ZKProof Standards
Applications Track Proceedings
1 August 2018 + subsequent revisions

This document is an ongoing work in progress.
Feedback and contributions are encouraged.

Track Chairs:
Daniel Benarroch, Ran Canetti and Andrew Miller

Track Participants:
Shashank Agrawal, Tony Arcieri, Vipin Bharathan, Josh Cincinnati, Joshua Daniel, Anuj Das Gupta, Angelo De Caro, Michael Dixon, Maria Dubovitskaya, Nathan George, Brett Hemenway Falk, Hugo Krawczyk, Jason Law, Anna Lysyanskaya, Zaki Manian, Eduardo Morais, Neha Narula, Gavin Pacini, Jonathan Rouach, Kartheek Solipuram, Mayank Varia, Douglas Wikstrom and Aviv Zohar

Introduction and Motivation

In this track we aim to overview existing techniques for building ZKP based systems, including designing the protocols to meet the best-practice security requirements. One can distinguish between high-level and low-level applications, where the former are the protocols designed for specific use-cases and the latter are the underlying operations needed to define a ZK predicate. We call gadgets the sub-circuits used to build the actual constraint system needed for a use-case. In some cases, a gadget can be interpreted as a security requirement (e.g.: using the commitment verification gadget is equivalent to ensuring the privacy of underlying data).

As we will see, the protocols can be abstracted and generalized to admit several use-cases; similarly, there exist compilers that will generate the necessary gadgets from commonly used programming languages. Creating the constraint systems is a fundamental part of the applications of ZKP, which is the reason why there is a large variety of front-ends available.

In this document, we present three use-cases and a set of useful gadgets to be used within the predicate of each of the three use-cases: identity framework, asset transfer and regulation compliance.

What this document is NOT about:

- A unique explanation of how to build ZKP applications
- An exhaustive list of the security requirements needed to build a ZKP system
- A comparison of front-end tools
- A show of preference for some use-cases or others

Notation and Definitions

See Security and Implementation track for definitions of predicate / prover / verifier / proof / proving key, etc.

When designing ZK based applications, one needs to keep in mind which of the following three models (that define the functionality of the ZKP) is needed:

1. Publicly verifiable as a requirement: a scheme / use-case where the proofs are transferable, where such property is actually a requirement of the system. Only non-interactive ZK (NIZK) can actually hold this property.
2. Designated verifier as a security feature: only the intended receiver of the proof can verify it, making the proof non-transferable.This property can apply to both interactive and non-interactive ZK.
3. The final model is one where neither of the above is needed: a ZK where there is no need to be able to transfer but also no non-transferability requirement. Again, this model can apply both in the interactive and non-interactive model.

For example, digital money based applications belong to the first model, compliance for regulation lives in the second model (albeit depending on the use-case). In general, the credential system can be in both of the last two models, given the extra constraints that would make it belong to the second model.

Previous works

This section will include an overview of some of the works and applications existing in the zero-knowledge world. We asked the Applications track participants to send us a description of their work. We are now In the process of collecting the content.

Gadgets within predicates

Formalizing the security of these protocols is a very difficult task, especially since there is no predetermined set of requirements, making it an ad-hoc process. Here we outline a set of initial gadgets to be taken into account. This list should be expanded continuously and on a case by case basis. For each of the gadget below, we write the following representations, specifying what is the secret / witness, what is public / statement:

NP statements for non-technical people:

> ***For*** **the [public] chess board configurations A and B;**
> ***I know*** **some [secret] sequence S of chess moves;**
> ***such that*** **when starting from configuration A, and applying S, all moves are legal and the**
> **final configuration is B.**

General form (Camenisch-Stadler): **Zk { (wit): P(wit, statement) }**
Example of ring signature: **Zk { (sig): VerifySignature(P1, sig) or VerifySignature(P2, sig) }**

