Balancing Privacy and Accountability in Blockchain Identity Management*

Ivan Damgård[1], Chaya Ganesh[2], Hamidreza Khoshakhlagh[1],
Claudio Orlandi[1], and Luisa Siniscalchi[1]

[1]Concordium Blockchain Research Center, Aarhus University, Denmark
[2]Indian Institute of Science, Bangalore, India

Abstract

The lack of privacy in the first generation of cryptocurrencies such as Bitcoin, Ethereum, etc. is a well known problem in cryptocurrency research. To overcome this problem, several new cryptocurrencies were designed to guarantee transaction privacy and anonymity for their users (examples include ZCash, Monero, etc.).

However, the anonymity provided by such systems appears to be fundamentally problematic in current business and legislation settings: banks and other financial institutions must follow rules such as "Know Your Customer" (KYC), "Anti Money Laundering" (AML), etc. It is also well known that the (alleged or real) anonymity guarantees provided by cryptocurrencies have attracted ill-intentioned individuals to this space, who look at cryptocurrencies as a way of facilitating illegal activities (tax-evasion, ransom-ware, trading of illegal substances, etc.).

The fact that current cryptocurrencies do not comply with such regulations can in part explain why traditional financial institutions have so far been very sceptical of the ongoing cryptocurrency and Blockchain revolution.

In this paper, we propose a novel design principle for identity management in Blockchains. The goal of our design is to maintain privacy, while still allowing compliance with current regulations and preventing exploitations of Blockchain technology for purposes which are incompatible with the social good.

*Research supported by: the Concordium Blockchain Research Center (COBRA), Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC);

Contents

1 Introduction **3**

2 Preliminaries and Building Blocks **7**
2.1 Notation . . . 7
2.2 Pseudorandom Functions . . . 7
2.2.1 Instantiation with Dodis-Yampolskiy PRF. . . . 7
2.3 Blind Signature Schemes . . . 8
2.4 (Ad-hoc) Threshold Encryption Scheme . . . 8

3 System Design **9**
3.1 Entities Involved . . . 9
3.2 Data Objects . . . 10
3.3 Protocols . . . 11
3.4 Informal Analysis of the Design . . . 12

4 ID-layer Formalization **12**
4.1 The UC-Model . . . 12
4.2 The ID Layer Functionality . . . 13
4.3 Issuing Credentials – the Functionality . . . 15

5 Formal Protocols Specifications **16**
5.1 Identity Layer Protocol . . . 16
5.2 Proof of Security for Identity Layer . . . 18
5.3 Credential Issue Protocol . . . 22
5.4 Proof of Security for Issue Protocol . . . 22

6 Putting Everything Together **23**

A Additional Definitions **30**
A.1 Bilinear groups . . . 30
A.2 Commitment Schemes . . . 30
A.3 (Blind) Signature Scheme . . . 31
A.3.1 Instantiation with PS Signature scheme [PS16] . . . 31
A.4 Threshold Encryption Scheme . . . 33
A.4.1 Construction based on Share and Encrypt paradigm. . . . 33
A.5 Secret Sharing Scheme . . . 34
A.6 Non-Interactive Zero-Knowledge Proofs . . . 35

B Implementation Details **36**
B.1 Auxiliary Σ-protocols. . . . 37
B.2 Using Fiat-Shamir for non-interactive Proof . . . 37
B.3 Sigma protocol for proving knowledge of discrete log . . . 37
B.4 Sigma protocol for proving equality of committed value and Elgamal encrypted value 38
B.5 Proof of Equality of Aggregated Discrete Logs & Commitments . . . 38
B.6 Proof of Aggregated Discrete Logs . . . 39

B.7 Proof of Equality for Commitments in Different Groups . . . 39
B.8 Proof of multiplicative relation on committed values . . . 40

C Protocols for ID layer **41**
C.1 Protocol for Π_{issue} . . . 41
C.2 Protocol for $\Pi_{\mathsf{id-layer}}$ . . . 41

D CL Framework **43**
D.1 Definition . . . 43
D.2 Hard Subgroup Membership Problem . . . 44
D.3 PKE scheme under HSM-CL assumption . . . 44

E Transaction Layer **45**
E.0.1 Plaintext Transactions. . . . 46
E.0.2 Encrypted Transactions. . . . 46
E.0.3 Compressing an account. . . . 46

F Universally Composable Non-interactive Zero-Knowledge **47**
F.1 Simulation Extractable NIZK . . . 47
F.2 SNARK-Lifting transformations via C∅C∅ framework [KZM+15] . . . 48

G Standard Ideal Functionalities **48**
G.0.1 Registration Functionality . . . 50
G.0.2 NIZK Functionality . . . 50
G.0.3 Ledger Functionality . . . 50
G.0.4 Functionality for MPC of PRF output . . . 51

1 Introduction

Early applications of blockchain to payment systems such as Bitcoin do not guarantee privacy. In the Bitcoin blockchain, blocks posted on the public ledger consist of transactions, making Bitcoin transparent – transactions are there for everybody to see. However, the identities are pseudonymous, and not tied to real world identities. Consequently, Bitcoin has the property that while the ownership of money is implicitly anonymous, the flow of money is globally visible. While this was perceived to be truly anonymous early on, there have been several works that deanonymize Bitcoin flow by analysing the payment graph [MPJ+13]. To overcome the problem of lack of privacy in the first generation of cryptocurrencies such as Bitcoin, Ethereum, etc., new systems were designed to guarantee transaction privacy and anonymity for their users [BCG+14, PBF+19, FMMO19]. Systems like Zerocash [BCG+14] fully hide both the value inside a transaction, and the sender and receiver identities. Most blockchain use-cases, however, are hindered by complete privacy as they need *accountability* and *identity management*. Privacy-preserving systems like ZCash are not designed with accountability in mind [1]. In order to conform with regulations like "Know your customer" (KYC) and "Anti-money laundering" (AML), a legal authority should be able to learn the value and identities of the parties involved in any transaction; this requirement seems to be at odds

[1] Zcash considers solutions to implement AML and KYC controls [Zca], however this solution requires trust on a single party.

with privacy. The seemingly contradictory requirements of transaction privacy & user anonymity, and regulatory requirements such as KYC/AML imposed on financial and banking institutions is a major hurdle in widespread adoption of the blockchain.

Our Contribution. In this work, we address the problem of balancing accountability with privacy in blockchain-based systems. We propose a new architectural design of an "identity layer" that will provide privacy for its users – that is, no one, observing the network transactions and the status of the Blockchain should be able to learn about the identity of the owner of any account in the system. At the same time, the identity layer achieves accountability in the sense that in the presence of a reasonable suspicion, law-enforcement agencies (or other authorized parties), will be able to access the transaction history of a given user and/or block its funds, in a way similar to what is guaranteed today by traditional financial institutions. We develop cryptographic mechanisms that enhance accountability measures against misuse of the blockchain, while still providing privacy. Towards this end, we employ cryptographic techniques to design provably secure protocols, with both privacy and accountability guarantees. We prove the security of our constructions in the Universal Composability (UC) framework. We provide a high-level overview of the design of the system, and then discuss the techniques and cryptographic tools used. We believe that such an identity layer design will make Blockchain and cryptocurrencies more attractive for regulators, public institutions and traditional businesses which are interested in complying with existing legislation. In fact, the identity layer of Concordium[2], an upcoming major Blockchain project, is based on the design presented in this paper.

Overview of the System. In the proposed system, the identity and credentials of each participant in the network are initially verified and stored by authorized parties called *Identity Providers* (IPs). Each user can open a limited number of accounts where an account has an identifier that is derived from a PRF applied to a value that is between 1 and the maximal number of accounts, say n. The PRF key K is held by the user. When a user registers with an IP, K is encrypted through a threshold encryption scheme, and this ciphertext is stored with the IP. This is set up such that an appropriate number of *Anonymity Revokers* (ARs) would be able to decrypt. Standard anonymous credentials are used to certify additional attributes of the user.

When a user creates an account, they prepare some data to be published on the blockchain. This includes a threshold encryption of the account holder's public key (that was also stored with the IP at registration time). It also includes zero-knowledge proofs that the attributes the user chooses to publish in the account have been signed by the IP, and that the account identifier has been correctly computed. Thus, an account may contain complete identification of the account holder, if the user chooses to include it, or it may reveal less information, for instance the citizenship and age of the account holder.

Finally, an account includes various account specific public keys. Using the corresponding secret keys, the account holder can then perform transactions anonymously in the network. Depending on the key material included in accounts, several different ways to do transactions can be realized – this is a problem orthogonal to that of implementing the identity layer, and we give some informal examples of how this could be done in Section E.

If it is suspected that an account is used for fraudulent purposes, the encrypted account information can be decrypted by a qualified set of the ARs, and an anonymous account can be linked

[2] concordium.com

(via the public key) to an id provided by the IP. On the other hand, if a particular user is suspected of fraud, the IP can provide its record for this user, and a qualified set of ARs can decrypt the information to learn the PRF key K. Now, they can generate the set of all values $\mathsf{PRF}_{\mathsf{K}}(x)$ for $x = 1, \ldots, n$ which are all the possible values for an account identifier. One can then identify all accounts of the user by searching the blockchain for accounts with these identifiers. Privacy is therefore guaranteed for all users, except those whose anonymity is revoked by a sufficient number of ARs.

Note that the above also implies that we have a mechanism for preventing a user from opening an unbounded number of accounts using a single certificate from the IP: if this were possible, it would open the door for attacks where an individual registers with an IP and then allow other individuals to open accounts in their name, perhaps after payment of a small sum of money. On the other hand, we do not want the account holder to have to interact with the IP for every new account it wants to create, as this would affect efficiency. While the concrete number of accounts allowed per user is an implementation dependent parameter, our technique allows to achieve a reasonable tradeoff. The zero-knowledge proofs force the user to compute the account id's correctly, which only allows n different id's. Thus, if one attempts to open more accounts than allowed, this must result in a pre-existing account identifier, and the Blockchain will reject it.

We prove security of the system when either any number of account holders are actively corrupt, or when the identity providers are semi-honest corrupt. Note that, similar to certification authorities in standard PKI, we need some trust in the IPs: a malicious identity provider (equivalently, a malicious account holder colluding with a semi-honest IP and therefore learning the key), could produce certificates containing false identities, therefore undermining the system. Finally, depending on which properties we want to emphasize, we could tolerate different corruption levels among the anonymity revokers. Thus our system is secure in the presence of actively corrupt users and a threshold number of passively corrupt anonymity revokers; or, in the presence of passively corrupt identity-provider and a threshold number of passively corrupt anonymity revokers. In our design it is paramount that the service provided by the anonymity revokers to be available, and we want to emphasize privacy. Thus, we opt for assuming a majority of semi-honest ARs. Using standard methods, we could instead tolerate a minority of actively corrupted ARs.

Overview of technical ideas. We use cryptographic schemes such as Pedersen commitments [Ped92], Dodis-Yampolskiy PRF [DY05], Pointcheval-Sanders (PS) signature scheme [PS16], CL encryption scheme [CL15]. We use zkSNARKs in combination with commitments and signatures in the spirit of [AGM18, CFQ19]: the PS blind signature we use is defined using groups of a certain prime order, and when a user proves knowledge of a signature, the message that is signed is committed to using a Pedersen-type commitment in such a group. Now, we can use standard sigma-protocols to provide commitments to individual attributes of the user in the same group, and finally use SNARKs on committed messages to show statements such as "the age attribute of the user is a number greater than 18". In this way, we only need to use SNARKs on rather small circuits, and we can achieve much greater efficiency than if we had to convert large statements involving, e.g, group operations into a Boolean circuit to be evaluated inside the SNARK. In this way, creating an account requires a constant number of exponentiations (i.e., independent of the security parameter), and likewise, the number of group elements in an account is constant.

We provide a generic lifting transformation for Fiat Shamir NIZKs for DL-languages into UC NIZKs. While such a transformation by encrypting the witness under a key that is part of the CRS

(and the secret key part of the CRS trapdoor) is folklore [DP92], using the CL encryption scheme allows us to efficiently prove statements about values in the exponent, which is novel to the best of our knowledge.

Related Work. The cryptographic tools used in building our solution, like commitment schemes, blind signatures, zero-knowledge proofs, and threshold encryption are based on anonymous credentials technology. Anonymous credentials [Cha83] allow a party to prove to a verifier that one has a set of credentials without revealing anything beyond this fact. Revocable anonymity [CMS96, KTY04] allows a trusted third party to discover the identity of all otherwise anonymous participants. Conditional anonymity requires that a user's transactions remain anonymous until certain conditions are violated [CHL05, DDP06, CHK+06]. In [DDP06], an unclonable identification scheme is introduced, that is, roughly, an identification scheme where honest users can identify themselves anonymously as members of a group, but where clones of users can be detected and have their identities revealed if they identify themselves simultaneously. This was extended from one-time authentication to n-times anonymous authentication in [CHK+06] where a certain number of unlinkable accounts are derived that can later be efficiently traced. The works of [TFS04, NS05, ASM06, TS06] addressed related problems of allowing a user to show a credential anonymously and unlinkably up to n times to a particular verifier. The potential for abuse of unconditional anonymity by misbehaving users has been articulated in the context of group signatures. In a group signature scheme, each group member can sign a message on behalf of a group such that anyone can verify that the group signature is produced by someone in the group, but not who exactly. Our idea for identifying all accounts of a user in case of revocation by using a PRF to generate account identifiers is reminiscent of the work of traceable signatures [Cho09] that enable a tracing agent to identify all signatures produced by a particular member. The idea of deriving a certain number of unlinkable accounts that can later be efficiently traced has been used in various forms in the anonymous credential literature [CHK+06, ASMC12, Cho09] for the purposes of balancing accountability and anonymity.

Unfortunately none of the previous works seem to fit our intended use case, which motivated us to design the system described in this paper. Moreover, the toolbox of efficient tools available to the protocol designer has grown in recent years (e.g., the CL encryption scheme, advances in SNARKs, etc.), which also motivates exploring new designs.

The zkLedger protocol [NVV18] is an asset transfer scheme that hides transaction amounts and sender-receiver relationship, and supports auditing. The protocol is for a setting where the transacting parties are banks, and requires the participation of the banks for an audit to take place. The work of [ACC+20] presents a privacy-preserving token management system that supports auditing in permissioned blockchains. The system of [ACC+20] is in the UTXO framework, where users own tokens that are certified, and prove ownership of tokens in a privacy-preserving manner. In contrast, we work in the account-based model; and our design is modular – the identity layer is separate from the transaction layer. One main difference of our work from the works of [NVV18] and [ACC+20] is that while both these works assume that the entire system is permissioned, again our design is more modular: our ID layer obviously assumes that IPs and ARs are known (and trusted to some extent) and is therefore in some sense permissioned. However, the ID layer can work on top of the consensus mechanism of a permissionless blockchain, i.e., any blockchain that can be abstracted using the ledger functionality $\mathcal{F}_{\mathsf{ledger}}$.

Finally, Solidus [CZJ+17] is a privacy-preserving system that allows customers of financial institutions (e.g., banks) to transfer assets and ensures that only the banks of the sender and

receiver can learn the transaction details. While there is no explicit audit functionality in Solidus, banks can reveal the content of a suspicious transaction to the authorized auditors. However this approach requires to trust a single party (i.e., the bank).

2 Preliminaries and Building Blocks

This section defines our notation and introduces the cryptographic schemes we use in our construction.

2.1 Notation

For any positive integer n, $[n]$ denotes the set $\{1, \ldots, n\}$. We write $f(\lambda) \approx_\lambda g(\lambda)$ if the difference between f and g is negligible in λ. We use DPT (resp. PPT) to mean a deterministic (resp. probabilistic) polynomial time algorithm. We denote by $Y \leftarrow\!\!\$\ \mathsf{F}(X)$ a probabilistic algorithm F that on input X outputs Y. Similarly, notation $Y \leftarrow \mathsf{F}(X)$ is used for a deterministic algorithm with input X and output Y. All adversaries will be stateful. We use the identifier AH for account holder, IP for identity provider and AR for anonymity revoker. By an identifier, we mean an arbitrary string that uniquely identifies a party. Throughout the paper, $\mathbb{F}_q$ will denote the field with q elements.

2.2 Pseudorandom Functions

We recall the standard notion of pseudorandom functions.

Definition 2.1 (PRF)**.** *Let $\mathit{PRF}\colon \mathcal{K} \times \mathsf{D} \to \mathsf{R}$ be a family of functions and let Γ be the set of all functions $\mathsf{D} \to \mathsf{R}$. We say that PRF is a pseudorandom function (PRF) (family) if it is efficiently computable and for all PPT distinguishers $\mathcal{D}$*

$$\left|\Pr\left[\mathsf{K} \leftarrow\!\!\$\ \mathcal{K}, \mathcal{D}^{\mathit{PRF}_{\mathsf{K}}(\cdot)}(1^\lambda)\right] - \Pr\left[g \leftarrow\!\!\$\ \Gamma, \mathcal{D}^{g(\cdot)}(1^\lambda)\right]\right| \approx_\lambda 0.$$

We define a weak notion of PRF robustness, meaning that is should be hard to find a key that produce collisions with the PRF evaluation of an honest user. Our definition is similar to the one in [FOR17], but here one of the two keys is chosen honestly.

Definition 2.2 (Weakly Robust PRF)**.** *A PRF is weakly robust if:*

$$\Pr[\mathsf{K} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda), (x^*, \mathsf{K}^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathit{PRF}_{\mathsf{K}}(\cdot)}(1^\lambda) : \exists (x, y) \in \mathcal{Q}, \mathit{PRF}_{\mathsf{K}^*}(x^*) = y] \approx_\lambda 0$$

where $\mathcal{Q}$ is the set of inputs/outputs of the oracle available to the adversary.

2.2.1 Instantiation with Dodis-Yampolskiy PRF.

We use the PRF of Dodis and Yampolskiy [DY05] that operates in a group $\mathbb{G}$ of order q with generator g. On input x and the PRF key $\mathsf{K} \leftarrow\!\!\$\ \mathbb{F}_q$, $\mathsf{PRF}_{\mathsf{K}}(x) = g^{1/(\mathsf{K}+x)}$. This is shown to be pseudorandom under the Decisional Diffie-Hellman Inversion assumption in group $\mathbb{G}$. Note that the security holds only for small domains, namely inputs that are slightly superlogarithmic in the security parameter, but this is sufficient for our work, as the maximum number of accounts a user can open is less than a constant $\mathsf{Max}_{\mathsf{ACC}}$. It can also be easily shown that the Dodis-Yampolskiy

PRF is weakly robust: using the PRF assumption, we can replace the output of the PRF oracle with random group elements. If the adversary outputs an input x^* and key K^* that are compatible with one of the output of the oracles, we can compute the discrete logarithm of that element as $1/(\mathsf{K}^* + x^*)$.

2.3 Blind Signature Schemes

We adapt the notation of [SU12] to two-round blind signature schemes.

Definition 2.3 (Blind Signature Schemes)**.** *An interactive signature scheme between a signer S and user U consists of a tuple of efficient algorithms* $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ *where*

- $\mathsf{Setup}(1^\lambda)$*, on input the security parameter* 1^λ *outputs* pp*, which is given implicitly as input to all other algorithms, even when omitted.*
- $\mathsf{KeyGen}(\mathsf{p})$*, on input the public parameter* p *generates a key pair* $(\mathsf{sk}, \mathsf{pk})$ *for security parameter* λ*.*
- $\mathsf{Sign}_1(\mathsf{pk}, m)$*, which is run by U, takes as input* pk *and a message* $m \in \{0,1\}^*$ *and outputs* $\mathtt{sign}_1$ *and* ω *(w.l.o.g. ω can be thought of as the randomness used to run* Sign_1*).*
- $\mathsf{Sign}_2(\mathsf{sk}, \mathtt{sign}_1)$*, which is run by S, takes as input* sk *and* $\mathtt{sign}_1$ *and outputs* $\mathtt{sign}_2$*.*
- $\mathsf{Unblind}(\mathtt{sign}_2, \omega)$*, which is run by U, takes as input* $\mathtt{sign}_2, \omega$ *and outputs* σ*.*
- $\mathsf{VerifySig}(\mathsf{pk}, m, \sigma)$ *outputs a bit.*

Remark 2.4. Note that a blind signature scheme implicitly defines a normal signature scheme as well, where the signing algorithm $\mathsf{Sign}(\mathsf{sk}, m)$ simply emulates a blind signature protocol and outputs the resulting signature σ.

The correctness property of the scheme requires that the following holds: for any $(\mathtt{pp}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathtt{pp})$, any message $m \in \{0,1\}^*$, if $(\mathtt{sign}_1, \omega) \leftarrow\!\!\$\ \mathsf{Sign}_1(\mathsf{pk}, m)$, $\mathtt{sign}_2 \leftarrow\!\!\$\ \mathsf{Sign}_2(\mathsf{sk}, \mathtt{sign}_1)$, $\sigma = \mathsf{Unblind}(\mathtt{sign}_2, \omega)$ then $\mathsf{VerifySig}(\mathsf{pk}, m, \sigma) = 1$ with overwhelming probability over $\lambda \in \mathbb{N}$.

We require the standard notion of *existential unforgeability under chosen message attacks* ($\mathsf{EUF\text{-}CMA}$) [GMR88]. The blind signature scheme we use should additionally satisfy two properties, namely *Blindness* and *simulatability*, where the second is an ad-hoc definition required for our UC proof of security that ensures the existence of an additional simulation algorithm Sim that can simulate $\mathtt{sign}_2$. The formal definition of these properties can be found in Appendix A.3.

2.4 (Ad-hoc) Threshold Encryption Scheme

We recall the definition of an ad-hoc threshold encryption scheme here.

