, it provides a SRS such that soundness holds and it keeps the corresponding trapdoor secret. Second, zero knowledge holds, since the regulated party $\mathcal{P}$ can simply check the SRS for that property.

C Protocol instantiation with Zeth

User registration

	User		Identity Provider
		/ Public parameters	
		$\mathsf{srs} \leftarrow_\$ \mathsf{NIZK.Setup}(\mathbf{R}_{idis}, 1^\lambda)$	
	/ Generate ID credentials pair		
	$(\mathsf{scred}, \mathsf{pcred}) \leftarrow_\$ \mathsf{GenIDCreds}(1^\lambda)$		
	/ Generate attribute list		
	$\mathsf{AL} \leftarrow \mathsf{GenAttributeList}()$		
	/ Generate PRF key		
	$\mathsf{k} \leftarrow_\$ \mathsf{GenPRFKey}(1^\lambda)$		
	/ Collect ARs public keys		
	$\mathsf{pkARs} \leftarrow (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})$		
	/ Threshold encrypt the PRF key (t out of ar)		
	$\mathsf{c}_{key} \leftarrow_\$ \mathsf{TE.E}^{\mathsf{t},\mathsf{ar}}_{\mathsf{pkARs}}(\mathsf{k})$		
	/ Blind message to get signed by I		
	$\mathsf{m}_{\mathsf{blind}} \leftarrow \mathsf{BS.PrepSign}(\mathsf{pk}_\mathsf{I}, (\mathsf{scred}, \mathsf{k}, \mathsf{AL}))$		
	/ Prove:		
	/ 1. Knowledge of scred		
	/ 2. That scred is used to derive $\mathsf{m}_{\mathsf{blind}}$		
	/ 3. That c_{key} and $\mathsf{m}_{\mathsf{blind}}$ are derived from k		
	$\pi \leftarrow_\$ \mathsf{NIZK.Pro}(\mathsf{srs},$		
	$((\mathsf{pcred}, \mathsf{pkARs}, \mathsf{c}_{key}), (\mathsf{scred}, \mathsf{k}))$		
		$\xrightarrow{(\mathsf{U}, \mathsf{AL})}$	
		$\xrightarrow{(\mathsf{c}_{key}, \mathsf{pcred})}$	
		$\xrightarrow{\mathsf{m}_{\mathsf{blind}}}$	
		$\xrightarrow{\pi}$	
			/ Verify user attributes (abort if fails)
			$\mathsf{Vf}(\mathsf{U}, \mathsf{AL})$
			/ Verify proof of credential derivation (abort if fails)
			$\mathsf{NIZK.Vf}(\mathsf{srs}, \pi, (\mathsf{pcred}, \mathsf{m}_{\mathsf{blind}}, \mathsf{pkARs}, \mathsf{c}_{key}))$
			/ Sign blinded message
			$\sigma_{\mathsf{blind}} \leftarrow \mathsf{BS.BlindSign}(\mathsf{sk}_\mathsf{I}, \mathsf{m}_{\mathsf{blind}})$
			/ Store IPIU to track user data
			$\mathsf{IPIU} \leftarrow (\mathsf{U}, \mathsf{pcred}, \mathsf{AR}_1, \cdots, \mathsf{AR}_t, \mathsf{c}_{key})$
		$\xleftarrow{\sigma_{\mathsf{blind}}}$	
	/ Unblind message		
	$\sigma_{idis} \leftarrow \mathsf{BS.Unblind}(\sigma_{\mathsf{blind}})$		
	/ Store UC		
	$\mathsf{UC} \leftarrow (\mathsf{pcred}, \mathsf{k}, \mathsf{AL}, \sigma_{idis})$		

Get compliance policy

	User		**Policy Assigner (e.g. DApp operator)**
2 :	/ Select the DApp to use		
3 :	DAppID		
4 :	/ Select the level (i.e. how the user wants to use the DApp)		
5 :	$\mathsf{DAppLevelID}$		
6 :		$\xrightarrow{(\mathsf{DAppID}, \mathsf{DAppLevelID})}$	
7 :			$\mathbf{P}_{al} \leftarrow \mathsf{GetAccountPolicy}(\mathsf{DAppID}, \mathsf{DAppLevelID})$
8 :			$\mathbf{P}_{comp} \leftarrow \mathsf{GetTxPolicy}(\mathsf{DAppID}, \mathsf{DAppLevelID})$
9 :			/ In pratice SRSes for the statements (i.e. policies) / to p
10 :			/ that these SRSes are accessible to the user (e.g. via
11 :			/ an open registry of SRSes and associated MPC transcripts)
12 :			$\sigma_{pal} \leftarrow \mathsf{Sig}(\mathbf{P}_{al})$
13 :			$\sigma_{pcomp} \leftarrow \mathsf{Sig}(\mathbf{P}_{comp})$
14 :			/ The signature algorithm is assumed to be EUF-CMA
15 :		$\xleftarrow{((\mathbf{P}_{al}, \sigma_{pal}), (\mathbf{P}_{comp}, \sigma_{pcomp}))}$	