Gadget Name	**English Description of the initial gadget (before adding ZKP)**	**The resulting enhanced gadget (after adding ZKP)**	**ZKP Statements (In a proof of knowledge notation)**	**Prover knows a witness...**	**...for the public instance...**	**...s.t. the following predicate holds**	**Technical notation (API)**
Commitment	Envelope	I know the value hidden inside this envelope, even though I cannot change it	Knowledge of committed value(s) (openings)	Opening(s) O = (v,r) containing a value and randomness	Committed value(s) C	C = Comm(O), componentwise if there are multiple C, O	
		I know that the value hidden inside these two envelopes are equal	Equality of committed values	Opening O	Committed values C1 and C2	C1 = Comm(O) and C2 = Comm(O)	
		I know that the values hidden inside these two envelopes are related in a specific way	Relationships between committed values -- logical, arithmetic, etc.	Witnesses O1 and O2	Committed values C1 and C2, relation R	C1 = Comm(O1), C2 = Comm(O2), and R(O1, O'2) = True	

		The value inside this envelope is within a particular range	Range proofs	Opening O	Committed value C, interval I	C = Comm(O) and O is in the range I	
Signatures			Knowledge of a signature on a message	Signature σ	Verification key VK, message M	Verify(VK, m, σ) = True	
propose: blind, ring, group, homom.			Knowledge of a signature on a committed value	Message M, signature σ	Verification key VK, committed value C	C = Comm(M) and Verify(VK, m, σ) = True	
Encryption	Envelope with a receiver stamp	The ciphertext is computed correctly and I know the plaintext	Knowledge of the plaintext	Plaintext P, KeyGen randomness R	Ciphertext C, Encryption key PK	(SK, PK) ← KeyGen(R) and Dec(SK, C) = P	
			Knowledge of the plaintext and encryption key	Plaintext P, Key pair (SK, PK)	Ciphertext C	Dec(SK, C) = P	
Distributed decryption	Envelope with a receiver stamp that requires multiple people to open	The output plaintext(s) correspond to the public ciphertext(s).	Knowledge of the plaintext	Secret shares of the decryption key	Ciphertext(s) C and Encryption key PK	Dec(SK, C) = P, componentwise if ∃ multiple C	
Random function	Lottery machine	Verifiable random function (VRF)	VRF was computed correctly from a secret seed and a public (or secret) input	Secret seed W	Input X, Output Y	Y = VRF(W, X)	
Set membership		Accumulator	Set inclusion				
			Set non-inclusion				

Mix-net	Ballot box	Shuffle	The set of plaintexts in the input and the output ciphertexts respectively are identical.	Permutation π, Decryption key SK	Input ciphertext list C and Output ciphertext list C'	Dec(SK, π(C_j)) = Dec(SK, C'_j) ∀ j	
		Shuffle and reveal	The set of plaintexts in the input ciphertexts is identical to the set of plaintexts in the output.	Permutation π, Decryption key SK	Input ciphertext list C and Output plaintext list P	Dec(SK, π(C_j)) = P_j ∀ j	
Generic circuits, TMs, or RAM programs	General calculations	There exists some secret input that makes this calculation correct	ZK proof of correctness of circuit/Turing machine/RAM program computation	Secret input w	Program C (either a circuit, TM, or RAM program), public input x, output y	C(x, w) = y	
		This calculation is correct, given that I already know that some sub-calculation is correct	ZK proof of verification + postprocessing of another output (Composition)	Secret input w	Program C with subroutine C', public input x, output y, intermediate value z = C'(x, w), zk proof π that z = C'(x, w)	C(x, w) = y	

Identity Framework

Overview

In this section we describe identity management solutions using zero knowledge proofs. The idea is that some user has a set of attributes that will be attested to by an issuer or multiple issuers, such that these attestations correspond to a validation of those attributes or a subset of them.

After attestation it is possible to use this information, hereby called a *credential*, to generate a *claim* about those attributes. Namely, consider the case where Alice wants to show that she is over 18 and lives in a country that belongs to the European Union. If two issuers were responsible for the attestation of Alice`s age and residence country, then we have that Alice could use zero knowledge proofs in order to show that she possesses those attributes, for instance she can use zero knowledge range proofs to show that her age is over 18, and zero knowledge set membership to prove that she lives in a country that belongs to the European Union. This proof can be presented to a Verifier that must validate such proof to authorize Alice to use some service. Hence there are three parties involved: (i) the credential holder; (ii) the credential issuer; (iii) and the verifier.