Definition 2.5. *A* (n, d)*-threshold encryption scheme* $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{ShareDec}, \mathsf{TCombine})$ *over message space M consists of the following algorithms:*

- $\mathsf{TKeyGen}(1^\lambda)$ *is a randomized key generation algorithm that takes the security parameter λ as input and returns a private-public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

- $\mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\vec{\mathsf{pk}}_R}(m)$, *a probabilistic encryption algorithm that encrypts a message* $m \in M$ *to a set of public keys* $\vec{\mathsf{pk}}_R = \{\mathsf{pk}_i\}_{i \in R}$ *in such a way that any size* $\mathsf{d}+1$ *subset of the recipient set should jointly be able to decrypt. We sometimes write* $\mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\vec{\mathsf{pk}}_R}(m; r)$ *when we want to be able to fix the value of the randomness* r *to a specific value.*
- $\mathsf{ShareDec}^{\mathsf{n},\mathsf{d}}_{\vec{pk}_R,\mathsf{sk}_i}(ct)$, *on input a ciphertext* ct *and a secret key* sk_i, *outputs a decryption share* μ_i.
- $\mathsf{TCombine}^{\mathsf{n},\mathsf{d}}_{\vec{pk}_R}(ct, \{\mu_i\}_{i \in I})$, *a deterministic algorithm that takes a subset* $I \subset [n]$ *with size* $\mathsf{d}+1$ *of decryption shares* $\{\mu_i\}_{i \in I}$ *and outputs either a message* $m \in M$ *or* $\perp$.

We use the static security definition of Reyzin et al. [RSY18] for threshold encryption schemes which requires two properties, namely *static semantic security* and *partial decryption simulatability* as defined in Appendix A.4.

Definition 2.6. *A threshold encryption scheme* $\mathsf{TE} = (\mathsf{KeyGen}, \mathsf{TEnc}, \mathsf{ShareDec}, \mathsf{TCombine})$ *is* (n, d)*-statically secure if it is both* (n, d)*-statically semantically secure (Definition A.5) and* (n, d)*-partial decryption simulatable (Definition A.6)*

The definitions of Commitments scheme, Secret Sharing and Zero-Knowledge can be found in Appendix A.

3 System Design

We give a high-level overview of the design of the identity layer in terms of the entities involved, data objects and protocols between the entities.

3.1 Entities Involved

The following entities are involved in our design:

- Account Holders (AH): those are individuals who hold accounts on the block-chain. We assume AHs possess some mean for performing legal identification (e.g., a passport), in the country where they live. They are interested in opening accounts and performing transactions on the blockchain but, before doing so, they have to register with an Identity Provider (IP).
- Identity Provider (IP): an identity provider is an entity that, as the name suggests, can provide a digital identity to an AH. The identity provider "authorizes" a user to open accounts on the blockchain, and therefore to perform transactions. Jumping ahead, when observing transactions on the blockchain, it should not be possible to find out the identity of an AH (not even for the IP itself), while everyone should be able to see which IP has authorized a given account, thus creating trust in the account.
- Anonymity Revoker (AR): anonymity revokers are parties which are involved in case where law-enforcement or other authorized entities need to be able to extract the identity of the owner of some account on the blockchain. We can make threshold assumptions on the AR and e.g., require that at least $\mathsf{d}+1$ ARs must give an approval before the anonymity of a user is revoked.

3.2 Data Objects

We now describe the data objects that are held by the entities.

Account Holder Certificate (AHC*).* After an account holder registers with an identity provider, the AH obtains a certificate containing:

- A public identity credential $\mathsf{IDcred}_{\mathtt{PUB}}$ and a secret identity credential $\mathsf{IDcred}_{\mathtt{SEC}}$.
- A key K for a pseudorandom function PRF.
- One or more attribute lists AL such as some identifier, age, citizenship, expiration date, etc.
- A signature on $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL})$ that can be checked using $\mathsf{pk}_{\mathsf{IP}}$. A valid signature proves that an AH with attributes as in AL has registered with IP and has proved knowledge of $\mathsf{IDcred}_{\mathtt{SEC}}$ corresponding to $\mathsf{IDcred}_{\mathtt{PUB}}$.

Account Creation Information (ACI*).* Given an AHC, an account holder can create new accounts and post the corresponding ACI on the ledger, containing:

- $\mathsf{RegID}_{\mathsf{ACC}}$, an account registration ID. This is defined to be $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$ where K is a key held by AH and signed by the IP, and where the account in question is the x'th account opened by the AH based on a given AHC. If AH behaves honestly, then $\mathsf{RegID}_{\mathsf{ACC}}$ is unique for the account, and $x \leq \mathsf{Max}_{\mathsf{ACC}}$. The latter condition is enforced by the proof below, the former can be checked publicly.
- Anonymity revocation data: this is a threshold encryption $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathtt{PUB}})$, where any subset of size $\mathsf{d}+1$ of anonymity revokers are able to decrypt E_{ID} and obtain $\mathsf{IDcred}_{\mathtt{PUB}}$.
- The identity IP of the identity provider who did the signature in the AHC used for this account.
- An account specific public key $\mathsf{pk}_{\mathsf{ACC}}$. It will be used, for instance, to verify transactions related to the account.
- A policy P, which asserts some information about the attribute list AL.
- A proof π that can be checked using $\mathsf{pk}_{\mathsf{IP}}$ and verifies that ACI can only be created by an AH that has obtained an AHC from IP, such that $\mathsf{P}(\mathsf{AL}) = \top$, where AH knows the secret keys corresponding to $\mathsf{pk}_{\mathsf{ACC}}$, as well as $\mathsf{IDcred}_{\mathtt{SEC}}$ corresponding to the $\mathsf{IDcred}_{\mathtt{PUB}}$ that was presented to the IP, and where $\mathsf{RegID}_{\mathsf{ACC}}, \mathsf{E}_{\mathsf{ID}}$ and $\mathsf{E}_{\mathsf{RegID}} = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{K})$ are correctly generated.

Identity Provider's information on Account Holder (IPIAH*).* This is the data record that the IP stores after an AH has registered. It contains:

- The name AH of the account holder and its public identity credential $\mathsf{IDcred}_{\mathtt{PUB}}$.
- A set of anonymity revokers $\mathsf{AR}_1, ..., \mathsf{AR}_{\mathsf{n}}$ with public keys $\mathsf{PK}_{\mathsf{AR}}$ and an encryption $\mathsf{E}_{\mathsf{RegID}} = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{K})$. Here, K is the PRF key chosen by the AH at registration time.

3.3 Protocols

The following are the main protocols in our design.

Account Holder Registration. The protocol takes place between an IP and an AH who owns a key pair $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{IDcred}_{\mathtt{PUB}})$ and an attribute list AL. At the end of the protocol, the AH receives an AHC and the IP obtains a IPIAH as described above. The AH sends their attribute list AL to IP and proves (via non-cryptographic means) their identity to IP. More concretely, this means that the IP must verify that the entity it is talking to indeed has the name AH and hence it received AL from the correct entity. It should also verify that the attributes in AL are correct w.r.t. the AH. The AH also sends to IP their public key $\mathsf{IDcred}_{\mathtt{PUB}}$ and an encryption $\mathsf{E}_{\mathsf{RegID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{K})$ where K is a PRF key. Next, AH and IP engage in a blind signature scheme, which allows AH to receive a signature on $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL})$ that is generated under the secret key $\mathsf{sk}_{\mathsf{IP}}$ of the IP. In addition, AH proves (cryptographically, in ZK) that they know $\mathsf{IDcred}_{\mathtt{SEC}}$ corresponding to $\mathsf{IDcred}_{\mathtt{PUB}}$, that the same $\mathsf{IDcred}_{\mathtt{SEC}}$ was input to the blind signature, and that the encryption contains the same K that was input to the blind signature scheme. IP stores $\mathsf{IPIAH} = (\mathsf{ID}_{\mathsf{AH}}, \mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}}, \mathsf{AR}_1, \ldots, \mathsf{AR}_{\mathsf{n}})$.
Create New Account. An account holder AH wants to create an account that satisfies some policy P (e.g., above 18, resident in country X, etc.). They take as input an AHC, a policy P and the public key $\mathsf{pk}_{\mathsf{AR}}$ of one (or more) anonymity revoker(s) with name AR. At the end, AH produces some ACI that can be posted to the blockchain. They also need to store secret key $\mathsf{sk}_{\mathsf{ACC}}$ that is specific to the account. The protocol works as follows: AH generates an account key pair $(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{sk}_{\mathsf{ACC}})$ and an encryption of their public identity credential $\mathsf{IDcred}_{\mathtt{PUB}}$ under the public key of the anonymity revokers' $\mathsf{PK}_{\mathsf{AR}}$, i.e., $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathtt{PUB}})$. Next, AH calculates $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$, where we assume this is the x'th account that is opened using the AHC that is input. At last, AH produces a a non-interactive zero-knowledge (NIZK) proof of knowledge π for statement

$$st = (\mathsf{P}, \mathsf{E}_{\mathsf{ID}}, \mathsf{RegID}_{\mathsf{ACC}}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}})$$

using secret witness

$$w = (\sigma, \mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL}, \mathsf{sk}_{\mathsf{ACC}})$$

for the relation $R(st, w)$ that outputs $\top$ if:

1. σ is a valid signature under $\mathsf{pk}_{\mathsf{IP}}$ for a message of form $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL})$.
2. AL satisfies the policy i.e., $\mathsf{P}(\mathsf{AL}) = \top$.
3. $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$ for some $x \leq \mathsf{Max}_{\mathsf{ACC}}$.
4. $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathtt{PUB}})$.
5. $(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{sk}_{\mathsf{ACC}})$ is a valid key pair.

Let $\mathsf{ACI} = (\mathsf{RegID}_{\mathsf{ACC}}, \mathsf{E}_{\mathsf{ID}}, \mathsf{AR}_1, ..., \mathsf{AR}_{\mathsf{n}}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{P}, \pi)$.
Revoke Anonymity of Account. Revocation of the anonymity of an account can be done by at least $\mathsf{d}+1$ of the n ARs involved in the set-up of the account, working together with the IP with whom the AH registered. The input is an account identifier $\mathsf{RegID}_{\mathsf{ACC}}$ and the output is the name AH of the account holder. The protocol proceeds as follows: Given an account $\mathsf{RegID}_{\mathsf{ACC}}$ whose anonymity needs to be revoked, the ARs find the ACI containing $\mathsf{RegID}_{\mathsf{ACC}}$ on the blockchain, collaborate

to decrypt E_{ID} and learn $\mathsf{IDcred}_{\mathtt{PUB}}$. The registration information also contains the public name IP of the identity provider who registered $\mathsf{IDcred}_{\mathtt{PUB}}$. The ARs contact this IP who then locates the $\mathsf{IPIAH} = (\mathsf{AH}, \mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}}, \mathsf{AR}_1, ..., \mathsf{AR}_n)$ record that contains the $\mathsf{IDcred}_{\mathtt{PUB}}$ that was decrypted. This record also includes AH, thus IP and the set of ARs have now identified the AH.
Trace accounts of User. If a user with a given name AH is suspected of engaging in illegal activities, the IP and a set of at least $\mathsf{d}+1$ ARs can identify all accounts of that user. The IP searches its database to locate the $\mathsf{IPIAH} = (\mathsf{AH}, \mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}}, \mathsf{AR}_1, ..., \mathsf{AR}_n)$ containing the relevant AH. This record also contains the names of the relevant ARs. A qualified set of these could decrypt the $\mathsf{E}_{\mathsf{RegID}}$ to learn the PRF key K and generate all values $\mathsf{PRF}_{\mathsf{K}}(x)$ for $x = 1, \ldots, \mathsf{Max}_{\mathsf{ACC}}$ in public. However, due to technicalities in the security reduction, this would require the PRF to satisfy some form of "selective opening attack" security. Instead, we let the ARs decrypt the ciphertext and evaluate the PRF on $x = 1, \ldots, \mathsf{Max}_{\mathsf{ACC}}$ inside an MPC protocol, so that K is never revealed to anyone. Either way, the produced values are all the possible values for $\mathsf{RegID}_{\mathsf{ACC}}$ that the AH could have used to form valid accounts, so one can now search the blockchain for accounts with these registration IDs.

3.4 Informal Analysis of the Design

If an AH misbehaves and opens more accounts than they are allowed to, this must result in two or more accounts with the same $\mathsf{RegID}_{\mathsf{ACC}}$. This can be publicly detected by the blockchain, and the second account will be discarded. We note that for this to work, we assume that incentives have been created so that some parties will indeed observe the duplicates and alert the relevant entities. Moreover, the construction satisfies *revocability* and *traceability*, meaning a malicious AH cannot create a valid account such that the anonymity revokers together with the identity provider are unable to revoke its anonymity or trace it. This follows from the soundness of the underlying zero-knowledge proofs which imply $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathtt{PUB}})$ and $\mathsf{E}_{\mathsf{RegID}} = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{K})$. Thus, any subset of size $\mathsf{d}+1$ of anonymity revokers can decrypt E_{ID} (resp. $\mathsf{E}_{\mathsf{RegID}}$) and revoke the AH's anonymity (resp. trace all the AH's accounts). Lastly, due to the security of the underlying PRF and the threshold encryption scheme and also the ZK property of the proof $\pi \in \mathsf{ACI}$, our design supports *anonymity* of the account holders, in the sense that a malicious identity provider even by cooperating with d anonymity revokers and other dishonest account holders cannot link a valid account to an account holder. Since we are using a Blockchain e.g., an imperfect bulletin board, we also need to worry that a malicious AH can't "rush" and steal an honest user account number by maliciously choosing a PRF key K which "hits" some of the account numbers of the honest users which have not yet been finalized by the Blockchain. In order to do this we define and use a *weakly robust* PRF.

4 ID-layer Formalization

4.1 The UC-Model

We use the UC-security [Can01] framework with static corruption. In the following, the reader is assumed to be familiar with the basic concepts of UC security and is referred to [Can01] for a more detailed description.

Let $\mathcal{Z}$ denote the environment. For a protocol Π and an adversary $\mathcal{A}$, we write $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ to denote the ensemble corresponding to the protocol execution. For an ideal functionality $\mathcal{F}$

and a simulator $\mathcal{S}$, we write $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ to denote the distribution ensemble of the ideal world execution. We say that a protocol Π UC-realizes a functionality $\mathcal{F}$ if for all PPT adversaries $\mathcal{A}$ corrupting a subset of parties, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, the ensembles $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable. In the rest of the section we describe the main ideal functionalities in our construction. We defer other (standard) ideal functionalities used by our protocol in Appendix G.

4.2 The ID Layer Functionality

The functionality $\mathcal{F}_{\mathsf{id\text{-}layer}}$ captures the security properties offered by the design of our identity layer while hiding the implementation details. After the functionality is initialized, it allows identity providers IP to issue credentials to account holders AH based on their attribute lists AL. At the level of the ideal functionality, a credential is just a pointer to a record storing the tuple $(\mathsf{AH}, \mathsf{IP}, \mathsf{AL})$. Armed with a credential, an AH can create up to $\mathsf{Max}_{\mathsf{ACC}}$ accounts. When creating an account, the AH can choose a predicate P of their attributes to be made public (e.g., "I am over 18, I am resident of country X" etc.) which, together with the IP who authorized this account, are the only information which are made public. We capture this by having the functionality leak only the fact that an account was created and not the identity of the AH.[3] Moreover, when creating an account, the AH also registers a key-pair associated to this account. The functionality is parametrized by any key-pair relation, which allows our ideal functionality to be used as a building block in more complex protocols, where the AH then can use those keys for authentication, encryption, etc. Our functionality also exposes some of the details about the underlying ledger on top of which it is implemented, thus new accounts are added to a buffer which can be permuted by the adversary before becoming finalized. This is inevitable as we run this on top of a ledger which has the same properties. The final two commands of the ideal functionality, revoke and trace, allow a qualified set of anonymity revokers AR and an IP to respectively disclose the AH behind a given account, or to find all accounts belonging to a certain AH.

Functionality Identity layer $\mathcal{F}_{\mathsf{id\text{-}layer}}$

We assume that $\{\mathsf{IP}_1, \ldots, \mathsf{IP}_\mathsf{m}, \mathsf{AR}_1, \ldots, \mathsf{AR}_\mathsf{n}\}$ is the set of identifiers for identity providers and anonymity revokers. The functionality is parameterized by values m, n and threshold d, together with an NP (key-pair) relation $\mathcal{R}_{\mathsf{ACC}}$ such that when parties input `CreateACC`, they also specify a key-pair and the functionality verifies if the key-pair satisfies $\mathcal{R}_{\mathsf{ACC}}$. Moreover, the functionality maintains the following initially empty records: Count, where Count[cid] counts the number of accounts created by certificate cid, and two records Cert and ACC, respectively for keeping track of certificates and accounts and a list L of public account information.

[3] Note that the environment provides all inputs and sees all outputs. It can therefore observe that an account is created right after it instructed an account holder to create an account, and can make the connection between the two. This corresponds to the fact that in a real application an adversary may know that in a long time interval, only one user creates an account, and so the next account that shows up on chain must belong to that user. Of course, our system cannot prevent this - the best we can do is to make sure that the account itself is anonymous. This follows in our model because the ideal adversary - the simulator - will not learn the identity of the holder and will still have to produce account information which are indistinguishable from the real protocol, thus proving that the account information leaks no information about its holder.

Initialize

On $(\texttt{INITIALIZE})$ from party $P \in \{\mathsf{IP}_1, \ldots, \mathsf{IP}_\mathsf{m}, \mathsf{AR}_1, \ldots, \mathsf{AR}_\mathsf{n}\}$, output to $\mathcal{A}$ $(\texttt{INITIALIZED}, P)$.

If all parties have been initialized, store $(\texttt{READY})$.

Issue

On $(\texttt{ISSUE}, \mathsf{IP}, \mathsf{AL})$ from an honest account holder AH (or the adversary in the name of corrupted account holder AH) and input $(\texttt{ISSUE}, \mathsf{AH})$ from identity provider IP:

- If not $(\texttt{READY})$, then ignore.
- If there is already a cid with $\mathsf{Cert}[\mathsf{cid}] = (\mathsf{AH}, \mathsf{IP}, \cdot, \cdot)$, then abort; otherwise, if IP is honest (resp. corrupt), then send $(\texttt{ISSUE})$ (resp. $(\texttt{ISSUE}, \mathsf{AH}, \mathsf{IP}, \mathsf{AL})$) to $\mathcal{A}$.
- Upon receiving $(\mathsf{cid}, \texttt{ISSUE})$ from $\mathcal{A}$, if $\mathsf{cid} = \bot$ (in the case of corrupt IP) or if there already exists cid s.t. $\mathsf{Cert}[\mathsf{cid}] \neq \bot$, then abort. Otherwise, set $\mathsf{Cert}[\mathsf{cid}] \leftarrow (\mathsf{AH}, \mathsf{IP}, \mathsf{AL})$ and output $(\texttt{ISSUED}, \mathsf{cid})$ to AH.

Account Creation

Upon inputs $(\texttt{CreateACC}, \mathsf{cid}, \mathsf{P}, (\mathsf{sk}_{\mathsf{ACC}}, \mathsf{pk}_{\mathsf{ACC}}))$ from honest account holders AH (or the adversary in the name of corrupted account holder AH), if not $(\texttt{READY})$, then ignore. Else, proceed as follows:

- If $\mathsf{Cert}[\mathsf{cid}] = \bot$ then abort, else retrieve $(\mathsf{AH}', \mathsf{IP}, \mathsf{AL}) \leftarrow \mathsf{Cert}[\mathsf{cid}]$.
- Check if $\mathsf{AH}' = \mathsf{AH}$ and $\mathsf{Count}[\mathsf{cid}] < \mathsf{Max}_{\mathsf{ACC}}$ and that AL satisfies the policy P.
- Verify that the key pair $(\mathsf{sk}_{\mathsf{ACC}}, \mathsf{pk}_{\mathsf{ACC}})$ satisfies the relation $\mathcal{R}_{\mathsf{ACC}}$ and abort otherwise.
- Output $(\texttt{CreateACC}, \mathsf{P}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{IP})$ to $\mathcal{A}$.
- Receiving a response $(\texttt{CreateACC}, \mathsf{aid})$ from $\mathcal{A}$, if $\mathsf{aid} = \bot$ or $\mathsf{ACC}[\mathsf{aid}] \neq \bot$, then abort, else do the followings:
  - set $\mathsf{ACC}[\mathsf{aid}] \leftarrow (\mathsf{cid}, \mathsf{P}, \mathsf{sk}_{\mathsf{ACC}})$.
  - set $\mathsf{Count}[\mathsf{cid}] \leftarrow \mathsf{Count}[\mathsf{cid}] + 1$.
  - add the tuple $(\mathsf{aid}, \mathsf{P}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{IP})$ to the account buffer.
- Return $(\texttt{Created}, \mathsf{aid})$ to AH.

Account Buffer Release

Upon input $(\texttt{RELEASE}, \Pi)$ from the adversary $\mathcal{A}$, remove all tuples from the account buffer and add the permuted tuples $(\mathsf{aid}, \mathsf{P}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{IP})$ of accounts to the account list L.

Accounts Retrieve

On $(\texttt{RETRIEVE})$ from an account holder or party $P \in \{\mathsf{IP}_1, \ldots, \mathsf{IP}_\mathsf{m}, \mathsf{AR}_1, \ldots, \mathsf{AR}_\mathsf{n}\}$, output a list including all existing tuples $(\mathsf{aid}, \mathsf{P}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{IP})$ in the account list L.