User DApp account creation that satisfies $\mathbf{P}_{al}$

	User		**Relay**
2 :		/ Public parameters	
3 :		$\mathsf{srs} \leftarrow_{\$} \mathsf{NIZK.Setup}(\mathbf{R}_{newacc}, 1^{\lambda})$	
4 :	/ Collect ARs public keys		
5 :	$\mathsf{pkARs} \leftarrow (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})$		
6 :	/ Generate the account registration ID (xth account)		
7 :	$\mathsf{aux}_{acc} \leftarrow \mathsf{PRF}_{\mathsf{k}}(x)$		
8 :	/ Generate the anonymity revokation data		
9 :	$\mathsf{c}_{pcred} \leftarrow_{\$} \mathsf{TE.E}^{\mathsf{t,ar}}_{\mathsf{pkARs}}(\mathsf{pcred})$		
10 :	/ Generate DApp account keypair (e.g. ZETH keypair)		
11 :	$(\mathsf{sk}_{acc}, \mathsf{pk}_{acc}) \leftarrow_{\$} \mathtt{ZETH}.\mathsf{GenAccountKeys}(1^{\lambda})$		
12 :	/ Generate ACI		
13 :	$\mathsf{ACI} \leftarrow (\mathsf{aux}_{acc}, \mathsf{c}_{pcred}, \mathsf{l}, \mathsf{pk}_{acc}, \mathbf{P}_{\mathbf{ACC}})$		
14 :	/ Generate the zk-proof of valid account creation		
15 :	/ Prove:		
16 :	/ 1. User had succesfully registered to l		
17 :	/ 2. That $\mathbf{P}_{\mathbf{ACC}}(\mathsf{AL})$ evaluates to true		
18 :	/ 3. User knows sk_{acc} associated to pk_{acc}		
19 :	/ 4. User knows scred associated to pcred		
20 :	/ 5. $\mathsf{aux}_{acc}, \mathsf{c}_{pcred}, \mathsf{c}_{key}$ are correctly generated		
21 :	$\pi \leftarrow_{\$} \mathsf{NIZK.Pro}(\mathsf{srs}, (\mathsf{ACI}, \mathsf{pkARs}, \mathsf{pk}_{\mathsf{l}}), (\mathsf{scred}, \mathsf{k}, x))$		
22 :	/ Add the generated zkp to the ACI tuple		
23 :	$\mathsf{ACI} \leftarrow \mathsf{ACI}\|\|\pi$		
24 :		$\xrightarrow{\mathsf{ACI}}$	
25 :			/ Add ACI to the Ledger

Revoke account anonymity

	Anonymity revokers		**Identity provider**
	/ Select the target account identifier		
	aux_{acc}		
	/ Find ACI corresponding to aux_{acc} on the ledger		
	$\mathsf{ACI} \leftarrow \mathsf{GetRegisteredACI}(\mathsf{aux}_{acc})$		
	/ Collect ARs public keys		
	$\mathsf{pkARs} \leftarrow (\mathsf{pk}_{\mathsf{AR}_1}, \ldots, \mathsf{pk}_{\mathsf{AR}_{\mathsf{ar}}})$		
	/ Threshold decrypt c_{pcred} ($n >$ t ARs collaborate))		
	$\{\mathsf{s}_i\}_{i\in[n]} \leftarrow \mathsf{TE.Dec}^{t,\mathsf{ar}}_{\mathsf{pkARs}}(\mathsf{sk}_i, \mathsf{c}_{pcred})$		
	$\mathsf{pcred} \leftarrow \mathsf{TE.Comb}^{t,\mathsf{ar}}_{\mathsf{pkARs}}(\mathsf{c}_{pcred}, \{\mathsf{s}_i\}_{i\in[n]})$		
		pcred →	
			/ Recover the IPIU of the user
			$\mathsf{IPIU} \leftarrow \mathsf{GetRegisteredIPIU}(\mathsf{pcred})$
		← IPIU	
		/ The IPIU contains the ID of the user	

Trace user account

	Anonymity revokers		**Identity provider**
			/ Recover the IPIU of the suspected user
			$\mathsf{IPIU} \leftarrow \mathsf{GetRegisteredIPIU}(\mathsf{pcred})$
		← IPIU	
	/ The relevant ARs decrypt the PRF key of the suspected user		
	$\{\mathsf{s}_i\}_{i\in[n]} \leftarrow \mathsf{TE.Dec}^{t,\mathsf{ar}}_{\mathsf{pkARs}}(\mathsf{sk}_i, \mathsf{c}_{key})$		
	$\mathsf{k} \leftarrow \mathsf{TE.Comb}^{t,\mathsf{ar}}_{\mathsf{pkARs}}(\mathsf{c}_{pcred}, \{\mathsf{s}_i\}_{i\in[n]})$		
	/ Evalute the PRF on all account indices via an MPC protocol		
	$\{\mathsf{aux}_{acci}\}_{i\in[\mathsf{maxAccounts}]} \leftarrow \mathsf{PRF}_{\mathsf{k}}(i)_{i\in[\mathsf{maxAccounts}]}$		
	/ All the user's account IDs are revealed		
	/ The ledger can be audited		

Send a transaction

	User		**Relay**	**Policy Assigner**
	/ Create DApp payload (e.g. Zeth proof etc)			
	/ Create proof of tx compliance (i.e. $\mathbf{P_{TX}}$ is satisfied)			
	/ or, seeks approval to the Policy assigner			
		send payload to Relay →		
			/ Relays Tx on-chain	