We are going to focus our description on a specific use case: *accredited investors*. In this scenario the credential holder will be able to show that she is accredited without revealing more information than necessary to prove such a claim.

Motivation for Identity and Zero Knowledge

Digital identity has been a problem of interest to both academics and industry practitioners since the creation of the internet. Specifically, it is the problem of allowing an individual, a company, or an asset to be identified online without having to generate a physical identification for it, such as an ID card, a signed document, a license, etc. Digitizing Identity comes with some unique risks, loss of privacy and consequent exposure to Identity theft, surveillance, social engineering and other damaging efforts. Indeed, this is something that has been solved partially, with the help of cryptographic tools to achieve moderate privacy (password encryption, public key certificates, internet protocols like tls and several others). Yet, these solutions are sometimes not enough to meet the privacy needs to the users / identities online. Cryptographic zero knowledge proofs can further enhance the ability to interact digitally and gain both privacy and the assurance of legitimacy required for the correctness of a process.

The following is an overview of the generalized version of the identity scheme. We define the terminology used for the data structures and the actors, elaborate on what features we include and what are the privacy assurances that we look for.

Terminology / Definitions

In this protocol we use several different data structures to represent the information being transferred or exchanged between the parties. We have tried to generalize the definitions as much as possible, while adapting to the existing Identity standards and previous ZKP works.

Attribute. The most fundamental information about a holder in the system (e.g.: age, nationality, univ. Degree, pending debt, etc.). These are the properties that are factual and from which specific authorizations can be derived.

(Confidential and Anonymous) Credential. The data structure that contains attribute(s) about a holder in the system (e.g.: credit card statement, marital status, age, address, etc). Since it contains private data, a credential is not shareable.

(Verifiable) Claim. A zero-knowledge predicate about the attributes in a credential (or many of them). A claim must be done about an identity and should contain some form of logical statement that is included in the constraint system defined by the zk-predicate.

Proof of Credential. The zero knowledge proof that is used to verify the claim attested by the credential. Given that the credential is kept confidential, the proof derived from it is presented as a way to prove the claim in question.

The following are the different parties present in the protocol:

Holder. The party whose attributes will be attested to. The holder *holds* the credentials that contain his / her attributes and generates Zero Knowledge Proofs to prove some claim about these. We say that the holder *presents a proof of credential* for some claim.

Issuer. The party that attests attributes of holders. We say that the issuer *issues* a credential to the holder.

Verifier. The party that verifies some claim about a holder by verifying the zero knowledge proof of credential to the claim.

Remark: The main difference between this protocol and a non-ZK based Identity protocol is the fact that in the latter, the holder presents the credentials themselves as the proof for the claim / authorization, whereas in this protocol, the holder presents a zero knowledge proof that was computed from the credentials.

The Protocol Description

Functionality. There are many interesting features that we considered as part of the identity protocol. There are four basic functionalities that we decided to include from the get go: (1) third party anonymous and confidential attribute attestations through **credential issuance** by the issuer, (2) confidentially proving claims using zero knowledge proofs through the **presentation of proof of credential** by the holder, (3) **verification of claims** through zero knowledge proof verification by the verifier and (4) unlinkable **credential revocation** by the issuer. There are further functionalities that we find interesting and worth exploring but that we did not include in this version of the protocol. Some of these are credential transfer, authority delegation and trace auditability. We explain more in detail what these are and explore ways they could be instantiated.

Privacy requirements. One should aim for a high level of privacy for each of the actors in the system, but without compromising the correctness of the protocol. We look at anonymity properties for each of the actors, confidentiality of their interactions and data exchanges, and at the unlinkability of public data (in committed form). These usually can be instantiated as cryptographic requirements such as commitment non-malleability, indistinguishability from

random data, unforgeability, accumulator soundness or as statements in zero-knowledge such as proving knowledge of preimages, proving signature verification, etc.