Revoke

Upon input $(\texttt{REVOKE}, \mathsf{aid})$ from a (possibly corrupt) identity provider IP and a set of (possibly corrupt) anonymity revokers $\{\mathsf{AR}_i\}_{i \in I \subseteq [\mathsf{n}]}$, proceed as follows:

- If $\mathsf{ACC}[\mathsf{aid}] = \perp$ then return $\perp$. Otherwise, retrieve $(\mathsf{cid}, \mathsf{P}, \mathsf{sk}_{\mathsf{ACC}}) \leftarrow \mathsf{ACC}[\mathsf{aid}]$.
- If IP is the same as the identity provider in $\mathsf{Cert}[\mathsf{cid}]$ and $|I| > \mathsf{d}$, then return $(\mathsf{aid}, \mathsf{AH})$ to the IP and $\{\mathsf{AR}_i\}_{i \in I}$. Otherwise, return $\perp$. Moreover, if the identity provider or at least one anonymity revoker is corrupt, output $(\mathsf{aid}, \mathsf{AH})$ to $\mathcal{A}$ as well.

Trace

Upon input $(\texttt{TRACE}, \mathsf{AH})$ from a (possibly corrupt) identity provider IP and a set of (possibly corrupt) anonymity revokers $\{\mathsf{AR}_i\}_{i \in I}$, proceed as follows:

- If there is no record $(\mathsf{AH}, \mathsf{IP}, \cdot, \cdot)$ in Cert, then return $\perp$. Otherwise, retrieve $(\mathsf{AH}, \mathsf{IP}, \cdot) \leftarrow \mathsf{Cert}[\mathsf{cid}]$.
- If $|I| > \mathsf{d}$, return to IP and $\{\mathsf{AR}_i\}_{i \in I}$ the list of all aids such that $\mathsf{ACC}[\mathsf{aid}] = (\mathsf{cid}, \cdot, \cdot)$. Moreover, if the identity provider or at least one anonymity revoker is corrupt, return the list to $\mathcal{A}$ as well.

4.3 Issuing Credentials – the Functionality

We describe here $\mathcal{F}_{\mathsf{issue}}$, an ideal functionality capturing the desired properties of the issue protocol, which allows an identity provider to issue credentials to account holders. Note that the functionality can be seen as an augmented blind signature functionality: the account holder receives a signature (under the secret key of the identity provider) on a secret message m (as in blind signatures) but also on some public auxiliary information aux and on a secret key chosen by the account holder. The identity provider is not supposed to learn m (as in blind signatures), but in addition the identity provider learns a ciphertext which is guaranteed to contain an encryption of m and the public key corresponding to the secret key which is being signed.

Functionality Issue $\mathcal{F}_{\mathsf{issue}}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}$

The functionality is parametrized by an NP relation $\mathcal{R}$ corresponding to the account holders key pair, a signature scheme $\mathsf{SIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerifySig})$ and a (n, d)-threshold encryption scheme $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{TDec})$. We assume that $\{\mathsf{IP}_1, \ldots, \mathsf{IP}_\mathsf{m}\}$ is the set of identifiers for identity providers.

Setup

- Upon input $(\texttt{SETUP})$ from any party, and only once, run $\mathsf{p} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ and return $(\texttt{SETUPREADY}, \mathsf{p})$ to all parties.

Initialize

- Upon input $(\texttt{INITIALIZE}, (\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}}))$ from identity provider IP, ignore if the party is already initialized or if $\texttt{SETUPREADY}$ has not been returned yet. Otherwise, if $(\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}})$ is a valid key pair according to the relation defined by $\mathsf{KeyGen}(\mathsf{p})$, store $(\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}})$ for this party and output $(\texttt{INITIALIZED}, \mathsf{pk}_{\mathsf{IP}}, \mathsf{IP})$ to $\mathcal{A}$.

- If all parties have been initialized, store (READY).

Issue

On input $\Big(\texttt{ISSUE}, (ct, m, r, \mathsf{pk}_{\mathsf{AR}}), \mathsf{aux}, (\mathsf{sk}_{\mathsf{AH}}, \mathsf{pk}_{\mathsf{AH}}), \mathsf{IP}\Big)$ from account holder AH and input $(\texttt{ISSUE}, \mathsf{AH}, \mathsf{pk}_{\mathsf{AR}})$ from identity provider IP:

- If not (READY), then ignore.
- If $(\mathsf{sk}_{\mathsf{AH}}, \mathsf{pk}_{\mathsf{AH}}) \notin \mathcal{R}$ or $ct \neq \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(m; r)$ then abort.
- Otherwise compute $\sigma \leftarrow\!\!\$\ \mathsf{Sign}((\mathsf{sk}_{\mathsf{AH}}, m, \mathsf{aux}), \mathsf{sk}_{\mathsf{IP}})$.
- Output σ to AH and $(\mathsf{pk}_{\mathsf{AH}}, \mathsf{aux}, ct)$ to IP.

5 Formal Protocols Specifications

5.1 Identity Layer Protocol

The protocol $\Pi_{\mathsf{id\text{-}layer}}$ is run by parties interacting with ideal functionalities $\mathcal{F}_{\mathsf{reg}}, \mathcal{F}_{\mathsf{issue}}, \mathcal{F}_{\mathsf{nizk}}, \mathcal{F}_{\mathsf{crs}}$, $\mathcal{F}_{\mathsf{ledger}}$ and $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$. Let $\mathcal{R}$ and $\mathcal{R}_{\mathsf{ACC}}$ be NP relations corresponding to the account holders' key-pair and accounts' key-pair, respectively. Let $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{TDec})$ denote a threshold encryption scheme, PRF a pseudorandom function and $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerifySig})$ a signature scheme. Protocol $\Pi_{\mathsf{id\text{-}layer}}$ proceeds as follows.

Protocol Identity layer $\Pi_{\mathsf{id\text{-}layer}}$

Parameters for ideal functionalities.

- We use a $\mathcal{F}_{\mathsf{ledger}}$ that implements the following VALIDATE predicate: the predicate accepts if the NIZK proof π is valid and if $\mathsf{RegID}_{\mathsf{ACC}}$ has not been seen before.
- We use a $\mathcal{F}_{\mathsf{crs}}$ functionality that outputs the public parameters for the signature scheme and the threshold encryption scheme.

The protocol description for an account holder AH.

- On input $(\texttt{ISSUE}, \mathsf{IP}, \mathsf{AL})$, retrieve the public key $\mathsf{PK}_{\mathsf{IP}}$ of identity provider IP and the vector $\mathsf{PK}_{\mathsf{AR}}$ of all public keys of the anonymity revokers via $\mathcal{F}_{\mathsf{reg}}$ and proceed as follows:
  - Generate a key pair $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{IDcred}_{\mathtt{PUB}})$ satisfying $\mathcal{R}$.
  - Choose a random key K for PRF and compute the encryption $\mathsf{E}_{\mathsf{RegID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{K}; r)$ with randomness r.
  - Call $\mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{BS}}_{\mathsf{issue}}$ on input $\Big(\texttt{ISSUE}, (\mathsf{E}_{\mathsf{RegID}}, \mathsf{K}, r, \mathsf{PK}_{\mathsf{AR}}), \mathsf{AL}, (\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{IDcred}_{\mathtt{PUB}}), \mathsf{IP}\Big)$. After receiving the response σ from $\mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{BS}}_{\mathsf{issue}}$, set $\mathsf{cid} = (\mathsf{IP}, \mathsf{AL}, \mathsf{IDcred}_{\mathtt{SEC}}, \sigma, \mathsf{K})$.
- On input $(\texttt{CreateACC}, \mathsf{cid}, \mathsf{P})$, proceed as follows

- If there is no record $\mathsf{cid} = (\mathsf{IP}, \mathsf{AL}, \mathsf{IDcred}_{\mathtt{SEC}}, \sigma, \mathsf{K})$, then abort.
- Generate an account key pair $(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{sk}_{\mathsf{ACC}})$ satisfying $\mathcal{R}_{\mathsf{ACC}}$.
- Compute $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathtt{PUB}}; \mathsf{r}')$.
- Compute $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$, where this is x'th account that is created using cid.
- Produce a NIZK π by calling $\mathcal{F}_{\mathsf{nizk}}$ for statement

$$st = (\mathsf{P}, \mathsf{E}_{\mathsf{ID}}, \mathsf{RegID}_{\mathsf{ACC}}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}})$$

  using secret witness

$$w = (\sigma, x, r', \mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL}, \mathsf{sk}_{\mathsf{ACC}}, \mathsf{IDcred}_{\mathtt{PUB}})$$

  for the relation $\mathcal{R}(st, w)$ that outputs $\top$ if:

  1. The signature σ is valid for $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL})$ under $\mathsf{pk}_{\mathsf{IP}}$.
  2. AL satisfies the policy i.e., $\mathsf{P}(\mathsf{AL}) = \top$.
  3. $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$ for some $0 < x \leq \mathsf{Max}_{\mathsf{ACC}}$.
  4. $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathtt{PUB}}; \mathsf{r}')$.
  5. $(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{sk}_{\mathsf{ACC}})$ is a valid key pair according to $\mathcal{R}_{\mathsf{ACC}}$.
  6. $(\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{IDcred}_{\mathtt{PUB}})$ is a valid key pair according to $\mathcal{R}$

- Let $\mathsf{ACI} = (\mathsf{RegID}_{\mathsf{ACC}}, \mathsf{E}_{\mathsf{ID}}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{P}, st, \pi)$ and $\mathsf{SI}_{\mathsf{ACC}} = \mathsf{sk}_{\mathsf{ACC}}$. Send the input $(\mathtt{APPEND}, \mathsf{ACI})$ to $\mathcal{F}_{\mathsf{ledger}}$.
- Store tuple $(\mathsf{ACI}, \mathsf{SI}_{\mathsf{ACC}})$ internally and return $(\mathtt{Created}, \mathsf{RegID}_{\mathsf{ACC}})$.

- On input $(\mathtt{RETRIEVE})$, call $\mathcal{F}_{\mathsf{ledger}}$ on input $\mathtt{RETRIEVE}$. After receiving $(\mathtt{RETRIEVE}, L)$ from $\mathcal{F}_{\mathsf{ledger}}$, output L.

The protocol description for identity providers and anonymity revokers.

- On input $\mathtt{INITIALIZE}$ from $P \in \{\mathsf{IP}_1, \ldots, \mathsf{IP}_{\mathsf{n}}\}$, obtain crs from $\mathcal{F}_{\mathsf{crs}}$, generate key pair $(\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$ and input $(\mathtt{INITIALIZE}, (\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}}))$ to $\mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}_{\mathsf{issue}}$.
- On input $\mathtt{INITIALIZE}$ from $P \in \{\mathsf{AR}_1, \ldots, \mathsf{AR}_{\mathsf{n}}\}$, obtain crs from $\mathcal{F}_{\mathsf{crs}}$, generate key pair $(\mathsf{sk}_{\mathsf{AR}}, \mathsf{pk}_{\mathsf{AR}}) \leftarrow\!\!\$\ \mathsf{TKeyGen}(1^\lambda)$ and input $(\mathtt{REGISTER}, \mathsf{sk}_{\mathsf{AR}}, \mathsf{pk}_{\mathsf{AR}})$ to $\mathcal{F}_{\mathsf{reg}}$.
- On input $(\mathtt{ISSUE}, \mathsf{AH})$ from identity provider IP, call $\mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}_{\mathsf{issue}}$ with input $(\mathtt{ISSUE}, \mathsf{AH}, \mathsf{pk}_{\mathsf{AR}})$. After receiving the response $(\mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}})$ from $\mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}_{\mathsf{issue}}$, set $\mathsf{IPIAH} = (\mathsf{AH}, \mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}})$.
- On input $(\mathtt{RETRIEVE})$, call $\mathcal{F}_{\mathsf{ledger}}$ on input $\mathtt{RETRIEVE}$. After receiving $(\mathtt{RETRIEVE}, L)$ from $\mathcal{F}_{\mathsf{ledger}}$, output L.
- On input $(\mathtt{REVOKE}, \mathsf{RegID}_{\mathsf{ACC}})$ from an identity provider IP and a set of anonymity revokers $\{\mathsf{AR}_i\}_{i \in I \subseteq [\mathsf{n}]}$, proceed as follows:

– Anonymity revokers call $\mathcal{F}_{\mathsf{ledger}}$ on input $\texttt{RETRIEVE}$. After receiving $(\texttt{RETRIEVE}, L)$ from $\mathcal{F}_{\mathsf{ledger}}$, they first look up ACI in L that contains $\mathsf{RegID}_{\mathsf{ACC}}$. Next, each AR_i decrypts the E_{ID} of the ACI by computing $\mu_i = \mathsf{ShareDec}^{\mathsf{n,d}}_{\vec{pk}_I, \mathsf{sk}_i}(\mathsf{E}_{\mathsf{ID}})$. Finally, all anonymity revokers combine their shares and compute $\mathsf{IDcred}_{\mathtt{PUB}} = \mathsf{TCombine}^{\mathsf{n,d}}_{\tilde{\mathsf{pk}}_{\mathsf{I}}}(\mathsf{E}_{\mathsf{ID}}, \{\mu_{\mathsf{i}}\}_{\mathsf{i}\in\mathsf{I}})$ and return $\mathsf{IDcred}_{\mathtt{PUB}}$ to the IP.

– The ACI contains the public name IP of the identity provider who registered $\mathsf{IDcred}_{\mathtt{PUB}}$. If IP is different from the requester, then ignore. Else, the IP locates the $\mathsf{IPIAH} = (\mathsf{AH}, \mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}})$ record, containing the $\mathsf{IDcred}_{\mathtt{PUB}}$ that was decrypted and outputs AH.

- On input $(\texttt{TRACE}, \mathsf{AH})$ from an identity provider IP and a set of anonymity revokers $\{\mathsf{AR}_i\}_{i\in I\subseteq[\mathsf{n}]}$, proceed as follows:

  – IP searches the database to locate the $\mathsf{IPIAH} = (\mathsf{AH}, \mathsf{IDcred}_{\mathtt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}})$ containing the relevant AH and sends $\mathsf{E}_{\mathsf{RegID}}$ via $\mathcal{F}_{\mathsf{smt}}$ to the set of anonymity revokers.

  – Each AR_i computes $\mathsf{K}_i = \mathsf{ShareDec}^{\mathsf{n,d}}_{\vec{pk}_I, \mathsf{sk}_i}(\mathsf{E}_{\mathsf{RegID}})$. Then, they call $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$ on input $(\texttt{COMPUTE}, \mathsf{K}_i)$ and receive all values $\mathsf{PRF}_{\mathsf{K}}(x)$ for $x = 1, \ldots, \mathsf{Max}_{\mathsf{ACC}}$. These values are all the possible values for $\mathsf{RegID}_{\mathsf{ACC}}$ that the AH could have used to form valid accounts.

5.2 Proof of Security for Identity Layer

Tolerated Corruptions. We prove security in two separate cases: with arbitrarily many malicious AHs and up to threshold semi-honest ARs, or with semi honest IP and up to threshold semi-honest ARs. Note that for technical reasons we cannot let the IP be corrupt (even if only semi-honest) at the same time with a malicious AH, since in this case the (monolithic) adversary would learn the secret key of the corrupt IP and would be able to forge invalid credentials for the corrupt AH.
Assumptions on the environment. We consider executions with restricted adversaries and environments, that only input attribute lists AL in the $\texttt{ISSUE}$ command which are valid with respect to the account holder. This restriction captures the fact that an honest IP in the real world is trusted to check (by non-cryptographic means) that an account holder AH actually satisfies the claimed attribute list AL.

Theorem 5.1. *Suppose that* $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{TDec})$ *is a* (n, d)*-threshold encryption scheme, PRF is a weakly robust pseudorandom function,* $\mathcal{R}$ *a hard relation, and* $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerifySig})$ *is a EUF-CMA signature scheme, then* $\Pi_{\mathsf{id\text{-}layer}}$*, for all restricted environment (as defined above), securely implements* $\mathcal{F}_{\mathsf{id\text{-}layer}}$ *in the* $\{\mathcal{F}_{\mathsf{crs}}, \mathcal{F}_{\mathsf{reg}}, \mathcal{F}_{\mathsf{nizk}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}_{\mathsf{issue}}, \mathcal{F}_{\mathsf{ledger}}, \mathcal{F}_{\mathsf{mpc\text{-}prf}}\}$*-hybrid model in the presence of an actively corrupted* AH*, and* d *semi-honest anonymity revokers* $\mathsf{AR}_1, \ldots, \mathsf{AR}_{\mathsf{d}}$*, or semi-honest* IP *and* d *semi-honest anonymity revokers* $\mathsf{AR}_1, \ldots, \mathsf{AR}_{\mathsf{d}}$*.*

The simulator Sim is internally emulating the functionalities $\mathcal{F}_{\mathsf{crs}}, \mathcal{F}_{\mathsf{reg}}, \mathcal{F}_{\mathsf{nizk}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}_{\mathsf{issue}}$, $\mathcal{F}_{\mathsf{ledger}}, \mathcal{F}_{\mathsf{mpc\text{-}prf}}$. At the start Sim initializes empty lists $\mathtt{list}_{\mathsf{issue}}, \mathtt{list}_{\mathsf{acc}}, \mathtt{list}_{\mathsf{ledger}}, \mathtt{list}_{\mathsf{h\text{-}aid}}, \mathtt{list}_{\mathsf{h\text{-}pk}}$, $\mathtt{list}_{\mathsf{h\text{-}sig}}$. Here is the description of the simulator:

- Command $\texttt{INITIALIZE}$: Sim emulates $\mathcal{F}_{\mathsf{crs}}$ and generates public parameters for the signature scheme and threshold encryption scheme. Every time an honest or semi-honest IP or AR invokes the initialize command, the simulator generates a key-pair for them.

- Command $\texttt{ISSUE}$. *Passively corrupted* IP*, honest* AH. The simulator receives $(\texttt{ISSUE}, \mathsf{AH}, \mathsf{IP}, \mathsf{AL})$ in case of corrupt IP from the ideal functionality $\mathcal{F}_{\mathsf{id\text{-}layer}}$, and has to produce a view indistinguishable from the protocol which is consistent with this. The simulator does so by emulating the output $\mathcal{F}_{\mathsf{issue}}$ for IP namely $(\mathsf{IDcred}_{\texttt{PUB}}, \mathsf{AL}, \mathsf{E}_{\mathsf{RegID}})$ to IP where the simulator encrypts a dummy value for $\mathsf{E}_{\mathsf{RegID}}$ and $\mathsf{IDcred}_{\texttt{PUB}}$ is generated according to $\mathcal{R}$. The simulator adds $\mathsf{E}_{\mathsf{RegID}}$ to $\texttt{list}_{\mathsf{h\text{-}ct}}$ and $\mathsf{IDcred}_{\texttt{PUB}}$ to $\texttt{list}_{\mathsf{h\text{-}pk}}$. Sim adds to $\texttt{list}_{\mathsf{issue}}$ the entry $\langle(\mathsf{AH}, \mathsf{AL}, \mathsf{IP}); (\mathsf{E}_{\mathsf{RegID}}, \mathsf{IDcred}_{\texttt{PUB}}, \mathsf{AL})\rangle$.

  Malicious AH*, honest* IP. When a corrupt AH invokes $\mathcal{F}_{\mathsf{issue}}^{\mathcal{R},\mathsf{TE},\mathsf{SIG}}$ on command $\Big(\texttt{ISSUE}, (\mathsf{E}_{\mathsf{RegID}}, \mathsf{K}, r, \mathsf{PK}_{\mathsf{AR}}), \mathsf{AL}, (\mathsf{IDcred}_{\texttt{SEC}}, \mathsf{IDcred}_{\texttt{PUB}}), \mathsf{IP}\Big)$, Sim aborts if $\mathsf{E}_{\mathsf{RegID}}$ is not an encryption of K or if $(\mathsf{IDcred}_{\texttt{SEC}}, \mathsf{IDcred}_{\texttt{PUB}})$ is not a valid key-pair; Sim outputs $\texttt{fail}$ if $\mathsf{E}_{\mathsf{RegID}} \in \texttt{list}_{\mathsf{h-ct}}$ or if $\mathsf{IDcred}_{\texttt{PUB}} \in \texttt{list}_{\mathsf{h\text{-}pk}}$.

  Otherwise, Sim calls command $\texttt{ISSUE}$ of the functionality on input $(\mathsf{IP}, \mathsf{AL})$ and returns $\mathsf{cid} = (\mathsf{IP}, \mathsf{AL}, \mathsf{IDcred}_{\texttt{SEC}}, \sigma, \mathsf{K})$ to AH where σ is a signature computed by Sim using Sign of SIG (since the simulator is internally emulating the honest IP w.r.t. the corrupt account holder AH). Sim adds $(\sigma; (\mathsf{IDcred}_{\texttt{SEC}}, \mathsf{K}, \mathsf{AL}); \mathsf{IP})$ to $\texttt{list}_{\mathsf{h\text{-}sig}}$.

- Command $\texttt{CreateACC}$. *Malicious* AH. When a corrupt AH invokes $\mathcal{F}_{\mathsf{ledger}}$ on input $\mathsf{ACI} = (st, \pi)$ (where $st = (\mathsf{P}, \mathsf{E}_{\mathsf{ID}}, \mathsf{RegID}_{\mathsf{ACC}}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}})$), the simulator Sim uses $\mathcal{F}_{\mathsf{nizk}}$ to extract the witness $w = (\sigma, x, r', \mathsf{IDcred}_{\texttt{SEC}}, \mathsf{K}, \mathsf{AL}, \mathsf{sk}_{\mathsf{ACC}}, \mathsf{IDcred}_{\texttt{PUB}})$ (or abort if the proof doesn't verify). The simulator outputs $\texttt{fail}$ if one of the following condition holds: $(\sigma; (\mathsf{IDcred}_{\texttt{SEC}}, \mathsf{K}, \mathsf{AL}); \mathsf{IP}) \notin \texttt{list}_{\mathsf{h\text{-}sig}}$, if $\mathsf{IDcred}_{\texttt{PUB}} \in \texttt{list}_{\mathsf{h\text{-}pk}}$, if $\mathsf{E}_{\mathsf{ID}} \in \texttt{list}_{\mathsf{h\text{-}ct}}$, if $\mathsf{RegID}_{\mathsf{ACC}} \in \texttt{list}_{\mathsf{h\text{-}aid}}$, or if $x > \mathsf{Max}_{\mathsf{ACC}}$.

  Otherwise the simulator inputs the $\texttt{CreateACC}(\mathsf{cid}, \mathsf{P}, (\mathsf{sk}_{\mathsf{ACC}}, \mathsf{pk}_{\mathsf{ACC}}))$ command to the ideal functionality and, when asked, inputs $\mathsf{aid} = \mathsf{RegID}_{\mathsf{ACC}}$ to the ideal functionality.

  Honest AH. For an honest account holder, the simulator upon receiving $(\texttt{CreateACC}, \mathsf{P}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{IP})$ from the ideal functionality, picks a random aid in the domain of the PRF and forwards it to the functionality (also adds it into $\texttt{list}_{\mathsf{h\text{-}aid}}$). Then, the simulator prepares $st = (\mathsf{P}, \mathsf{E}_{\mathsf{ID}}, \mathsf{RegID}_{\mathsf{ACC}} = \mathsf{aid}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}})$ where E_{ID} is an encryption of a dummy value (and is added to $\texttt{list}_{\mathsf{h\text{-}ct}}$), simulates a proof π via $\mathcal{F}_{\mathsf{nizk}}$ and appends (st, π) to the buffer of the ledger.

  Add entry $(\mathsf{RegID}_{\mathsf{ACC}}, \mathsf{E}_{\mathsf{ID}}, \mathsf{IP}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{P}, \pi)$ in $\texttt{list}_{\mathsf{acc}}$.

- Command $\texttt{RELEASE}$: the simulator simulates these commands directly simulating the calls to $\mathcal{F}_{\mathsf{ledger}}$ e.g., when the adversary invokes $(\texttt{RELEASE}, \Pi)$ adds the permuted buffer to the list $\texttt{list}_{\mathsf{ledger}}$ and then resets the buffer.

- Command $\texttt{RETRIEVE}$ Sim emulates the retrieve command in $\mathcal{F}_{\mathsf{ledger}}$ and gives as output $\texttt{list}_{\mathsf{ledger}}$.

- Command $\texttt{REVOKE}$. *Semi-honest* IP *and up to* d AR*, honest* AH. When the IP and a qualified set of AR (of which up to d are corrupt) invoke $\texttt{REVOKE}$, the simulator Sim obtains $\mathsf{RegID}_{\mathsf{ACC}} =$

aid and AH from the input/output of the functionality $\mathcal{F}_{\mathsf{id\text{-}layer}}$. Now Sim, using $\mathsf{RegID}_{\mathsf{ACC}}$, searches $\mathtt{list}_{\mathsf{acc}}$ and retrieves the corresponding E_{ID}. Similarly, using $(\mathsf{AH}, \mathsf{IP})$, searches $\mathtt{list}_{\mathsf{issue}}$ and retrieves the corresponding $\mathsf{IDcred}_{\mathtt{PUB}}$. Then the simulator Sim equivocates the decryption of E_{ID} to $\mathsf{IDcred}_{\mathtt{PUB}}$ using $\mathsf{SimShare}$ (defined in the simulatability property of the threshold encryption scheme).

Malicious AH *and up to* d AR. The simulator receives $(\mathsf{RegID}_{\mathsf{ACC}}, \mathsf{AH})$ from the ideal functionality, looks up the ciphertext E_{ID} corresponding to $\mathsf{RegID}_{\mathsf{ACC}}$ and runs the threshold decryption protocol as honest parties would do.

- Command `TRACE`. *Semi-honest* IP *and up to* d AR*, honest* AH. When the IP and a qualified set of AR (of which up to d are corrupt) invoke `TRACE`, the simulator Sim receives AH and a list $\mathtt{list}_{\mathsf{RegID}_{\mathsf{ACC}}}$ of aids from the input/output of the functionality $\mathcal{F}_{\mathsf{id\text{-}layer}}$. Now Sim recovers the ciphertext $\mathsf{E}_{\mathsf{RegID}}$. Finally Sim programs the output of $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$ to be consistent with $\mathtt{list}_{\mathsf{RegID}_{\mathsf{ACC}}}$.

  Malicious AH *and up to* d AR. The simulator receives AH and a list of accounts $\{\mathsf{RegID}_{\mathsf{ACC}}\}$ from the ideal functionality. The simulator looks up the ciphertext $\mathsf{E}_{\mathsf{RegID}}$ corresponding to AH and emulates $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$ to output the list $\{\mathsf{RegID}_{\mathsf{ACC}}\}$.

We now argue that the view of the environment in the real world and in the ideal world with the simulator described above are indistinguishable.

Analysis of the Simulated Game. We first argue that the probability of the simulator outputting `fail` is negligible. In the `ISSUE` command, since the emulation of the $\mathcal{F}_{\mathsf{issue}}$ functionality is unconditionally secure, the simulator outputs `fail` only in the following cases: 1. the adversary submits the opening for a ciphertext $\mathsf{E}_{\mathsf{RegID}}$ generated by the simulator (whose randomness is never used anywhere else, and for which the simulator only uses less than d shares of the secret key). An adversary that makes the simulation fail this way can be turned into an attack on the semantic security of the threshold encryption scheme; 2. the adversary submits the secret key for a public key $\mathsf{IDcred}_{\mathtt{PUB}}$ generated by the simulator (whose secret key is never used by the simulator). An adversary that makes the simulation fail this way can be turned into an attack onto the one-wayness of the key generation algorithm (which contradicts the assumption that $\mathcal{R}$ is a hard-relation). In the `CreateACC` command, since the emulation of the $\mathcal{F}_{\mathsf{nizk}}$ functionality is unconditionally secure, and since the adversary cannot make the ledger accept replayed accounts, the simulator outputs `fail` only in the following cases: 3. the adversary outputs a message/signature pair $(\sigma; (\mathsf{IDcred}_{\mathtt{SEC}}, \mathsf{K}, \mathsf{AL}); \mathsf{IP})$ which passes the verification algorithm for the public key of IP and was not generated by the simulator. An adversary that make the simulation fail this way can be turned into an attack on the unforgeability of the signature scheme; 4. the adversary inputs to the $\mathcal{F}_{\mathsf{nizk}}$ functionality the secret key for a public key $\mathsf{IDcred}_{\mathtt{PUB}}$ generated by the simulator (see bullet 2. above); 5. the adversary submits the opening for a ciphertext E_{ID} generated by the simulator (see bullet 1. above); 6. the adversary inputs to the $\mathcal{F}_{\mathsf{nizk}}$ functionality an account id $\mathsf{RegID}_{\mathsf{ACC}}$, together with the key K and input x such that $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}(\mathsf{K}, x)$ and that same $\mathsf{RegID}_{\mathsf{ACC}}$ also had been chosen at random by the simulator when simulating an honest party. An adversary that can make the simulation fail this way can be turned into an attack on the weak robust property of the PRF. 7. the adversary creates an account with $x \geq \mathsf{Max}_{\mathsf{ACC}}$. Since the $\mathcal{F}_{\mathsf{nizk}}$ is emulated with unconditional security, the probability that the simulator outputs `fail` in this way is 0.

We now slowly turn the simulated game into the real protocol via a series of hybrids.

Hybrid 0. This is the simulated protocol with the exception that we never output `fail`. As argued above, the probability that the simulator outputs `fail` is negligible, therefore the distribution generated by the simulated game and this hybrid is computationally close.

Hybrid 1. In the first hybrid we change the way in which the simulator samples the account id aid for for honest AH. Now, instead of picking random values, the simulator picks keys K for each pair of honest account holders and certificate id $(\mathsf{AH}, \mathsf{cid})$ and computes the aid as $\mathsf{PRF}_{\mathsf{K}}(x)$ where x counts how many accounts were opened by that account holder for that cid. An adversary that can distinguish between this hybrid and the simulated protocol can be used to break the pseudorandomness property of PRF.

Hybrid 2. In the second hybrid we change the way in which the simulator generates the ciphertexts E_{ID} for the honest parties during the simulation of `CreateACC`. Now the simulator encrypts the value $\mathsf{IDcred}_{\mathsf{PUB}}$ corresponding to the certificate cid of the account holder AH. An adversary that can distinguish between this hybrid and the previous one can be used to break the semantic security of the threshold encryption scheme.

Hybrid 3. In the third hybrid we change the way in which the simulator generates the ciphertexts $\mathsf{E}_{\mathsf{RegID}}$ for the honest parties during the simulation of `ISSUE`. Now the simulator encrypts the value K corresponding to the certificate cid of the account holder AH, consistently with the value of K which had been sampled for that cid in hybrid 1. An adversary that can distinguish between this hybrid and the previous one can be used to break the semantic security of the threshold encryption scheme.

Hybrid 4. In the fourth hybrid we change the behaviour of the simulator during `REVOKE` for honest AH. Now we let the simulator run the threshold decryption protocol honestly. Note that, since we changed the encryptions $\mathsf{E}_{\mathsf{RegID}}$ in a previous hybrid to encrypt the right value, the output of the decryption in this and the previous hybrid is identical. Hence, an adversary that can distinguish between this and the previous hybrid can be used to break the partial decryption simulatability (PDS) of the threshold encryption scheme.

Hybrid 5. In this hybrid we change the behaviour of the simulator during `TRACE` for honest AH. Now we let the simulator run the code of $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$ honestly instead of programming the output of the functionality. Note that, since we changed the encryptions E_{ID} in a previous hybrid to encrypt the right value, the output of the decryption in this and the previous hybrid is identical. Since the simulation of $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$ is perfect, this hybrid is identically distributed to the previous one.

Hybrid 6. In the final hybrid we change the behaviour of the simulator when simulating the proofs π for honest account holders AH. Up until now the simulator had simply picked random strings π, stored them together with the statement x, and then emulated the answers to `VERIFY` queries of the adversary for such proofs by outputting 1 on behalf of the $\mathcal{F}_{\mathsf{nizk}}$ functionality. Now instead the simulator runs the code of the functionality $\mathcal{F}_{\mathsf{nizk}}$, that is it only outputs that a proof is valid if it can come up with a witness for it. Note that, since we changed all elements in the statement (the encryption, the account id, etc.) in previous hybrids to match the behaviour of a

honest account holder, the simulator knows the necessary witnesses, thus the previously simulated proofs still verify, and this hybrid is identically distributed to the previous one.

Since hybrid 6 is identical to the real protocol (that is, the real protocol run in the hybrid world where all the helping functionalities are available), this concludes the proof.

5.3 Credential Issue Protocol

The issue protocol Π_{issue} uses as its main ingredient a two-round blind signature scheme (as defined in Section 2.3), augmented with a NIZK that proves that the input to the blind-signature protocol is consistent with the ciphertext and the public-key that the account holder sends to the identity provider.

Protocol Issue Π_{issue}

The protocol operates in the $\{\mathcal{F}_{\mathsf{crs}}, \mathcal{F}_{\mathsf{reg}}, \mathcal{F}_{\mathsf{nizk}}, \mathcal{F}_{\mathsf{smt}}\}$-hybrid model. Let $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ be a blind signature scheme and $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{TDec})$ be a (n, d)-threshold encryption scheme, and $\mathcal{R}$ an NP relation corresponding to the account holders' key pair.

- Upon input $(\texttt{SETUP})$, use $\mathcal{F}_{\mathsf{crs}}^{\mathsf{Setup}}$ to generate $\mathsf{p} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ and publicize it to all parties.
- Upon input $(\texttt{INITIALIZE}, (\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}}))$, the identity provider IP checks if the key pair has a correct distribution with respect to the $\mathsf{KeyGen}(\mathsf{p})$. If yes, stores $(\mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}})$ and sends $(\texttt{REGISTER}, \mathsf{sk}_{\mathsf{IP}}, \mathsf{pk}_{\mathsf{IP}})$ to $\mathcal{F}_{\mathsf{reg}}$.
- Upon an input $\Big(\texttt{ISSUE}, (ct, m, r, \mathsf{pk}_{\mathsf{AR}}), \mathsf{aux}, (\mathsf{sk}_{\mathsf{AH}}, \mathsf{pk}_{\mathsf{AH}}), \mathsf{IP}\Big)$ to the account holder AH and an input $(\texttt{ISSUE}, \mathsf{AH}, \mathsf{pk}_{\mathsf{AR}})$ to an identity provider IP, proceed as follows:
  1. AH retrieves $\mathsf{pk}_{\mathsf{IP}}$ from $\mathcal{F}_{\mathsf{reg}}$, computes $\mathtt{sign}_1 = \mathsf{Sign}_1(\mathsf{pk}_{\mathsf{IP}}, (\mathsf{sk}_{\mathsf{AH}}, m, \mathsf{aux}), \mathsf{pp}; r')$ and sends $(\texttt{PROVE}, st, w)$ to $\mathcal{F}_{\mathsf{nizk}}$ for statement $st = (\mathsf{pk}_{\mathsf{AH}}, \mathtt{sign}_1, ct, \mathsf{aux}, \mathsf{pp})$ using secret witness $w = (\mathsf{sk}_{\mathsf{AH}}, m, r', r)$ for the relation $\mathcal{R}_1(st, w)$ that outputs $\top$ if:
     (a) $(\mathsf{sk}_{\mathsf{AH}}, \mathsf{pk}_{\mathsf{AH}}) \in \mathcal{R}$.
     (b) $\mathtt{sign}_1 = \mathsf{Sign}_1(\mathsf{pk}_{\mathsf{IP}}, (\mathsf{sk}_{\mathsf{AH}}, m, \mathsf{aux}), \mathsf{pp}; r')$.
     (c) $ct = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(m; r)$
  2. Upon receiving the proof π, AH sends $(\texttt{SEND}, \mathsf{IP}, (st, \pi))$ to $\mathcal{F}_{\mathsf{smt}}$.
  3. Upon receiving $(\texttt{SENT}, \mathsf{AH}, (st, \pi))$ from $\mathcal{F}_{\mathsf{smt}}$, IP inputs $(\texttt{VERIFY}, st, \pi)$ (for relation $\mathcal{R}_1$) to $\mathcal{F}_{\mathsf{nizk}}$. If they pass, IP computes and sends (through $\mathcal{F}_{\mathsf{smt}}$) $\mathtt{sign}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_{\mathsf{IP}}, \mathtt{sign}_1)$.
     AH runs $\mathsf{Unblind}(\mathtt{sign}_2, r')$ and obtains a signature σ on $(\mathsf{sk}_{\mathsf{AH}}, m, \mathsf{aux})$.

5.4 Proof of Security for Issue Protocol

Theorem 5.2. *Assume that* $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ *is a blind signature scheme. Then,* Π_{issue} *securely implements* $\mathcal{F}_{\mathsf{issue}}$ *in the* $\{\mathcal{F}_{\mathsf{crs}}, \mathcal{F}_{\mathsf{reg}}, \mathcal{F}_{\mathsf{nizk}}, \mathcal{F}_{\mathsf{smt}}\}$*-hybrid model in the presence of an actively corrupted* AH *or a passively corrupted* IP*.*

Proof. We consider corruptions of AH and IP separately.

(Actively) Corrupt AH. The simulator for a corrupt AH receives (st, π) from the $\mathcal{F}_{\mathsf{smt}}$ functionality and extracts the witness w from $\mathcal{F}_{\mathsf{nizk}}$ (or abort if the proof doesn't verify). Now the simulator can input the `ISSUE` command to the ideal functionality which includes the message and randomness used to generate ct, the auxiliary information, and the account public and secret keys. The simulator therefore receives a signature σ from the ideal functionality. Using the simulator guaranteed from the simulatability property (Definition A.3), the simulator computes the appropriate $\mathtt{sign}_2$ message. Note that we can use this simulator since (due to the soundness of the NIZK) we know that $\mathtt{sign}_1$ was computed according to the protocol and our simulator knows the message m and the randomness r' which is used by the adversary to do so.

We claim that this simulation is indistinguishable from the real protocol. To see why, notice that the output of the IP is identical in the real and ideal world (remember that in the $\mathcal{F}_{\mathsf{nizk}}$ hybrid model the proofs are unconditionally sound). Moreover, the view of the corrupted AH consists only of the message $\mathtt{sign}_2$. Therefore, an adversary that can distinguish between the real and simulated protocol can directly be turned into an adversary that breaks the simulatability property (Definition A.3) of the blind signature scheme.

(Passively) Corrupt IP. The case of a corrupted IP is relatively simple since we only consider passively corrupt IP: Here the simulator gets the input and output of the IP from the ideal functionality (which consists of the account public key, the ciphertext, and the auxiliary information), and has to produce a view indistinguishable from the protocol which is consistent with this output. The simulator does so by computing the first message of the blind signature $\mathtt{sign}_1$ by running Sign_1 with some dummy input. Note that all other parts of the statement st are at this point known to the simulator, which therefore only needs to simulate the proof π, but this is trivial to do in the $\mathcal{F}_{\mathsf{nizk}}$-hybrid model by simply appending an arbitrary proof π^* to the view, and then answering verify queries so that π^* is a valid proof for the statement st.

We claim that this simulation is indistinguishable from the real protocol. To see why, notice that the only difference in the view of IP between the real and simulated protocol is in the distribution of $\mathtt{sign}_1$ and therefore an adversary that can distinguish between the real and simulated protocol can directly be turned into an adversary that breaks the blindness property (Definition A.2) of the blind signature scheme.

Finally, note that at first glance it might appear strange that the proof does not mention at all the security of the threshold encryption scheme. However, this is because our ideal functionality just guarantees that the ciphertext is correctly computed according to the encryption algorithm. Thus, our protocol securely implements the functionality regardless of which security (if any) is guaranteed by the encryption scheme!

□

6 Putting Everything Together

We presented all components of the system in a modular way. We now describe how to instantiate each of the components needed in the ID-layer.

UC-NIZK. We use two different types of non-interactive zero knowledge proofs in our implementation. One is based on Σ-protocols made non-interactive with the Fiat-Shamir (FS) transform [FS87], and the other is preprocessing-based zkSNARKs [GGPR13, PHGR13] in the crs model. Unfortunately, known instantiations of both types of NIZKs do not satisfy UC-security.

In order to lift SNARK to be UC-secure we use the transformation of Kosba et al. [KZM+15], which is recalled in Appendix F.2. At a high level, the transformation works having the prover prove an augmented relation $\mathcal{R}_{\mathcal{L}'}$ that is given in Appendix F.2. A pair of one-time signing/verification keys are generated for each proof. The prover is additionally required to show that a ciphertext encrypts the witness of the underlying relation $\mathcal{R}_{\mathcal{L}}$, or the PRF was correctly evaluated on the signature key under a committed key. Then the prover is required to sign the statement together with the proof of $\mathcal{L}'$. Since our goal is to use SNARKs on small circuits for the purposes of prover efficiency, we treat the augmented relation $\mathcal{R}_{\mathcal{L}'}$ as a composite statement [AGM18, CFQ19] and use a combination of SNARKs and sigma protocols to prove the augmented relation of the transformation. We use the CL scheme [CL15] for encryption, and $f_k : x \to H(x)^k$, for $k \in \mathbb{Z}_q$ as the PRF where H maps bit strings to group elements. This PRF can be shown to be secure under the DDH assumption where H is modeled as a random oracle. We can use a sigma protocol to prove correct evaluation of the PRF given public input, public output, and committed key g^k. A standard sigma protocol proof of equality of discrete logarithms can be used to prove equality of CL encrypted and Pedersen committed messages. The composition theorem from [AGM18] can be invoked to argue security of the NIZK for the composite statement formed as the AND of the statements of the lifting transformation.

In Appendix F.1, we show that a simulation-sound NIZK (such as Fiat-Shamir as shown in [FKMV12]) and a perfectly correct CPA-secure encryption scheme are sufficient to instantiate a simulation-extractable NIZK by transforming the relation to include a ciphertext encrypting the witness.[4]

While this lifting technique for transforming a (sound) NIZK to a knowledge-sound NIZK is folklore, the use of CL encryption scheme [CL15] for this goal is novel up to our knowledge. Our choice of the CL encryption scheme in the transformation means that we can at the same time encrypt messages in the same plaintext space as the commitment schemes (thus allowing for efficient proofs of equality of discrete logarithms), and guarantee efficient decryption by the extractor. This is as opposed to using e.g., Paillier (where we could have efficient decryption but would need range proofs to prove equality of exponents in different groups) or ElGamal "in the exponent" (where the group order could be the same but efficient decryption can only be achieved by encrypting the witness in short chunks).

Implementation of Π_{issue}. We instantiate the blind signature scheme BS by the Pointcheval-Sanders (PS) signature scheme [PS16]. We recall the PS scheme in Appendix A.3, and prove that the PS signature satisfies the definitions of Blindness A.2 (here we prove a stronger variant than what given in the original paper) and Simulatability A.3 (which we define, as it is needed for proving UC security of the overall construction). We adopt the threshold encryption TE scheme (Definition A.5) described in [DHMR07] which follows the share and encrypt paradigm. We use the CL encryption scheme [CL15] to encrypt. Once again, our choice of CL encryption scheme means that we can at the same time encrypt messages in the same plaintext space as the commitment schemes (for efficient equality proofs) and guarantee efficient decryption when needed in the `TRACE` command by the ARs.