- Holder anonymity: the underlying physical identity of the holder must be hidden from the general public, and if needed from the issuer and verifier too. For this we use pseudo-random strings called *identifiers*, which are tied to a secret only known to the holder.
- Issuer anonymity: only the holder should know what issuer issued a specific credential.
- Anonymous credential: when a holder presents a credential, the verifier may not know who issued the certificate. He / She may only know that the credential was issued by some approved issuer.
- Holder untraceability: the holder identifiers and credentials can't be used to track holders through time.
- Confidentiality: no one but the holder and the issuer should know what the credential attributes are.
- Identifier linkability: no one should be able to link two identifier unless there is a proof presented by the holder.
- Credential linkability: No one should be able to link two credentials from the publicly available data. Mainly, no two issuers should be able to collude and link two credentials to one same holder by using the holder's digital identity.

In depth view. For the specific instantiation of the scheme, we examine the different ways that these requirements can be achieved and what are the trade-offs to be done (e.g.: using pairwise identifiers vs. one fixed public key; different revocation mechanisms; etc.) and elaborate on the privacy and efficiency properties of each.

Functionality / Problem	**Instantiation Method**	**Proof Details**	**Privacy / Robustness**	**Reference**
Holder identification: how to identify a holder of credentials	Single identifier in the federated realm: PRF based Public Key (idPK) derived from the physical ID of the entity and attested / onboarded by a federal authority	- The first credential an entity must get is the *onboarding credential* that attests to its identity on the system - Any proof of credential generated by the holder must include a verification that the idPK was issued an onboarding credential	- Physical identity is hidden yet connected to the public key. - Issuers can collude to link different credentials by the same holder. - An entity can have only one identity in the system	
	Single identifier in the self-sovereign realm: PRF based Public Key (idPK) self derived by the entity.	- Any proof of credential must show the holder knows the preimage of the idPK and that the credential was issued to the idPK in question	- Physical identity is hidden and does not necessarily have to be connected to the public key - Issuers can collude to link different credentials by the same holder - An entity can have several identities and conveniently forget any of them upon issuance of a “negative credential”	
	Multiple identifiers: Pairwise identification through identifiers. For each new interaction the holder generates a new identifier.	- Every time a holder needs to connect to a previous issuer, it must prove a connection of the new and old identifiers in ZK - Any proof of credential must show the holder knows the secret of the identifier that the credential was issued to.	- Physical identity is hidden and does not necessarily have to be connected to the public key - Issuers cannot collude to link the credentials by the same holder - An entity can have several identities and conveniently forget any of them upon issuance of a “negative credential”	

Issuer identification	Federated permissions: there is a list of approved issuers that can be updated by either a central authority or a set of nodes	- To accept a credential one must validate the signature against one from the list. To maintain the anonymity of the issuer, ring signatures can be used - For every proof of credential, a holder must prove that the signature in its credential is of an issuer in the approved list	- The verifier / public would not know who the issuer of the credential is but would know it is approved.	
	Free permissions: anyone can become an issuer, which use identifiers: - Public identifier: type 1 is the issuer whose signature verification key is publicly available - Pair-wise identifiers: type 2 is the issuer whose signature verification key can be identified only pair-wise with the holder / verifier	- The credentials issued by type 1 issuers can be used in proofs to unrelated parties - The credentials issued by type 2 issuers can only be used in proofs to parties who know the issuer in question.	- If ring signatures are used, the type one issuer identifiers would not imply that the identity of the issuer can be linked to a credential, it would only mean that “Key K_a belongs to company A” - Otherwise, only the type two issuers would be anonymous and unlinkable to credentials	
Credential Issuance	Blind signatures: the issuer signs on a commitment of a self-attested credential after seeing a proof of correct attestation; a second kind of proof would be	- The proof of correct attestation must contain the structure, data types, ranges and credential type that the issuer allows - In some cases, the proof must contain verification of the	- Issuer’s signatures on credentials add limited legitimacy: a holder could add specific values / attributes that are not real and the issuer would not know - An Issuer can collude with a holder to	