We now describe the Σ-protocols we use to prove relation $\mathcal{R}_1$ in Π_{issue}. We let $\mathcal{R}$ be the discrete log relation where $\mathsf{pk} = g^{\mathsf{sk}}$. Then, we can prove that public keys and secret keys satisfy $\mathcal{R}$ using the standard Σ-protocol dlog from Appendix B.3. The message output by $\mathtt{Sign}_1$ in the PS blind signature is essentially a Pedersen commitment for vectors (see Appendix A.3). So we prove that

[4] The notions of black-box simulation extractable NIZK and UC-secure NIZK are interchangeable [Gro06, CLOS02].

$\mathtt{Sign}_1$ was executed correctly using the $\mathsf{AggregateDL}$ protocol described in Appendix B.6 (note that due to the homomorphic nature of Pedersen commitment we don't need to prove that the values in the AL which are leaked to the IP are correct, since both parties can add those to the commitment "in public"). Finally, we use the protocol in Appendix C.1 for proving that the ciphertext encrypts the right value, and use standard "AND" composition of Σ-protocols to assert that the values appearing in different proofs are consistent.

Implementation of $\Pi_{\mathsf{id-layer}}$. We instantiate the weakly robust PRF scheme (Definition 2.2) with Dodis-Yampolskiy PRF [DY05]. In this case we use ElGamal as the base encryption scheme for the "share-and-encrypt" ad-hoc threshold encryption scheme TE (Definition 2.4). This is because we are encrypting the public key as a group element, which can also be seen as an encryption of the secret key for "ElGamal in the exponent". Note that this allows to both easily prove knowledge of the secret key, and to make sure that the ARs will only learn the AH public key when decrypting. Due to the algebraic nature of the DY PRF, we can efficiently evaluate it inside an MPC protocol as required to implement $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$ using techniques described in [ST19, DOK+20].

When creating a new account, we use both SNARKs and Sigma protocols for proving a single composite statement consisting of a circuit-part and an algebraic part using the technique of [AGM18] to obtain SNARK on algebraically committed input. This commitment is used to tie the witness of the Sigma protocol to the witness used in the SNARK. We describe briefly how we use a combination of Σ-protocols and SNARKs in order to prove relation $\mathcal{R}$ in $\Pi_{\mathsf{id-layer}}$. We use the protocol in Appendix C.2 for proving knowledge of a signature. We use SNARKs on committed input to prove that account holder's attribute list satisfies a certain policy, and for proving that $x \leq \mathsf{Max}_{\mathsf{ACC}}$ for committed x and public $\mathsf{Max}_{\mathsf{ACC}}$. The protocols for proving that $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$ and $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathsf{PUB}})$ are described in Appendix C.2. Note that all the Σ-protocol proofs are made non-interactive using Fiat-Shamir as discussed in Appendix B.2.

Implementation of a transaction layer. In Appendix E, we include a high-level description of a method for transferring money on the ledger in the accounts-based model.

Acknowledgements. The authors would like to thank all members of the Concordium Blockchain Research Center and the Concordium AG for useful feedback, and in particular: Matthias Hall-Andersen, Jesper Buus Nielsen, Torben Pedersen, Daniel Tschudi.

References

[ACC+20] Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, pages 255–267, 2020.

[AGM18] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, CRYPTO 2018, Part III, volume 10993 of LNCS, pages 643–673. Springer, Heidelberg, August 2018.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, SCN 06, volume 4116 of LNCS, pages 111–125. Springer, Heidelberg, September 2006.

[ASMC12] Man Ho Au, Willy Susilo, Yi Mu, and Sherman SM Chow. Constant-size dynamic k-times anonymous authentication. IEEE Systems Journal, 7(2):249–261, 2012.

[Bag19] Karim Baghery. On the efficiency of privacy-preserving smart contract systems. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, AFRICACRYPT 19, volume 11627 of LNCS, pages 118–136. Springer, Heidelberg, July 2019.

[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.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In 42nd FOCS, pages 136–145. IEEE Computer Society Press, October 2001.

[CCL+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, PKC 2020, Part II, volume 12111 of LNCS, pages 266–296. Springer, Heidelberg, May 2020.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, ACM CCS 2019, pages 2075–2092. ACM Press, November 2019.

[Cha83] David Chaum. Blind signature system. In David Chaum, editor, CRYPTO'83, page 153. Plenum Press, New York, USA, 1983.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, ACM CCS 2006, pages 201–210. ACM Press, October / November 2006.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, EUROCRYPT 2005, volume 3494 of LNCS, pages 302–321. Springer, Heidelberg, May 2005.

[Cho09] Sherman S. M. Chow. Real traceable signatures. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, SAC 2009, volume 5867 of LNCS, pages 92–107. Springer, Heidelberg, August 2009.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, CT-RSA 2015, volume 9048 of LNCS, pages 487–505. Springer, Heidelberg, April 2015.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In 34th ACM STOC, pages 494–503. ACM Press, May 2002.

[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, ASIACRYPT 2018, Part II, volume 11273 of LNCS, pages 733–764. Springer, Heidelberg, December 2018.

[CMS96] Jan Camenisch, Ueli M. Maurer, and Markus Stadler. Digital payment systems with passive anonymity-revoking trustees. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, ESORICS'96, volume 1146 of LNCS, pages 33–43. Springer, Heidelberg, September 1996.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, CRYPTO'97, volume 1294 of LNCS, pages 410–424. Springer, Heidelberg, August 1997.

[CZJ+17] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed E. Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via PVORM. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, ACM CCS 2017, pages 701–717. ACM Press, October / November 2017.

[DDP06] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In Serge Vaudenay, editor, EUROCRYPT 2006, volume 4004 of LNCS, pages 555–572. Springer, Heidelberg, May / June 2006.

[DHMR07] Vanesa Daza, Javier Herranz, Paz Morillo, and Carla Ràfols. CCA2-secure threshold broadcast encryption with shorter ciphertexts. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, ProvSec 2007, volume 4784 of LNCS, pages 35–50. Springer, Heidelberg, November 2007.

[DOK+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, ESORICS 2020, Part II, volume 12309 of LNCS, pages 654–673. Springer, Heidelberg, September 2020.

[DP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In 33rd FOCS, pages 427–436. IEEE Computer Society Press, October 1992.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, PKC 2005, volume 3386 of LNCS, pages 416–431. Springer, Heidelberg, January 2005.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, INDOCRYPT 2012, volume 7668 of LNCS, pages 60–79. Springer, Heidelberg, December 2012.

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, ASIACRYPT 2019, Part I, volume 11921 of LNCS, pages 649–678. Springer, Heidelberg, December 2019.

[FOR17] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. IACR Trans. Symm. Cryptol., 2017(1):449–473, 2017.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, CRYPTO'86, volume 263 of LNCS, pages 186–194. Springer, Heidelberg, August 1987.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, EUROCRYPT 2013, volume 7881 of LNCS, pages 626–645. Springer, Heidelberg, May 2013.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing, 17(2):281–308, April 1988.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. J. ACM, 59(3):11:1–11:35, 2012.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, ASIACRYPT 2006, volume 4284 of LNCS, pages 444–459. Springer, Heidelberg, December 2006.

[KTY04] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, EUROCRYPT 2004, volume 3027 of LNCS, pages 571–589. Springer, Heidelberg, May 2004.

[KZM+15] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. How to use SNARKs in universally composable protocols. Cryptology ePrint Archive, Report 2015/1093, 2015. https://eprint.iacr.org/2015/1093.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, EUROCRYPT 2016, Part II, volume 9666 of LNCS, pages 705–734. Springer, Heidelberg, May 2016.

[Mau09] Ueli M. Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, AFRICACRYPT 09, volume 5580 of LNCS, pages 272–286. Springer, Heidelberg, June 2009.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In Proceedings of the 2013 conference on Internet measurement conference, pages 127–140, 2013.

[NS05] Lan Nguyen and Reihaneh Safavi-Naini. Dynamic k-times anonymous authentication. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, ACNS 05, volume 3531 of LNCS, pages 318–333. Springer, Heidelberg, June 2005.

[NVV18] Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-preserving auditing for distributed ledgers. Cryptology ePrint Archive, Report 2018/241, 2018. https://eprint.iacr.org/2018/241.

[PBF+19] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, FC 2018 Workshops, volume 10958 of LNCS, pages 43–63. Springer, Heidelberg, March 2019.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, CRYPTO'91, volume 576 of LNCS, pages 129–140. Springer, Heidelberg, August 1992.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy, pages 238–252. IEEE Computer Society Press, May 2013.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, CT-RSA 2016, volume 9610 of LNCS, pages 111–126. Springer, Heidelberg, February / March 2016.

[RSY18] Leonid Reyzin, Adam Smith, and Sophia Yakoubov. Turning HATE into LOVE: Homomorphic ad hoc threshold encryption for scalable MPC. Cryptology ePrint Archive, Report 2018/997, 2018. https://eprint.iacr.org/2018/997.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, CRYPTO'89, volume 435 of LNCS, pages 239–252. Springer, Heidelberg, August 1990.

[ST19] Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. In Martin Albrecht, editor, 17th IMA International Conference on Cryptography and Coding, volume 11929 of LNCS, pages 342–366. Springer, Heidelberg, December 2019.

[SU12] Dominique Schröder and Dominique Unruh. Security of blind signatures revisited. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, PKC 2012, volume 7293 of LNCS, pages 662–679. Springer, Heidelberg, May 2012.

[TFS04] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-Times anonymous authentication (extended abstract). In Pil Joong Lee, editor, ASIACRYPT 2004, volume 3329 of LNCS, pages 308–322. Springer, Heidelberg, December 2004.

[TS06] Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, PKC 2006, volume 3958 of LNCS, pages 525–542. Springer, Heidelberg, April 2006.

[Zca] Zcash regulatory and compliance brief. https://z.cash/wp-content/uploads/2020/07/Zcash-Regulatory-Brief-062020.pdf. 2020-06-01.

A Additional Definitions

A.1 Bilinear groups

Throughout the paper we let $\mathcal{G}$ be a bilinear group generator that on input security parameter λ returns $(q, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow\!\!\$\ \mathcal{G}(1^\lambda)$ with the following properties:

- $\mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T$ are groups of prime order q.
- $e : \mathbb{G} \times \widetilde{\mathbb{G}} \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear map such that $\forall U \in \mathbb{G}, \forall V \in \widetilde{\mathbb{G}}, \forall a, b \in \mathbb{Z} : e(U^a, V^b) = e(U, V)^{ab}$.
- g and $\tilde{g}$ are generators for $\mathbb{G}$ and $\widetilde{\mathbb{G}}$ respectively.

A.2 Commitment Schemes

Definition A.1 (Commitment Schemes)**.** *A commitment scheme* $\mathsf{COM} = (\mathit{Setup}, \mathsf{Commit}, \mathsf{OpenVrf})$ *consists of the following algorithms:*

- $\mathit{Setup}(1^\lambda)$*: given the security parameter* 1^λ*, it outputs a commitment key* ck*.*
- $\mathsf{Commit}_{\mathsf{ck}}(m)$*: a probabilistic algorithm that given a message* m*, outputs a pair* $(cm, r_{cm}) \leftarrow\!\!\$\ \mathsf{Commit}_{\mathsf{ck}}(m)$ *of commitment* cm *and an opening* r_{cm}*. We sometimes write* $\mathsf{Commit}_{\mathsf{ck}}(m; r_{cm})$ *when we want to be able to fix the value of the randomness* r_{cm} *to a specific value.*
- $\mathsf{OpenVrf}_{\mathsf{ck}}(cm, m, r_{cm})$*: a deterministic algorithm that outputs a bit indicating acceptance or rejection.*

Hiding property. We say that COM is hiding if for all non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that

$$\left| \frac{1}{2} - \Pr\left[b' = b \;\middle|\; \begin{array}{c} \mathsf{ck} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ck}); \\ \\ b \leftarrow\!\!\$\ \{0,1\}; \quad (cm, r_{cm}) \leftarrow\!\!\$\ \mathsf{Commit}_{\mathsf{ck}}(m_b); \quad b' \leftarrow\!\!\$\ \mathcal{A}(cm); \end{array} \right] \right| \leq negl(\lambda)$$

where the probability is over the randomness of $\mathcal{A}$, Setup, Commit and the choice of bit b. We say that COM is perfectly hiding if $negl(\lambda) = 0$.

Binding property. A commitment scheme COM is binding if for all non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that

$$\Pr\left[\begin{array}{c} \mathsf{OpenVrf}_{\mathsf{ck}}(cm, m_0, r_0) = 1 \wedge \\ \mathsf{OpenVrf}_{\mathsf{ck}}(cm, m_1, r_1) = 1 \wedge m_0 \neq m_1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{ck} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda); \\ \\ (cm, m_0, m_1, r_0, r_1) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{ck}); \end{array} \right] \leq negl(\lambda)$$

where the probability is over the randomness of $\mathcal{A}$ and Setup. We say that COM is perfectly binding if $negl(\lambda) = 0$.

$\mathsf{Exp}^{\mathsf{Blind}}_{\mathcal{A}}(\lambda)$

$\mathsf{p} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda); (\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(\mathsf{p});$

$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{sk}, \mathsf{p});$

$b \leftarrow\!\!\$\ \{0, 1\};$

$(\mathtt{sign}_1, \omega) \leftarrow\!\!\$\ \mathsf{Sign}_1(\mathsf{pk}, m_b);$

$(\overline{\mathtt{sign}}_1, \overline{\omega}) \leftarrow\!\!\$\ \mathsf{Sign}_1(\mathsf{pk}, m_{1-b});$

$\mathtt{sign}_2, \overline{\mathtt{sign}}_2 \leftarrow \mathcal{A}(\mathtt{sign}_1, \overline{\mathtt{sign}}_1);$

let σ_b, σ_{1-b} denote the (possibly undefined) local outputs of

$\mathsf{Unblind}(\mathtt{sign}_2, \omega)$ and $\mathsf{Unblind}(\overline{\mathtt{sign}}_2, \overline{\omega})$ respectively.

if $\mathsf{VerifySig}(\mathsf{pk}, m_0, \sigma_0) = 0$ or $\mathsf{VerifySig}(\mathsf{pk}, m_1, \sigma_1) = 0$, **then** $(\sigma_0, \sigma_1) = (\perp, \perp)$;

$b^* \leftarrow \mathcal{A}(\sigma_0, \sigma_1);$

if $b = b^* \wedge |m_0| = |m_1|$, **then return** 1; **else return** 0;

Figure 1: Experiment in the definition of blindness property

A.3 (Blind) Signature Scheme

We define the properties we require from blind signature schemes in this work – Blindness and Simulatability. Blindness captures the requirement that we run two blind signing protocols on two messages of the adversary's choice, and the adversary should not be able to say which input-signature pair corresponds to which execution. This is stronger than the definition in [PS16]; however, we can show that the PS blind signature scheme can be made to satisfy this stronger, standard definition. This follows from the rerandomizability of PS signatures.

Definition A.2 (Blindness)**.** *An interactive signature scheme* $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ *is called blind if for any PPT adversary* $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\mathsf{Blind}}_{\mathcal{A}}(\lambda) = 1] \approx_\lambda 1/2$, *where the experiment* $\mathsf{Exp}^{\mathsf{Blind}}_{\mathcal{A}}(\lambda)$ *is defined in Fig. 1.*

Definition A.3 (Simulatability)**.** *An interactive signature scheme* $\mathsf{BS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Unblind}, \mathsf{VerifySig})$ *is called simulatable if there exist a PPT algorithm* Sim *s.t. for any PPT adversary* $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\mathsf{Sim}}_{\mathcal{A}}(\lambda) = 1] \approx_\lambda 1/2$, *where the experiment* $\mathsf{Exp}^{\mathsf{Sim}}_{\mathcal{A}}(\lambda)$ *is defined in Fig. 2.*

A.3.1 Instantiation with PS Signature scheme [PS16]

We specify here the multi-message Pointcheval-Sanders (PS) signature scheme and how it realizes our definition of Blind signature. The PS scheme is a randomizable signature that uses groups with asymmetric pairing and allows to sign messages that are completely unknown to the signer.

- $\mathsf{Setup}(1^\lambda)$: Sample $\mathsf{p} := (q, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow\!\!\$\ \mathcal{G}(1^\lambda)$.
- $\mathsf{KeyGen}(\mathsf{p})$: Given $(q, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$, the key generation works as follows:
  1. Choose $x, y_i \leftarrow\!\!\$\ \mathbb{F}_q$ for $i = 1, \ldots, \ell$[5].
  2. Set $X = g^x, \tilde{X} = \tilde{g}^x, Y_i = g^{y_i}$ and $\tilde{Y}_i = \tilde{g}^{y_i}$ for $i = 1, \ldots, \ell$.

[5]Note that the scheme allows to sign vectors over $\mathbb{F}_q$ of length ℓ, so ℓ should be chosen with this in mind.

$\mathsf{Exp}^{\mathsf{Sim}}_{\mathcal{A}}(\lambda)$

$\mathsf{p} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda); (\mathsf{sk}, \mathsf{pk}) \leftarrow\!\$\ \mathsf{KeyGen}(\mathsf{p});$

$b \leftarrow\!\$\ \{0,1\};$

$(m; r) \leftarrow \mathcal{A}(\mathsf{sk}, \mathsf{p});$

$(\mathtt{sign}_1, \omega) \leftarrow\!\$\ \mathsf{Sign}_1(\mathsf{pk}, m; r);$

$\mathtt{sign}_2^0 \leftarrow\!\$\ \mathsf{Sign}_2(\mathsf{sk}, \mathtt{sign}_1);$

$\sigma = \mathsf{Unblind}(\mathtt{sign}_2^0, \omega);$

$\mathtt{sign}_2^1 \leftarrow\!\$\ \mathsf{Sim}(\mathsf{pk}, m, \omega, \sigma);$

$b^* \leftarrow \mathcal{A}(\sigma, m, \mathtt{sign}_2^b);$

if $b = b^*$, **then return** 1; **else return** 0;

Figure 2: Experiment in the definition of simulatability property

3. The public key is now $(\tilde{X}, \{Y_i\}_{i\in[\ell]}, \{\tilde{Y}_i\}_{i\in[\ell]})$, while the secret key is X.

- $\mathsf{Sign}_1(\mathsf{pk}, m = (m_1, \ldots, m_\ell))$: The user chooses $\omega \leftarrow\!\$\ \mathbb{F}_q$, and computes $\mathtt{sign}_1 = g^\omega \prod_{i=1}^{\ell} Y_i^{m_i}$. Note that ω is random and not used for anything else, and therefore $\mathtt{sign}_1$ perfectly hides the rest of the m_i.
- $\mathsf{Sign}_2(\mathsf{sk}, \mathtt{sign}_1)$: The signer chooses $\alpha \leftarrow\!\$\ \mathbb{F}_q$ and sets $a' = g^\alpha \neq 1_{\mathbb{G}}$ and $b' = (X \cdot \mathtt{sign}_1)^\alpha$. They output $\mathtt{sign}_2 = (a', b')$.
- $\mathsf{Unblind}(\mathtt{sign}_2, \omega)$: For $\mathtt{sign}_2 = (a', b')$, the user first computes $\hat{\sigma} = (a', b'/a'^\omega)$. Then, they rerandomize $\hat{\sigma}$ by choosing $\gamma \leftarrow\!\$\ \mathbb{F}_q$ and computing $\sigma = \hat{\sigma}^\gamma = (\hat{\sigma}_1^\gamma, \hat{\sigma}_2^\gamma)$. The user returns σ.
- $\mathsf{VerifySig}(\mathsf{pk}, m = (m_1, \ldots, m_\ell), \sigma)$: The algorithm parses σ as (σ_1, σ_2) and outputs 1 if the following checks pass:
  1. $\sigma_1 \neq 1_{\mathbb{G}}$.
  2. $e(\sigma_1, \tilde{X} \prod \tilde{Y}_j^{m_j}) = e(\sigma_2, \tilde{g})$.

PS satisfies Correctness, Unforgeability, Blindness and Simulatability. **Correctness.** If $\sigma = (\sigma_1 = h, \sigma_2 = h^{x+\sum y_j m_j})$, then

$$\begin{aligned} e(\sigma_1, \tilde{X} \prod \tilde{Y}_j^{m_j}) &= e(h, \tilde{X} \prod \tilde{Y}_j^{m_j}) = e(h, \tilde{g})^{x+\sum y_j m_j} \\ &= e(h^{x+\sum y_j m_j}, \tilde{g}) = e(\sigma_2, \tilde{g}). \end{aligned}$$

Unforgeability. As shown in [PS16], the EUF-CMA security of this signature is equivalent to the single-message version, where its security relies on the following assumption[6].

Definition A.4 (PS Assumption 1 [PS16])**.** *Let $(q, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$ be a bilinear group setting of type 3. For $(X = g^x, Y = g^y)$ and $(\tilde{X} = \tilde{g}^x, \tilde{Y} = \tilde{g}^y)$, where x and y are random scalars in $\mathbb{Z}_q$, we define the oracle $\mathcal{O}(m)$ on input $m \in \mathbb{Z}_q$ that chooses a random $h \in \mathbb{G}$ and outputs the*

[6]The assumption is related to the LRSW assumption and is proved in [PS16] to hold in the bilinear generic group model.

$\mathsf{Exp}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d})$

$[n] \supseteq \mathcal{C} \leftarrow \mathcal{A}(\lambda, \mathsf{n}, \mathsf{d});$

$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow\!\$\ \mathsf{TKeyGen}(1^\lambda)$ **for** $i \in [n]$;

$([n] \supseteq R, m_0, m_1) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$;

$b \leftarrow\!\$\ \{0,1\}; ct \leftarrow\!\$\ \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\vec{\mathsf{pk}}_R}(m_b)$;

$b' \leftarrow \mathcal{A}(ct)$;

if $\Big(b' = b \wedge |m_0| = |m_1| \wedge |R \cap \mathcal{C}| \leq \mathsf{d}\Big)$;

then return 1; **else return** 0;

Figure 3: Static Semantic Security Experiment for Threshold Encryption

pair $P = (h, h^{x+my})$. *Given* $(g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ *and unlimited access to this oracle, no adversary can efficiently generate such a pair, with* $h \neq 1_{\mathbb{G}}$, *for a new scalar* m^*, *not asked to* $\mathcal{O}$.

Blindness. Let us define a a game H_0 which is define as game $\mathsf{Exp}^{\mathsf{Blind}}_{\mathcal{A}}$ with the only difference that $b = 0$. We can also define a symmetrical experiment H_1 where the bit $b = 1$.

To claim indistinguishability between this two games we make the following observations: 1) $\mathtt{sign}_1$ perfectly hides m; 2) the message $\mathtt{sign}_2$ coming from a corrupted signer is rerandomized (thought $\mathsf{Unblind}$ algorithm) before the final signature σ is given to adversary $\mathcal{A}$. From the above two observations follow that the execution in which $\mathcal{A}$ participates first in the computation of σ_0 and then of σ_1 is identically distributed to the one where $\mathcal{A}$ participate first in the computation of σ_1 and then of σ_0. Therefore H_0 is indistinguishable from H_1.

Simulatability. We start by defining the simulator Sim. Sim given m, ω and the signature $\sigma = (\sigma_1, \sigma_2)$ chooses $\beta \leftarrow\!\$\ \mathbb{F}_q$ and returns $\mathtt{sign}_2 = (a', b')$ where $a' = \sigma_1^\beta$ and $b' = \sigma_2^\beta \cdot \sigma_1^{\omega\cdot\beta}$.

We now argue that the game H_0 where $\mathtt{sign}_2$ is computed honestly (i.e., as output of Sign_2) is identical distribute to a simulated game H_1 where $\mathtt{sign}_2$ is computed by Sim. This indistinguishability follows from the fact that distribution of the message $\mathtt{sign}_2$ output by Sim and the distribution of the output of Sign_2 are identical. We conclude the proof observing that H_0 corresponds to the experiment $\mathsf{Exp}^{\mathsf{Sim}}_{\mathcal{A}}$ where the bit $b = 0$ and H_1 corresponds to $\mathsf{Exp}^{\mathsf{Sim}}_{\mathcal{A}}$ with $b = 1$. □

A.4 Threshold Encryption Scheme

Definition A.5. *A* (n, d)*-threshold encryption scheme* $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{ShareDec}, \mathsf{TCombine})$ *is* (n, d)*-statically semantic secure (SSS) if for all PPT adversaries* $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d}) = 1] \approx_\lambda 1/2$, *where the experiment* $\mathsf{Exp}^{\mathsf{SSS}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d})$ *is defined in Fig. 3.*

Definition A.6. *We say that a* (n, d)*-threshold encryption scheme* $\mathsf{TE} = (\mathsf{TKeyGen}, \mathsf{TEnc}, \mathsf{ShareDec}, \mathsf{TCombine})$ *is* (n, d)*-partial decryption simulatable (PDS) if there exists an efficient algorithm* $\mathsf{SimPart}$ *such that for all PPT adversaries* $\mathcal{A}$, $\Pr[\mathsf{Exp}^{\mathsf{PDS}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d}) = 1] \approx_\lambda 1/2$, *where the experiment* $\mathsf{Exp}^{\mathsf{PDS}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d})$ *is described in Fig. 4.*

A.4.1 Construction based on Share and Encrypt paradigm.

Below, we recall the construction of [DHMR07]. Let $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme and $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconstruct})$ be a secret sharing scheme.

$\mathsf{Exp}^{\mathsf{PDS}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d})$

$[n] \supseteq \mathcal{C} \leftarrow \mathcal{A}(\lambda, \mathsf{n}, \mathsf{d})$;

$(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow\!\$\ \mathsf{TKeyGen}(1^\lambda)$ **for** $i \in [n]$;

$([n] \supseteq R, m_0, m_1) \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$;

$b \leftarrow\!\$\ \{0,1\}; ct_0 \leftarrow\!\$\ \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\vec{\mathsf{pk}}_R}(m_0); ct_1 \leftarrow\!\$\ \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\vec{\mathsf{pk}}_R}(m_1)$;

if $b = 0$ **then** $\mu_i = \mathsf{ShareDec}^{\mathsf{n},\mathsf{d}}_{\vec{pk}_R,\mathsf{sk}_i}(ct_0)$ **for** $i \in R\backslash\mathcal{C}$;

else if $b = 1$ **then** $\Big(\mu_i = \mathsf{ShareDec}^{\mathsf{n},\mathsf{d}}_{\vec{pk}_R,\mathsf{sk}_i}(ct_1)$ **for** $i \in R \cap \mathcal{C}$

$\wedge\ \mu_i = \mathsf{SimPart}(\mathsf{n}, \mathsf{d}, \vec{pk}_R, c_0, \{\mu_j\}_{j\in R\cap\mathcal{C}}, m_0)$ **for** $i \in R\backslash\mathcal{C}\Big)$;

$b' \leftarrow \mathcal{A}(ct_b, \{\mu_j\}_{j\in R\backslash\mathcal{C}})$;

if $\Big(b' = b \wedge |m_0| = |m_1| \wedge |R \cap \mathcal{C}| \leq \mathsf{d}\Big)$;

then return 1; **else return** 0;

Figure 4: Partial Decryption Simulatability Experiment for Threshold Encryption

- $\mathsf{TKeyGen}(1^\lambda)$: return $\mathsf{KeyGen}(1^\lambda)$.
- $\mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\vec{\mathsf{pk}}_R}(m)$: return $ct = (ct_1, \ldots, ct_\mathsf{n})$, where $ct_i = \mathsf{Enc}_{\mathsf{pk}_i}(m_i)$ for $i \in R$ and $(m_1, \ldots, m_\mathsf{n}) = \mathsf{Share}^{\mathsf{n},\mathsf{d}}(m)$.
- $\mathsf{ShareDec}^{\mathsf{n},\mathsf{d}}_{\vec{pk}_R,\mathsf{sk}_i}(ct)$: let $ct = (ct_1, \ldots, ct_\mathsf{n})$. Return decryption share $\mu_i = \mathsf{Dec}_{\mathsf{sk}_i}(ct_i)$.
- $\mathsf{TCombine}^{\mathsf{n},\mathsf{d}}_{\vec{pk}_R}(ct, \{\mu_i\}_{i\in I})$: Return $m = \mathsf{Reconstruct}^{\mathsf{n},\mathsf{d}}(\{\mu_i\}_{i\in I})$.

The following theorem is proved in [RSY18] (see Appendix D, Thm. 11 of [RSY18]).

Lemma A.7. *The threshold encryption scheme* TE *described above is* (n, d)*-statically secure, as long as* SS *is a secure share simulatable* (n, d)*-secret sharing scheme, and* PKE *is a CPA-secure public key encryption scheme.*

A.5 Secret Sharing Scheme

Informally, a (n, d)-secret sharing of a secret value s is an encoding of s into n shares, such that any $\mathsf{d}+1$ shares together can reconstruct s, whereas fewer shares reveal no information about s.

Definition A.8. *A* (n, d)*-secret sharing scheme* $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconstruct})$ *over message space* S *consists of the following algorithms:*

- $\mathsf{Share}^{\mathsf{n},\mathsf{d}}(s; r) \rightarrow ([s]_1, \ldots, [s]_\mathsf{n})$ *is a randomized algorithm that on any input* $s \in S$ *and randomness* r*, outputs* n *shares* $([s]_1, \ldots, [s]_\mathsf{n})$.
- $\mathsf{Reconstruct}^{\mathsf{n},\mathsf{d}}([s]_{i_1}, \ldots, [s]_{i_{\mathsf{d}+1}}) \rightarrow s'$ *is a deterministic algorithm that takes* $\mathsf{d}+1$ *shares as input and returns the reconstructed message* $s' \in S$.

$\mathsf{Exp}^{\mathsf{SS-Sim}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d})$

$(\mathcal{C}, s_0, s_1) \leftarrow \mathcal{A}(\mathsf{n}, \mathsf{d})$;

if $s_0 \notin S \vee s_1 \notin S \vee |s_0| \neq |s_1|$ **then return** 0;

$([s_0]_1, \ldots, [s_0]_{\mathsf{n}}) \leftarrow \mathsf{Share}^{\mathsf{n},\mathsf{d}}(s_0)$;

$([s_1]_1, \ldots, [s_1]_{\mathsf{n}}) \leftarrow \mathsf{Share}^{\mathsf{n},\mathsf{d}}(s_1)$;

$\{[s'_0]_i\}_{i\in[\mathsf{n}]\setminus\mathcal{C}} \leftarrow \mathsf{SimShare}(\mathsf{n}, \mathsf{d}, \{[s_1]_i\}_{i\in\mathcal{C}}, s_0)$;

$b \leftarrow \{0,1\}$; **if** $b = 0$ **then** $x = \{[s_0]_i\}_{i\in[\mathsf{n}]}$;

else $x = \{[s_1]_i\}_{i\in\mathcal{C}} \cup \{[s'_0]_i\}_{i\in[\mathsf{n}]\setminus\mathcal{C}}$;

$b' \leftarrow \mathcal{A}(x)$; **if** $b' = b \wedge |\mathcal{C}| \leq \mathsf{d}$ **then return** 1;

else return 0;

Figure 5: Share Simulatability experiment for Secret Sharing

Informally, correctness requires that $s' = s$, and privacy requires that given d or fewer shares of either s_0 or s_1, no efficient adversary can guess which message was shared.

Share Simulatability. We additionally use another property for a secret sharing as defined in [RSY18] which requires that given any set of d or fewer shares of s_0 and given s_1, there exists a PPT algorithm $\mathsf{SimShare}$ that generates the rest of the shares in such a way that is indistinguishable from a fresh sharing of s_1.

Definition A.9 (Share Simulatability)**.** *We say that a* (n, d)*-secret sharing scheme* $\mathsf{SS} = (\mathsf{Share}, \mathsf{Reconstruct})$ *over message space* S *is share simulatable if there exists an efficient simulation algorithm* $\mathsf{SimShare}$ *such that for all PPT adversaries* $\mathcal{A}$*,* $\Pr[\mathsf{Exp}^{\mathsf{SS-Sim}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d}) = 1] \approx_\lambda 1/2$*, where the experiment* $\mathsf{Exp}^{\mathsf{SS-Sim}}_{\mathcal{A}}(\lambda, \mathsf{n}, \mathsf{d})$ *is described in Fig. 5.*

A.6 Non-Interactive Zero-Knowledge Proofs

Definition A.10 (NIZK)**.** *A non-interactive zero-knowledge proof system (NIZK) for an NP language* $\mathcal{L}$ *with relation* $\mathcal{R}_{\mathcal{L}}$ *consists of the following four algorithms:*

- $\mathsf{CRSGen}(1^\lambda, \mathcal{L})$*. On input* 1^λ *and the description of the language* $\mathcal{L}$*, generates a common reference string* crs*, a trapdoor* τ *and an extraction key* ek*.*
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w})$*. On input of a* crs*, a statement* x *with witness* w*, outputs a proof* π*.*
- $\mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi)$*. Given a* crs*, a statement* x *and a proof* π*, outputs a bit indicating accept or reject.*
- $\mathsf{SimProve}(\mathsf{crs}, \tau, \mathsf{x})$*. On input a* crs*, a trapdoor* τ *and a statement* x*, outputs a simulated proof* π *without needing a witness for* x*.*

Completeness. A NIZK is (perfectly) complete, if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathcal{L}}$, we have:

$$\Pr\left[\begin{array}{c}(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathsf{CRSGen}(1^\lambda, \mathcal{L}), \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w}) : \\ \mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi) = 1\end{array}\right] = 1$$

Soundness. A NIZK scheme for the language $\mathcal{L}$ is called (computationally) sound, if for all PPT adversaries $\mathcal{A}$, we have:

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\tau,\mathsf{ek})\leftarrow\mathsf{CRSGen}(1^\lambda,\mathcal{L}),(\mathsf{x},\pi)\leftarrow\mathcal{A}(\mathsf{crs}):\\ \mathsf{Verify}(\mathsf{crs},\mathsf{x},\pi)=1\wedge\mathsf{x}\notin\mathcal{L}\end{array}\right]\approx_\lambda 0$$

Zero-knowledge. A NIZK scheme for the language $\mathcal{L}$ is called (computationally) zero-knowledge if for all PPT adversary $\mathcal{A}$,

$$\begin{aligned}&\Pr\left[\ (\mathsf{crs},\tau,\mathsf{ek})\leftarrow\mathsf{CRSGen}(1^\lambda,\mathcal{L}):\mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})=1\ \right]\\ \approx_\lambda&\Pr\left[\ (\mathsf{crs},\tau,\mathsf{ek})\leftarrow\mathsf{CRSGen}(1^\lambda,\mathcal{L}):\mathcal{A}^{\mathsf{SimProve}(\mathsf{crs},\tau,\cdot)}(\mathsf{crs})=1\ \right]\end{aligned}$$

Simulation Soundness. A NIZK proof for the language $\mathcal{L}$ is called simulation sound if for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\tau,\mathsf{ek})\leftarrow\mathsf{CRSGen}(1^\lambda,\mathcal{L});(\mathsf{x},\pi)\leftarrow\mathcal{A}^{\mathsf{SimProve}(\mathsf{crs},\tau,\cdot)}(\mathsf{crs}):\\ (\mathsf{x},\pi)\notin\mathcal{Q}\wedge\mathsf{x}\notin\mathcal{L}\wedge\mathsf{Verify}(\mathsf{crs},\mathsf{x},\pi)=1\end{array}\right]\approx_\lambda 0$$

where $\mathcal{Q}$ is the list of simulation queries and responses (x_i,π_i).

Simulation extractability. This is a strong notion which requires that for any adversary that outputs a proof after seeing many simulated proofs (for either true or false statements), there exists an efficient extractor that can extract the witness from the proof. More formally, we say that a NIZK scheme for the language $\mathcal{L}$ is simulation extractable if there exists an efficient algorithm Ext (called extractor), such that for any PPT adversary $\mathcal{A}$, we have:

$$\Pr\left[\begin{array}{c}(\mathsf{crs},\tau,\mathsf{ek})\leftarrow\mathsf{CRSGen}(1^\lambda,\mathcal{L});(\mathsf{x},\pi)\leftarrow\mathcal{A}^{\mathsf{SimProve}(\mathsf{crs},\tau,\cdot)}(\mathsf{crs},\mathsf{ek});\\ \mathsf{w}\leftarrow\mathsf{Ext}(\mathsf{crs},\mathsf{ek},\mathsf{x},\pi):(\mathsf{x},\pi)\notin\mathcal{Q}\wedge(\mathsf{x},\mathsf{w})\notin\mathcal{R}_\mathcal{L}\wedge\mathsf{Verify}(\mathsf{crs},\mathsf{x},\pi)=1\end{array}\right]\approx_\lambda 0$$

where $\mathcal{Q}$ contains the list of statement-proof pairs that $\mathcal{A}$ asks the oracle $\mathsf{SimProve}(\mathsf{crs},\tau,\cdot)$. Note that this definition considers a strong version of simulation extractability, where the adversary should not even be able to generate a new proof (different from oracle's response) for a statement that has been queried before. We can relax this definition by considering a weaker version where the adversary may be able to generate a new proof for a queried statements. However, for the universally composable NIZKs that we use for realizing $\mathcal{F}_{\mathsf{nizk}}$ (defined in section 4), we require the stronger definition. Thus, when we discuss about the notion of simulation extractability in section F, we mean the stronger version defined above.

B Implementation Details

This section contains a detailed description of the protocols to implement for the ID layer.

B.1 Auxiliary Σ-protocols.

Let $(q, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$ be a bilinear group as defined in A.1. In the Σ protocols described in the following we assume that all group elements and commitment keys and public keys for signature and encryption schemes come from a group of order q, where in some cases it does not matter if we use $\mathbb{G}$, $\widetilde{\mathbb{G}}$ or $\mathbb{G}_T$, but there are also cases where a particular group has to be used. The protocol applying a Σ-protocol will specify which group is used. We denote by $\mathsf{PK}\{(x, y, \cdots) : \text{ statements about } x, y\}$, a zero knowledge proof of knowledge of $x, y, \cdots$ that satisfies statements. Here, $x, y, \cdots$ are private (witness), and other values in statements are public. The sigma protocols we describe below are standard protocols that have appeared in other works [Sch90, CS97] and follow from the *pre-image protocol* of Maurer [Mau09] for proving knowledge of a pre-image of a group homomorphism that unifies and generalizes a large class of protocols in literature.

B.2 Using Fiat-Shamir for non-interactive Proof

All Σ protocols below are described as interactive 2-party protocols. When using them for non-interactive proofs, we use the Fiat-Shamir paradigm with hash function to replace a random oracle. More precisely, this will be done as follows: Suppose the messages in the protocol are (a, c, z), where c is the random challenge from the verifier, and where x is the public input. Then the prover executes the protocol on his own as follows: they compute a first, and then $\mathsf{H}_{\mathtt{RO}}(x, a)$. They set c to be the first $\lfloor \log_2 q \rfloor$ bits of the hash value. Finally they compute the last message z. The non-interactive proof is (c, z).

To verify a proof we use the fact that all our Σ protocols have a verification equation that allows you to compute a from x, c and z. So given x and e, z, the verifier computes a, $\mathsf{H}_{\mathtt{RO}}(x, a)$ and compares c to the first $\lfloor \log_2 q \rfloor$ bits of the hash value.

If a proof consists of several, say T Σ protocols, we do them all in parallel as follows: the prover computes all T first messages and concatenates them to get M_1. Then they compute $\mathsf{H}_{\mathtt{RO}}(X, M_1)$ where X is the concatenation of all public inputs to the protocols. They set c to be the first $\lfloor \log_2 q \rfloor$ bits of the hash value. Finally they compute the T last messages and concatenates them all to get M_2. The non-interactive proof is (c, M_2).

To verify X and (c, M_2), where X contains all the public inputs, the verifier splits M_2 into T individual last messages. As above, they use the verification equations to compute the T first messages. They concatenate them to get M_1, computes $\mathsf{H}_{\mathtt{RO}}(X, M_1)$ and sets c to the first $\lfloor \log_2 q \rfloor$ bits of the hash value.

B.3 Sigma protocol for proving knowledge of discrete log

Protocol dlog : $\mathsf{PK}\{(x) : y = g^x\}$

- The prover computes and sends $a = g^\alpha$ for randomly chosen $\alpha \in \mathbb{Z}_q^*$.
- The verifier sends a random challenge c at random from $\mathbb{Z}_q$.
- The prover sends $z = \alpha + cx \mod q$
- The verifier checks if $a = g^{-z} y^c$. If yes, the verifier accepts.

We describe here for completeness the non-interactive variant using Fiat-Shamir but remember that this approach will be used in parallel for several protocols for applications that require more than one Sigma protocol.

Protocol NI dlog : $\mathsf{PK}\{(x) : y = g^x\}$

- The prover computes $a = g^\alpha$ for randomly chosen $\alpha \in \mathbb{Z}_q^*, c = H(y||a), z = \alpha + cx$ and sends (c, z) to the verifier.
- The verifier computes if $a = y^{-c}g^z$ and checks that $c = H(y||a)$. If yes, the verifier accepts.

B.4 Sigma protocol for proving equality of committed value and Elgamal encrypted value

The prover has produced an ElGamal encryption $(e_1, e_2) = (g_1^R, g_1^x h_1^R)$ under public key g_1, h_1 that is an encryption of g_1^x with randomness R. They have also committed to x: $C = g^x h^r$ and wants to show that the same x appears in the encryption and in the commitment.

Protocol com−enc−eq : $\mathsf{PK}\{(x, r, R) : e_1 = g_1^R \wedge e_2 = g_1^x h_1^R \wedge C = g^x h^r\}$

1. The prover computes $a_1 = g_1^\alpha, a_2 = g_1^\beta h_1^\alpha, a_3 = g^\beta h^\gamma$ for randomly chosen $\alpha, \beta, \gamma \in \mathbb{Z}_q^*$ and sends (a_1, a_2, a_3) to the verifier.
2. The verifier sends a random challenge c at random from $\mathbb{Z}_q$.
3. The prover computes $z_1 = \alpha + cR \mod q, z_2 = \beta + cx \mod q, z_3 = \gamma + cr \mod q$ and sends (z_1, z_2, z_3) to the verifier.
4. The verifier checks if $a_1 = g_1^{-z_1} e_1^c, a_2 = g_1^{-z_2} h_1^{-z_1} e_2^c, a_3 = g^{-z_2} h^{-z_3} C^c$. If yes, the verifier accepts.

B.5 Proof of Equality of Aggregated Discrete Logs & Commitments

We now describe a protocol for proving equality of the discrete logarithms $(a_1, \ldots, a_n)$ in $y = \prod_{i=1}^n G_i^{a_i}$ and individual algebraic commitments to them. Using standard notation, we denote the protocol by $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : y = \prod_{i=1}^n G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$.

Let $\mathbb{G}$, $\mathbb{H}$ be groups of prime order q. Given $y = \prod G_i^{a_i}$ and $C_i = g^{a_i} h^{r_i}$, where G_i are generators of the group $\mathbb{G}$, g is a generator of $\mathbb{H}$ and h is a random element in $\mathbb{H}$. The prover does not know the discrete logarithm of h with respect to g, and the discrete logarithms of G_is with respect to each other. We want to prove equality of the discrete logarithms in y and the respective values committed to in C_is.

Protocol comEq : $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : y = \prod_{i=1}^{n} G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$

Given $y = \prod_{i=1}^{n} G_i^{a_i}$ and $C_i = g^{a_i} h^{r_i}$

1. The prover computes the following values: $u = \prod_{i=1}^{n} G_i^{\alpha_i}$ and $v_i = g^{\alpha_i} h^{R_i}$ for randomly chosen $\alpha_i, R_i \in \mathbb{Z}_q$ and sends u, v_i to the verifier.
2. The verifier chooses a challenge c at random from $\mathbb{Z}_{2^k}$ for a fixed k, such that $2^k < q$, and sends it to the prover.
3. For a challenge string c, prover computes and sends the tuple (s_i, t_i)

$$s_i = \alpha_i - c a_i \pmod{q}, \qquad t_i = R_i - c r_i \pmod{q}$$

4. Verification: Check if $u = y^c \prod G_i^{s_i}$ and $v_i = (C_i)^c g^{s_i} h^{t_i}$. The verifier accepts if checks succeed for all i.

B.6 Proof of Aggregated Discrete Logs

This is a specialization of the previous protocol and proves knowledge of discrete logarithms $(a_1, \ldots, a_n)$ in $y = \prod_{i=1}^{n} G_i^{a_i}$. Protocol is denoted by $\mathsf{PK}\{(a_1, \ldots, a_n) : y = \prod_{i=1}^{n} G_i^{a_i}\}$. The protocol is derived by simply ignoring in the previous protocol everything related to the commitments C_i.

Protocol AggregateDL : $\mathsf{PK}\{(a_1, \ldots, a_n) : y = \prod_{i=1}^{n} G_i^{a_i}\}$

Given $y = \prod_{i=1}^{n} G_i^{a_i}$

1. The prover computes the following values: $u = \prod_{i=1}^{n} G_i^{\alpha_i}$ for randomly chosen $\alpha_i \in \mathbb{Z}_q$ and sends u to the verifier.
2. The verifier chooses a challenge c at random from $\mathbb{Z}_{2^k}$ for a fixed k, such that $2^k < q$, and sends it to the prover.
3. For a challenge string c, prover computes and sends the tuple (s_i)

$$s_i = \alpha_i - c a_i \pmod{q}$$

4. Verification: Check if $u = y^c \prod G_i^{s_i}$. The verifier accepts if checks succeed for all i.

B.7 Proof of Equality for Commitments in Different Groups

This is essentially a specialization of the protocol from section B.5. Assume we have discrete logarithms (a_1, a_2) in $y = G_1^{a_1} G_2^{a_2}$ which can be seen as a commitment to a_1 with randomness a_2, as well as a commitment to a_1 in a different group. Using standard notation, we denote the protocol by $\mathsf{PK}\{(a_1, a_2, r) : y = G_1^{a_1} G_2^{a_2} \wedge C = g^{a_1} h^r\}$.

Let $\mathbb{G}$, $\mathbb{H}$ be groups of prime order q. Given $y = G_1^{a_1} G_2^{a_2}$ and $C = g^{a_1} h^r$, where G_i are generators of the group $\mathbb{G}$, g is a generator of $\mathbb{H}$ and h is a random element in $\mathbb{H}$. The prover does not know the discrete logarithm of h with respect to g, and the discrete logarithms of G_is with respect to each other. We want to prove that y and C are commitments to the same value.

Protocol $\mathsf{PK}\{(a_1, a_2, r) : y = G_1^{a_1} G_2^{a_2} \wedge C = g^{a_1} h^r\}$

Given $y = G_1^{a_1} G_2^{a_2}$ and $C = g^{a_1} h^r$

1. The prover computes the following values: $u = G_1^{\alpha_1} G_2^{\alpha_2}$ and $v = g^{\alpha_1} h^R$ for randomly chosen $\alpha_i, R \in \mathbb{Z}_q$ and sends u, v to the verifier.
2. The verifier chooses a challenge c at random from $\mathbb{Z}_{2^k}$ for a fixed k, such that $2^k < q$, and sends it to the prover.
3. For a challenge string c, prover computes and sends the tuple (s_1, s_2, t), computed as follows:

$$s_1 = \alpha_1 - ca_1 \pmod q,\ s_2 = \alpha_2 - ca_2 \pmod q,\ t = R - cr \pmod q$$

4. Verification: Check if $u = y^c \prod G_1^{s_1} G_2^{s_2}$ and $v = C^c g^{s_1} h^t$. The verifier accepts if all checks succeed.

B.8 Proof of multiplicative relation on committed values

This protocol proves, for committed values a_1, a_2, a_3 in commitments C_1, C_2, C_3, that $a_1 a_2 = a_3 \bmod q$. Protocol is denoted by $\mathsf{PK}\{(a_1, a_2, a_3, r_1, r_2, r_3) : C_i = g^{a_i} h^{r_i} \text{ for } i = 1..3 \wedge a_1 a_2 = a_3 \bmod q\}$.

Protocol $\mathsf{comMult} : \mathsf{PK}\{(a_1, a_2, a_3, r_1, r_2, r_3) : C_i = g^{a_i} h^{r_i} \text{ for } i = 1..3 \wedge a_1 a_2 = a_3 \bmod q\}$

Given $C_i = g^{a_i} h^{r_i}$ for $i = 1, 2, 3$, we will prove the relation $a_1 a_2 = a_3 \bmod q$ by proving that $C_3 = C_1^{a_2} h^r$ in parallel with proving that $C_i = g^{a_i} h^{r_i}$ for $i = 1, 2, 3$. The condition is satisfied if we set $r = r_3 - r_1 a_2 \bmod q$, so the prover can do this.
If the proof succeeds, it follows that C_3 can be opened to both a_3 and $a_1 a_2$, so we get what we want unless the binding condition is broken.

1. The prover computes the following values: $v_i = g^{\alpha_i} h^{R_i}$ for $i = 1, 2, 3$, $v = C_1^{\alpha_2} h^R$ for randomly chosen $\alpha_i, R_i, \alpha, R \in \mathbb{Z}_q$ and sends $\{v_i\}, v$ to the verifier.

   Note that v will later be used in a verification equation to test $C_3 = C_1^{a_2} h^r$. This is why we need to use α_2 in the exponent of C_1 in the expression for v.
2. The verifier chooses a challenge c at random from $\mathbb{Z}_{2^k}$ for a fixed k, such that $2^k < q$, and sends it to the prover.
3. For a challenge string c, prover computes and sends the tuple (s_i, t_i) for $i = 1, 2, 3$ and t:

$$s_i = \alpha_i - ca_i \pmod q,\ t_i = R_i - cr_i \pmod q,\ t = R - cr \pmod q$$

4. Verification: Check if $v_i = (C_i)^c g^{s_i} h^{t_i}$ for $i = 1, 2, 3$ and $v = C_3^c C_1^{s_2} h^t$. The verifier accepts if all checks succeed.

C Protocols for ID layer

C.1 Protocol for Π_{issue}

Protocol For proving that $ct = \mathsf{TEnc}^{\mathsf{n},\mathsf{d}}_{\mathsf{PK}_{\mathsf{AR}}}(m; r)$

Recall that we follow the share-and-encrypt paradigm for the threshold encryption by using Shamir Secret sharing and the CL encryption scheme. This means that the encryption of m is supposed to be consisting of n ciphertexts $\mathsf{CL.Enc}(\mathsf{pk}_{\mathsf{AR}_i}, \mathsf{sh}(m)_i)$ for $i = 1, \ldots, \mathsf{n}$ where $\mathsf{sh}(m)_i$ is the i'th share of m.
Let M' be the commitment of m on group $\widetilde{\mathbb{G}}$. The prover AH proves to the verifier IP that M' contains the same value m as does ct:

1. AH makes a commitment B_0 to m under the default commitment key ck (which is in the CRS), so $B_0 \in \mathbb{H}$. They use the protocol from Section B.7 to show that M' and B_0 both contain m.

2. AH establishes a commitment M_i to each share $\mathsf{sh}(m)_i$ of m as follows: when AH secret-shares m, they use a polynomial g to get shares $\mathsf{sh}(m)_i$. Let the coefficients of g be $b_0 = m, b_1, ..., b_{\mathsf{d}}$. Note that we have $\mathsf{sh}(m)_i = g(i) = \sum_{j=0}^{\mathsf{d}} b_j \cdot i^j$. Now, the AH makes and sends commitments B_j to b_j, where we notice that B_0 has already been constructed above. Using the homomorphic property of commitments one can compute $K_i = \prod_{j=0}^{\mathsf{d}} B_j^{i^j}$ which is a commitment to $\mathsf{sh}(m)_i$. Since AH, the prover, has created the B_is, they also know how to open the K_is.

3. AH finally runs a protocol similar to $\mathsf{com{-}enc{-}eq}$ from section B.4 but for the CL encryption scheme to show the IP that each $\mathsf{CL.Enc}(\mathsf{pk}_{\mathsf{AR}_i}, \mathsf{sh}(m)_i)$ contains the same value as K_i for $i = 1...\mathsf{n}$.

C.2 Protocol for $\Pi_{\mathsf{id-layer}}$

Proving that you know a signature. The user has a signature $\sigma = (a, b)$ on a message $(m_0, ..., m_\ell)$ and wants to convince a verifier of this fact. This is done as follows:

Protocol For proving knowledge of a signature

1. Choose $r, r' \leftarrow \mathbb{F}_q$, and compute a blinded version of the signature as follows:

$$\hat{a} = a^r, \hat{b} = (b \cdot a^{r'})^r$$

   and sends the blinded signature $(\hat{a}, \hat{b})$ to the verifier.

2. Both parties compute locally the following values:

$$v_1 = e(\hat{a}, \tilde{g}), v_2 = e(\hat{b}, \tilde{g}), v_3 = e(\hat{a}, \tilde{X}), u_i = e(\hat{a}, \tilde{Y}_i)$$

3. The verifier checks that $\hat{a} \neq 1_{\mathbb{G}}$. If not, the verifier rejects. The user gives a ZK proof of knowledge:

$$PK((m_0, ..., m_\ell, r') : v_2 = v_3 \cdot v_1^{r'} \prod_{i=1}^{\ell} u_i^{m_i})$$

4. The verifier accepts if the proof is valid.

Adaptation for Implementing $\Pi_{\mathsf{id-layer}}$. In the ID layer, the AH will act as user/prover and anyone else can act as verifier. However, the above protocol needs to be further adapted so we can show specific properties of attributes: recall that we set $(m_0, ..., m_\ell) = (r, \mathsf{IDcred}_{\mathsf{SEC}}, \mathsf{K}, a_1, ..., a_v)$, padding the right-hand side with 0's if needed to get ℓ entries. The AH can supply also commitments to all fields and attributes: $i = 1...\ell : C_i = \mathsf{Commit}_{\mathsf{ck}}(m_i, r_i)$. So we execute the protocol exactly as specified above, but the proof of knowledge in step 3 is extended as follows:

$$PK((m_0, ..., m_\ell, r') : v_2 = v_3 \cdot v_1^{r'} \prod_{i=1}^{\ell} u_i^{m_i},\ C_i = \mathsf{Commit}_{\mathsf{ck}}(m_i, r_i) \text{ for } i = 1...\ell).$$

For this proof of knowledge, we use the comEq protocol from section B.5-
The AH can now use the C_i to prove desired properties of the fields and attributes. This is fleshed out later in this appendix.

Protocol For proving that $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$ for some $x \leq \mathsf{Max}_{\mathsf{ACC}}$

Recall that $\mathsf{PRF}_{\mathsf{K}}(x) = \bar{g}^{1/(x+\mathsf{K})}$. This value can be seen as $\mathsf{Commit}_{\mathsf{ck}}(1/(\mathsf{K}+x))$ where the randomness is 0. Let $C_1 = \mathsf{Commit}_{\mathsf{ck}}(\mathsf{K})$ and $C_3 = \mathsf{Commit}_{\mathsf{ck}}(a_1) = \mathsf{Commit}_{\mathsf{ck}}(\mathsf{Max}_{\mathsf{ACC}})$.

1. AH makes commitments $F_x = \mathsf{Commit}_{\mathsf{ck}}(x)$ and $F_1 = g$ which can be written as $\mathsf{Commit}_{\mathsf{ck}}(1)$ where the randomness is 0. Note that $C_1 F_x = \mathsf{Commit}_{\mathsf{ck}}(\mathsf{K} + x)$.

2. AH runs Protocol comMult from Section B.8 with input commitments $C_1 F_x, \mathsf{RegID}_{\mathsf{ACC}}$ and F_1. This shows that $\mathsf{RegID}_{\mathsf{ACC}} = \mathsf{PRF}_{\mathsf{K}}(x)$.

3. AH finally uses a zkSNARK on committed inputs based on F_x and C_3 to show that

$x \leq \mathsf{Max}_{\mathsf{ACC}}$.

Protocol For proving that $\mathsf{E}_{\mathsf{ID}} = \mathsf{TEnc}^{\mathsf{n,d}}_{\mathsf{PK}_{\mathsf{AR}}}(\mathsf{IDcred}_{\mathsf{PUB}}; \mathsf{r}')$

Let C_0 be the commitment to $\mathsf{IDcred}_{\mathsf{SEC}}$. Recall that we follow share-and-encrypt paradigm for the threshold encryption (see A.4.1) which means in order to encrypt $\mathsf{IDcred}_{\mathsf{PUB}}$ correctly, AH first secret shares $\mathsf{IDcred}_{\mathsf{SEC}}$ using a polynomial f to get shares $\mathsf{sh}(\mathsf{IDcred}_{\mathsf{SEC}})_{\mathsf{i}}$ and then computes the i'th component of E_{ID} as Elgamal encryption $\mathsf{Enc}_{\mathsf{pk}_{\mathsf{AR}_i}}(\bar{g}^{\mathsf{sh}(\mathsf{IDcred}_{\mathsf{SEC}})_{\mathsf{i}}}; r'_i)$. Let the coefficients of f be $a_0 = \mathsf{IDcred}_{\mathsf{SEC}}, a_1, ..., a_{\mathsf{d}}$. Note that we have $\mathsf{sh}(\mathsf{IDcred}_{\mathsf{SEC}})_{\mathsf{i}} = \mathsf{f}(\mathsf{i}) = \sum_{\mathsf{j}=0}^{\mathsf{d}} \mathsf{a}_{\mathsf{j}} \cdot \mathsf{i}^{\mathsf{j}}$. Now,

1. AH makes commitments A_j to a_j, where we set $A_0 = C_0$. Using the homomorphic property of commitments one can compute $S_i = \prod_{j=0}^{\mathsf{d}} A_j^{i^j}$ which is a commitment to $\mathsf{sh}(\mathsf{IDcred}_{\mathsf{SEC}})_{\mathsf{i}}$. Since AH, the prover, has created the A_is, they also know how to open the S_is.

2. For $i = 1...\mathsf{n}$, the AH runs the protocol $\mathsf{com{-}enc{-}eq}$ from section B.4 using the two components in $E_{\mathsf{pk}_{\mathsf{AR}_i}}(\bar{g}^{\mathsf{sh}(\mathsf{IDcred}_{\mathsf{SEC}})_{\mathsf{i}}}; r'_i)$ as e_1, e_2 and S_i as C.

D CL Framework

In [CL15], Castagnos and Laguillaumie introduced the so-called CL framework, which allows to construct cyclic groups $\mathbb{G}$ where the decisional Diffie-Hellman (DDH) assumption is believed to hold and furthermore, there exists a subgroup $\mathbb{H}$ of $\mathbb{G}$ such that the discrete logarithm problem in $\mathbb{H}$ is easy. As the main application of this framework, they construct the first practical linearly homomorphic encryption scheme where the plaintext space is $\mathbb{Z}_q$ with prime order q, where the size of q can be made independent of the security parameter. We make use of the CL encryption schemes in some of our constructions. In particular, the CL framework allows to encrypt values "in the exponent", so that proving statements about the message is easy using efficient Σ-protocols in [CCL+20], while at the same time allowing for efficient decryption. (As opposed to standard ElGamal encryption, where it is necessary to limit the size of the message in the exponent to allow for decryption). This will be useful both to instantiate our threshold encryption scheme (used to allow anonymity revokers to "trace" a suspicious account holder), and in our construction of extractable NIZK proofs in the crs model (see F).

In the following, we recall the definition of CL framework from [CCL+20].

D.1 Definition

Let λ be a positive integer and q be a μ-bit prime for $\mu \geq \lambda$. The framework $\mathsf{Gen}_{\mathsf{CL}}$ consists of two algorithms $\mathsf{Gen}_{\mathsf{CL}} = (\mathsf{Gen}, \mathsf{Solve})$ defined as follows:

- $\mathsf{Gen}(1^\lambda, q) \to \mathsf{p} := (\widehat{\mathbb{G}}, \mathbb{G}, \mathbb{H}, \mathbb{G}^q, \tilde{s}, g, h, g_q)$: the algorithm takes the security parameter λ and a prime q as inputs and outputs a tuple $\mathsf{p} := (\widehat{\mathbb{G}}, \mathbb{G}, \mathbb{H}, \mathbb{G}^q, \tilde{s}, g, h, g_q)$, where

- $(\widehat{\mathbb{G}}, \cdot)$ is a finite abelian group with order $\widehat{n} := q \cdot \widehat{s}$ where the bitsize of $\widehat{s}$ is a function of λ and $\gcd(q, \widehat{s}) = 1$. It is required that valid encodings of elements in $\widehat{\mathbb{G}}$ can efficiently be recognized.
- $(\mathbb{G}, \cdot)$ is a cyclic subgroup of $\widehat{\mathbb{G}}$ of order $n := q \cdot s$ where s divides $\widehat{s}$.
- $(\mathbb{H}, \cdot)$ is the unique cyclic subgroup of $\widehat{\mathbb{G}}$ of order q, generated by h.
- $\mathbb{G}^q := \{x^q | x \in \mathbb{G}\}$ is the subgroup of $\mathbb{G}$ of order s. Since $\mathbb{H} \subset \mathbb{G}$, it holds that $\mathbb{G} \simeq \mathbb{G}^q \times \mathbb{H}$.
- $\tilde{s}$ is an upper bound of $\widehat{s}$.
- g, h and g_q are respectively the generators of $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{G}^q$, where $g := h \cdot g_q$.

- $\mathsf{Solve}(q, \mathsf{p}, X) \to x'$: this is a DPT algorithm that solves the discrete logarithm problem in $\mathbb{H}$:

$$\Pr\Big[x = x' : \mathsf{p} \leftarrow\!\$\ \mathsf{Gen}(1^\lambda, q), x \leftarrow\!\$\ \mathbb{Z}/q\mathbb{Z}, X \leftarrow h^x \\ x' \leftarrow \mathsf{Solve}\,(q, \mathsf{p}, X)\Big] = 1$$

D.2 Hard Subgroup Membership Problem

We now recall the hard subgroup membership problem within a group with an easy DL subgroup (HSM-CL) from [CLT18]. The HSM-CL assumption states that it is hard to distinguish the elements of $\mathbb{G}^q$ in $\mathbb{G}$.

Definition D.1 (HSM-CL assumption)**.** *Let λ be a positive integer and $\mathsf{Gen}_{\mathsf{CL}} = (\mathsf{Gen}, \mathsf{Solve})$ be a generator that generates a group with an easy DL subgroup as defined above. Let $\mathcal{D}$ (resp. $\mathcal{D}_q$) be a distribution over the integers such that the distance between the distribution $\{g^x, x \leftarrow\!\$\ \mathcal{D}\}$ (resp. $\{g_q^x, x \leftarrow\!\$\ \mathcal{D}_q\}$) and uniform distribution in $\mathbb{G}$ (resp. in $\mathbb{G}^q$) is less than $\delta(\lambda) = \mathsf{negl}(\lambda)$. The advantage of an adversary $\mathcal{A}$ for the HSM-CL problem is now defined as*

$$\mathrm{Adv}^{\mathsf{hsm\text{-}cl}}_{\mathcal{A}}(\lambda) = |\Pr[b = b' : \mathsf{p} \leftarrow\!\$\ \mathsf{Gen}(1^\lambda, q), |q| \geq \lambda, x \leftarrow\!\$\ \mathcal{D}, x' \leftarrow\!\$\ \mathcal{D}_q, \\ b \leftarrow\!\$\ \{0,1\}, X_0 \leftarrow g^x, X_1 \leftarrow g_q^{x'}, b' \leftarrow \mathcal{A}\,(q, \mathsf{p}, X_b, \mathsf{Solve}(\cdot))] - 1/2|$$

We say that the HSM-CL problem is ε-hard for $\mathsf{Gen}_{\mathsf{CL}}$ if for all PPT adversary $\mathcal{A}$, $\mathrm{Adv}^{\mathsf{hsm\text{-}cl}}_{\mathcal{A}}(\lambda) \leq \varepsilon(\lambda)$. And we say HSM-CL holds for $\mathsf{Gen}_{\mathsf{CL}}$ if it is ε-hard for $\mathsf{Gen}_{\mathsf{CL}}$ and $\varepsilon(\lambda) = \mathsf{negl}(\lambda)$.

D.3 PKE scheme under HSM-CL assumption

We recall the scheme described in [CLT18] with the following setting: the plaintext space is $\mathbb{Z}_q$ for μ-bit prime q and $\mu \geq \lambda$; and the secret key x and randomness r are drawn from distribution $\mathcal{D}_q$. As shown in [CLT18] (lemma 4), to instantiate $\mathcal{D}_q$ in practice, one can use the uniform distribution on $\{0, \ldots, S\}$ for $S := 2^{\lambda-2} \cdot \tilde{s}$. The scheme is depicted in Fig.

Theorem D.2. *[CLT18] The scheme described in Fig. 6 is semantically secure under chosen plaintext attacks (IND-CPA) under the HSM-CL assumption.*

$\mathsf{CL.Setup}(1^\lambda, 1^\mu)$

1. sample a μ-bit prime q
2. $\mathsf{p} \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda, q)$
3. return p

$\mathsf{CL.KeyGen}(\mathsf{p})$

1. sample $\alpha \leftarrow\!\!\$\ \mathcal{D}_q$ and $y \leftarrow g_q^\alpha$
2. set $\mathsf{pk} := y$ and $\mathsf{sk} := \alpha$
3. return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{CL.Enc}(\mathsf{pk}, m)$

1. sample $r \leftarrow\!\!\$\ \mathcal{D}_q$
2. return $(g_q^r, h^m y^r)$

$\mathsf{CL.Dec}(\mathsf{sk}, (c_1, c_2))$

1. compute $M \leftarrow c_2 / c_1^\alpha$
2. return $\mathsf{Solve}(q, \mathsf{p}, M)$

Figure 6: Encryption scheme from HSM-CL [CLT18]

E Transaction Layer

Our paper shows how account holders in our design can create accounts after registering with identity providers. As we have seen, an account includes an account-specific public key where the account holder has the private key. Since our protocol is completely generic, the "key" can in fact be a vector of keys which includes a key for encryption, one for signatures, etc. and therefore accounts can be used for transactions in many different ways.

In this section, we include an informal presentation of one way in which accounts can be used for transferring money on the blockchain, assuming each account has a public signing key and a public encryption key. We stress that this is just as example of one of many possible ways of doing this and, as we prove our ID layer secure in the UC framework one can use our ID layer with any other transaction layer, or even other applications not involving payments.

The system we sketch supports plaintext transactions and encrypted transactions. All transactions must be signed by the account holder from which the transaction originates. When a transaction is published, the ledger will check the signature and allow the transaction to go through if the signature is valid, and possibly if other constraints (described in detail below) are satisfied. Note that payments are linked to the sending and the receiving account and if several payments are made between the same accounts this will be visible on chain. We allow this to have a trade-off between efficiency and privacy: one can make several payments from an account and only suffer

the cost of opening it once. On the other hand, one can also choose to use each account once for complete privacy.

Let ACC be an account with encryption key $\mathsf{ek}_{\mathsf{ACC}}$. An account will hold a public amount p and a secret amount s. The secret amount is represented as a set $\mathcal{S} = \{S_i\}_{i=1}^{|\mathcal{S}|}$, where $S_i = \mathsf{Enc}_{\mathsf{ek}_{\mathsf{ACC}}}(s_i)$ and $s = \sum_i s_i$.

E.0.1 Plaintext Transactions.

Plaintext transactions happens by a matched reduction and increment of the public amounts on the sending and receiving accounts.

E.0.2 Encrypted Transactions.

Let $\mathcal{S}_1 = \{S_i\}_{i=1}^n$ be the representation of the secret amount for account ACC_1 owned by AH_1, where $S_i = \mathsf{Enc}_{\mathsf{ek}_{\mathsf{ACC}_1}}(s_i)$. To do an encrypted transaction of amount a from ACC_1 to some account ACC_2, AH_1 proceeds as follows:

1. Compute $s' = \sum_{i=1}^n s_i - a$.
2. Create $S' = \mathsf{Enc}_{\mathsf{ek}_{\mathsf{ACC}_1}}(s')$.
3. Create $A = \mathsf{Enc}_{\mathsf{ek}_{\mathsf{ACC}_2}}(a)$.
4. Compute a NIZK proof π that $(S', A, S_1, \ldots, S_n)$ contains numbers $(s', a, s_1, \ldots, s_n)$ such that

$$a \geq 0, \tag{1}$$

$$s' \geq 0, \tag{2}$$

$$s' = \sum_{i=1}^{n} s_i - a. \tag{3}$$

The transaction contains (S', A, π). When the transaction is executed, the ledger checks the proof and if it is valid, then let $\mathcal{S}_1 = \{S'\}$ and $\mathcal{S}_2 = \mathcal{S}_2 \cup \{A\}$, where $\mathcal{S}_2$ is the representation of the secret amount for ACC_2.

E.0.3 Compressing an account.

From the above, it follows that after receiving some number of transactions, the receiving account will contain several encrypted amounts, and this may become impractical to handle at some point. Also, note that an account owner does not actually know the amount on his account until he has decrypted the transactions that come into his account. To solve this, the account owner of ACC_2 can execute a compression transaction, which works as follows:

1. Let $\mathcal{S} = \{S_i\}$ be the representation of existing secret amount for ACC_2. Use decryption key $\mathsf{dk}_{\mathsf{ACC}_2}$ to decrypt all S_i and let s be the sum of the decrypted amounts. Compute $e = \mathsf{Enc}_{\mathsf{ek}_{\mathsf{ACC}_2}}(s)$ and let $\mathcal{S}' = \{e\}$. Compute a NIZK proof π showing that e contains the sum of the amounts in the S_i's.
2. publish a compression transaction that contains the identity of the account and $\mathcal{S}', \pi$.

When a compression transaction appears, the ledger checks the proof π and if it is valid, then update the account of ACC_2 to contain $\mathcal{S}'$ instead of $\mathcal{S}$.

F Universally Composable Non-interactive Zero-Knowledge

F.1 Simulation Extractable NIZK

In this section we briefly recall how to construct a simulation extractable NIZK $\Pi' = (\mathsf{CRSGen}', \mathsf{Prove}', \mathsf{Verify}')$ from a simulation-sound NIZK $\Pi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ and a CPA-secure encryption scheme $\mathsf{ES} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. While this is a folklore compiler and similar result appears in [Bag19], we give the complete proof here for the sake of completeness.

In Figure 7 is described the construction of the simulation-sound NIZK $\Pi' = (\mathsf{CRSGen}', \mathsf{Prove}', \mathsf{Verify}')$ for the NP-language $\mathcal{L}$. $\Pi = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ is associated to the NP-language $\mathcal{L}'$ and the relation $\mathcal{R}' = \{(\mathsf{x}, \mathtt{enc}, \mathsf{pk}) : (\mathsf{sk}, \mathsf{w}) \text{ s.t. } \mathsf{w} \leftarrow \mathsf{Dec}(\mathtt{enc}, \mathsf{pk}, \mathsf{sk}) \wedge (\mathsf{x}, \mathsf{w}) \in \mathcal{R}\}$.

- $\mathsf{CRSGen}'(1^\lambda, \mathcal{L})$:
  - Run $(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \mathsf{CRSGen}(1^\lambda, \mathcal{L}')$
  - Run $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - Output $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk})$
- $\mathsf{Prove}'(\mathsf{crs}', \mathsf{x}, \mathsf{w})$:
  - Compute $\mathtt{enc} \leftarrow \mathsf{Enc}(\mathsf{sk}, \mathsf{w})$
  - Compute $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}', (\mathsf{x}, \mathtt{enc}), \mathsf{w})$
  - Output π
- $\mathsf{Verify}'(\mathsf{crs}', \mathsf{x}, \pi)$ outputs 1 if and only if $\mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi) = 1$

Figure 7: Simulation-extractable NIZK.

Theorem F.1. *Assume that* $\mathsf{ES} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a CPA-secure encryption scheme and* $\Pi = (\mathsf{CRSGen}, \mathit{Prove}, \mathit{Verify})$ *is a simulation-sound NIZK for the NP-language* $\mathcal{L}'$*. Then* $\Pi' = (\mathsf{CRSGen}', \mathit{Prove}', \mathit{Verify}')$ *for the NP-language* $\mathcal{L}$*, described in Figure 7, is simulation extractable.*

Proof. (Sketch.) Let Sim the corresponding simulator associated to Π. Let us now describe the simulator-extractor Sim' of Π'.

CRS generation: Sim' runs Sim to obtain the simulated CRS $\mathsf{crs}, \tau, \mathsf{ek}$ and computes $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. They set the $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk})$.

Simulation: When $\mathcal{A}$ asks to see a proof for the statement x, Sim' computes an encryption of a dummy value $\mathtt{enc}$ and runs Sim on input $(\mathsf{x}, \mathtt{enc}, \tau, \mathsf{ek})$ to obtain π.

Extraction: On input $\overline{\pi'} = (\overline{\pi}, \overline{\mathtt{enc}})$ received by $\mathcal{A}$ the simulator Sim runs the decryption algorithm of ES and outputs $\overline{\mathsf{w}}$ for theorem $\overline{\mathsf{x}}$.

We first observe that the zero-knowledge property of Π' follows from the zero-knowledge property of Π and the CPA-security of ES.

The proof proceeds by contradiction. Let us assume that $\mathcal{A}$ is able to produce valid proofs but the extraction procedure described above fails with non-negligible probability. If this is the

case, then $\mathcal{A}$ is able to produce valid proofs (different from the one received from the oracle) for the statements of the form $(\overline{\mathsf{x}}, \overline{\mathtt{enc}})$, where $\overline{\mathtt{enc}}$ is not an encryption of a valid $\overline{\mathsf{w}}$ for $\mathsf{x} \in \mathcal{L}$ (by assumption) and therefore we can show a reduction to the simulation-sound property of Π.

□

F.2 SNARK-Lifting transformations via CØCØ framework [KZM+15]

To transform a SNARK into a strong simulation extractable SNARK, Kosba et al. [KZM+15] makes use of four primitives: a pseudo-random function, a perfectly-binding commitment scheme, a one-time signature (OTS) and a public key encryption (PKE) scheme. Let $\mathsf{PRF}_{\mathsf{K}}$ be a pseudorandom function with key K such that without the knowledge of K, $\mathsf{PRF}_{\mathsf{K}}(\cdot)$ behaves like a random function, whereas given K, computing $\mathsf{PRF}_{\mathsf{K}}(\cdot)$ is easy. Let $\mathsf{COM} = (\mathsf{Commit}, \mathsf{OpenVrf})$ be a commitment scheme. Let $\Omega = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme and $\Sigma_{\mathsf{OT}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VerifySig})$ be a OTS scheme. Given a language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, let $\mathcal{L}'$ be the language with relation $\mathcal{R}_{\mathcal{L}'}$ defined as follows: $\{(\mathsf{x}, c, \mu, \mathsf{pk}_{\mathsf{OT}}, \mathsf{pk}_{\mathsf{e}}, \rho), (\mathsf{w}, r_1, r_0, \mathsf{K})\} \in \mathcal{R}_{\mathcal{L}'}$ iff:

$$\mathsf{c} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{e}}}(\mathsf{w}; r_1) \wedge \Big((\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathcal{L}} \ \vee \\ (\mu = \mathsf{PRF}_{\mathsf{K}}(\mathsf{vk}_{\mathsf{OT}}) \wedge \rho = \mathsf{Commit}(\mathsf{K}; r_0))\Big)$$

The transformation from a NIZK Π to one that satisfies SSE is then defined in Fig. 8. The following theorem is taken from [KZM+15].

Theorem F.2. *Assume that the NIZK scheme* $\Pi = (\mathsf{CRSGen}, \mathit{Prove}, \mathit{Verify})$ *satisfies perfect completeness, computational soundness, computational zero-knowledge, that the encryption scheme* $\Omega = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure and perfectly correct, that the pseudo-random function family* $\mathsf{PRF}_{\mathsf{K}}$ *is secure, that the commitment scheme* $\mathsf{COM} = (\mathsf{Commit}, \mathsf{OpenVrf})$ *is perfectly binding and computational hiding, and that the one-time signature scheme is strongly unforgeable. Then the resulting transformation described in Fig. 8 is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and strongly simulation extractability as defined in Section A.6.*

G Standard Ideal Functionalities

We describe here some of the (standard) ideal functionalities that are used in our protocol design.

Functionality $\mathcal{F}_{\mathsf{crs}}$

The functionality is parametrized by a distribution $\mathcal{D}$ and proceeds as follows.

- Choose a value $\mathsf{crs} \leftarrow\!\!\$\ \mathcal{D}$.
- On input $(\mathtt{CRS})$ from a party P, return $(\mathtt{CRS}, \mathsf{crs})$ to P.

$\mathsf{CRSGen}(1^\lambda, \mathcal{L})$

- $\Pi.\mathsf{crs} \leftarrow \Pi.\mathsf{CRSGen}(1^\lambda, \mathcal{L}'); (\mathsf{pk_e}, \mathsf{sk_e}) \leftarrow \Omega.\mathsf{KeyGen}(1^\lambda);$
- $\tau := (s_0, r_0) \leftarrow\$ \{0,1\}^\lambda; \rho \leftarrow \mathsf{Commit}(s_0; r_0);$
- **return** $(\mathsf{crs} := (\Pi.\mathsf{crs}, \mathsf{pk_e}, \rho), \tau, \mathsf{ek} := \mathsf{sk_e})$

$\mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w})$

- Parse $\mathsf{crs} := (\Pi.\mathsf{crs}, \mathsf{pk_e}, \rho);$
- $(\mathsf{vk_{OT}}, \mathsf{sk_{OT}}) \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{KeyGen}(1^\lambda); r_1, z_0, z_1, z_2 \leftarrow\$ \{0,1\}^\lambda;$
- $\mathsf{c} = \Omega.\mathsf{Enc}(\mathsf{pk}_e, \mathsf{w}; r_1);$
- $\pi_\Pi \leftarrow \Pi.\mathsf{Prove}(\Pi.\mathsf{crs}, (\mathsf{x}, \mathsf{c}, z_0, \mathsf{pk}_e, \mathsf{vk_{OT}}, \rho), (\mathsf{w}, r_1, z_1, z_2));$
- $\sigma_{\mathsf{OT}} \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{Sign}(\mathsf{sk_{OT}}, (\mathsf{x}, \mathsf{c}, z_0, \pi_\Pi));$
- **return** $\pi := (\mathsf{c}, z_0, \pi_\Pi, \mathsf{vk_{OT}}, \sigma_{\mathsf{OT}});$

$\mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi)$

- Parse $\mathsf{crs} := (\Pi.\mathsf{crs}, \mathsf{pk_e}, \rho)$ and $\pi := (\mathsf{c}, \mu, \pi_\Pi, \mathsf{vk_{OT}}, \sigma_{\mathsf{OT}});$
- **if** $\Sigma_{\mathsf{OT}}.\mathsf{Verify}(\mathsf{vk_{OT}}, (\mathsf{x}, \mathsf{c}, \mu, \pi_\Pi), \sigma_{\mathsf{OT}}) = 0$
- $\vee\, \Pi.\mathsf{Verify}(\Pi.\mathsf{crs}, (\mathsf{x}, \mathsf{c}, \mu, \mathsf{pk}_e, \mathsf{vk_{OT}}, \rho), \pi_\Pi) = 0$
- **then return** 0 **else return** $1;$

$\mathsf{SimProve}(\mathsf{crs}, \tau, \mathsf{x})$

- Parse $\mathsf{crs} := (\Pi.\mathsf{crs}, \mathsf{pk_e}, \rho)$ and $\tau := (s_0, r_0);$
- $(\mathsf{vk_{OT}}, \mathsf{sk_{OT}}) \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{KeyGen}(1^\lambda); \mu = \mathsf{PRF}_{s_0}(\mathsf{vk_{OT}});$
- $r_1, z_3 \leftarrow\$ \{0,1\}^\lambda; \mathsf{c} \leftarrow \Omega.\mathsf{Enc}(\mathsf{pk}_e, z_3; r_1);$
- $\pi_\Pi \leftarrow \Pi.\mathsf{Prove}(\Pi.\mathsf{crs}, (\mathsf{x}, \mathsf{c}, \mu, \mathsf{pk}_e, \mathsf{vk_{OT}}, \rho), (r_1, r_0, z_3, s_0));$
- $\sigma_{\mathsf{OT}} \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{Sign}(\mathsf{sk_{OT}}, (\mathsf{x}, \mathsf{c}, \mu, \pi_\Pi));$
- **return** $\pi := (\mathsf{c}, \mu, \pi_\Pi, \mathsf{vk_{OT}}, \sigma_{\mathsf{OT}});$

$\mathsf{Ext}(\mathsf{crs}, \mathsf{x}, \pi, \mathsf{ek})$

- Parse $\pi := (\mathsf{c}, \mu, \pi_\Pi, \mathsf{vk_{OT}}, \sigma_{\mathsf{OT}});$
- **return** $\mathsf{w} \leftarrow \Omega.\mathsf{Dec}(\mathsf{ek}, \mathsf{c});$

Figure 8: The strong version of C∅C∅ transformation [KZM+15]

Functionality $\mathcal{F}_{\mathsf{smt}}$

The functionality models a secure channel between a sender S and a receiver R.

- Upon input $(\texttt{SEND}, R, m)$ from a party S, if both S and R are honest, output $(\texttt{SENT}, S, m)$ to R and $(\texttt{SENT}, S, R, |m|)$ to $\mathcal{A}$. Otherwise, if at least one of S and R is corrupt, output $(\texttt{SENT}, S, R, m)$ to $\mathcal{A}$.

G.0.1 Registration Functionality

The functionality allows identity providers and anonymity revokers to input a key pair $(\mathsf{sk}, \mathsf{pk}) \in \{0,1\}^* \times \{0,1\}^*$ once such that they would not be allowed to modify or delete it later. The account holders can retrieve the public keys of registered parties by the `RETRIEVE` command.

Functionality $\mathcal{F}_{\mathsf{reg}}$

Register

Upon receiving a message $(\texttt{REGISTER}, \mathsf{sk}_P, \mathsf{pk}_P)$ from party P, if this is the first request from P, then keep the record $(P, \mathsf{sk}_P, \mathsf{pk}_P)$ and return $(\texttt{INITIALIZED}, P)$. Otherwise, ignore the message.

Retrieve

Upon receiving a message $(\texttt{RETRIEVE}, P_i)$ from some party P_j or the adversary, output $(\texttt{RETRIEVE}, P_i, \mathsf{pk}_{P_i})$ to P_j where $\mathsf{pk}_{P_i} = \perp$ if no record $(P_i, \mathsf{sk}_{P_i}, \mathsf{pk}_{P_i})$ exists.

G.0.2 NIZK Functionality

We use $\mathcal{F}_{\mathsf{nizk}}$ as defined by Groth et al. in [GOS12].

Functionality $\mathcal{F}_{\mathsf{nizk}}$

The functionality is parametrized by a relation $\mathcal{R}$ for which we can efficiently check membership.

Proof

- On input $(\texttt{PROVE}, x, w)$ from party P, ignore if $(x, w) \notin \mathcal{R}$. Send $(\texttt{PROVE}, x)$ to $\mathcal{A}$.
- Upon receiving the answer $(\texttt{PROOF}, \pi)$ from $\mathcal{A}$, store (x, π) and send $(\texttt{PROOF}, \pi)$ to P.

Verify

- On input $(\texttt{VERIFY}, x, \pi)$ from V check whether (x, π) is stored. If not send $(\texttt{VERIFY}, x, \pi)$ to $\mathcal{A}$.
- Upon receiving the answer $(\texttt{WITNESS}, w)$ from $\mathcal{A}$, check $(x, w) \in \mathcal{R}$ and if so, store (x, π). If (x, π) has been stored, output $(\texttt{VERIFICATION}, 1)$ to V, else output $(\texttt{VERIFICATION}, 0)$.

G.0.3 Ledger Functionality

The functionality $\mathcal{F}_{\mathsf{ledger}}$ (which is an adapted and simplified version of the functionality given by [KZZ16]) abstracts a public ledger which can be accessed globally by protocol parties or en-

vironment $\mathcal{Z}$. When accounts are created and posted by account holders, the ideal functionality first validates the account and then appends them to the buffer. The environment can later specify when to free the buffer and append it to the ledger.

Functionality $\mathcal{F}_{\mathsf{ledger}}$

The functionality is globally available to all parties and is parameterized by a predicate $\texttt{VALIDATE}$, an initially empty list L of bit strings and a variable buffer initially set to ε.

Append

Upon input $(\texttt{APPEND}, x)$ from party P, if $\texttt{VALIDATE}(\mathsf{state}, (\mathsf{buffer}, x)) = 1$, then set $\mathsf{buffer} \leftarrow \mathsf{buffer}||x$.

Retrieve

Upon input $(\texttt{RETRIEVE})$ from a party P or $\mathcal{A}$, return $(\texttt{RETRIEVE}, L)$ to the requestor in case of an honest party or $(\texttt{RETRIEVE}, L, \mathsf{buffer})$ in case of a corrupt party.

Buffer Release

Upon input $(\texttt{RELEASE}, \Pi)$ from $\mathcal{A}$, append a permutation Π of buffer to L by setting $L \leftarrow L||\Pi(\mathsf{buffer})$ and set $\mathsf{buffer} := \varepsilon$.

G.0.4 Functionality for MPC of PRF output

This functionality allows a set of parties who own secret shares of a PRF key to evaluate the PRF on a fixed set of inputs.

Functionality $\mathcal{F}_{\mathsf{mpc\text{-}prf}}$

The functionality interacts with parties $P_1, \ldots, P_{\mathsf{n}}$ and is parameterized by a pseudo-random function PRF, a constant $\mathsf{Max}_{\mathsf{ACC}}$, and a (n, d)-secret sharing scheme SS.

Compute

Upon input $(\texttt{COMPUTE}, \mathsf{K}_i)$ from at least $\mathsf{d}+1$ distinct P_i, proceed as follows:

- compute $\mathsf{K} = \mathsf{Reconstruct}^{\mathsf{n},\mathsf{d}}(\mathsf{K}_{i_1}, \ldots, \mathsf{K}_{i_{\mathsf{d}+1}})$.
- evaluate $\mathsf{PRF}_{\mathsf{K}}(x)$ on all inputs $x \in [\mathsf{Max}_{\mathsf{ACC}}]$ and return the list of outputs to all the requesters P_i for $i \in \mathsf{n}$.