	needed in the system	attributes themselves (e.g.: address is in Florida, but not know the city) - The proof of credential must not be accepted if the signature of the credential was not verified either in zero-knowledge or as part of some public verification	produce blind signatures without the issuer being blamed	
	In the clear signatures: the issuer generates the attestation, signing the commitment and sending the credential in the clear to the holder	- The proof of credential must not be accepted if the signature of the credential was not verified either in zero-knowledge or as part of some public verification	- Issuer must be trusted, since she can see the Holder's data and could share it with others - The signature of the issuer can be trusted and blame could be allocated to the issuer	
Credential Revocation	Positive accumulator revocation: the issuer revokes the credential by removing an element from an accumulator	- The holder must prove set membership of a credential to prove it was issued and was not revoked at the same time - The issuer can revoke a credential by removing the element that represents it from the accumulator	- If the accumulator is maintained by a central authority, then only the authority can link the revocation to the original issuance, avoiding timing attacks by general parties (join-revoke linkability) - If the accumulator is maintained through a public state, then there can be linkability of revocation with issuance since one can track the added values and test its membership	[4]
	Negative accumulator revocation: the	- The holder must prove set membership of a credential to prove	- Even when the accumulator is maintained through a	

	issuer revokes by adding an element to an accumulator	it was issued - The issuer can revoke a credential by adding to the negative accumulator the revocation secret related to the credential to be revoked - The holder must prove set non-membership of a revocation secret associated to the credential in question - The verifier must use the most recent version of the accumulator to validate the claim	public state, the revocation cannot be linked to the issuance since the two events are independent of each other	

Gadgets

Each of the methods for instantiating the different functionalities use some of the following gadgets that have been described in the Gadgets section. There are three main parts to the predicate of any proof.

1. The first is proving the veracity of the identity, in this case the holder, for which the following gadgets can / should be used:
    - **Commitment** for checking that the identity has been attested to correctly.
    - **PRF** for proving the preimage of the identifier is known by the holder
    - **Equality of strings** to prove that the new identifier has a connection to the previous identifier used or to an approved identifier.\

2. Then there is the part of the constraint system that deals with the legitimacy of the credentials, the fact that it was correctly issued and was not revoked.
    - **Commitment** for checking that the credential was correctly committed to.
    - **PRF** for proving that the holder knows the credential information, which is the preimage of the commitment .
    - **Equality of strings** to prove that the credential was issued to an identifier connected to the current identifier.
    - **Accumulators (Set membership / non-membership)** to prove that the commitment to the credential exists in some set (usually an accumulator), implying that it was issued correctly and that it was not revoked.

3. Finally there is the logic needed to verify the rules / constraints imposed on the attributes themselves. This part can be seen as a general gadget called “credentials”, which allows to verify the specific attributes embedded in a credential. Depending on the credential type, it uses the following low level gadgets:
    - **Data Type** used to check that the data in the credential is of the correct type
    - **Range Proofs** used to check that the data in the credential is within some range
    - **Arithmetic Operations (field arithmetic, large integers, etc.)** used for verifying arithmetic operations were done correctly in the computation of the instance.
    - **Logical Operators (bigger than, equality, etc.)** used for comparing some value in the instance to the data in the credentials or some computation derived from it.

Security caveats

1. If the Issuer colludes with the Verifier, they could use the revocation mechanism to reveal information about the Holder if there is real-time sharing of revocation information.
2. Furthermore, if the commitments to credentials and the revocation information can be tracked publicly and the events are dependent of each other (e.g.: revocation by removing a commitment), then there can be linkability between issuance and revocation.
3. In the case of self-attestation or collusion between the issuer and the holder, there is a much lower assurance of data integrity. The inputs to the ZKP could be spoofed and then the proof would not be sound.
4. The use of Blockchains create a reliance on a trusted oracle for external state. On the other hand, the privacy guaranteed at blockchain-content level is orthogonal to network-level traffic analysis.

A use-case example of credential aggregation

Use-case description

As a way to illustrate the above protocol, we present a specific use-case and explicitly write the predicate of the proof. Mainly, there is an identity, Alice, who wants to prove to some company, Bob Inc. that she is an accredited investor, under the SEC rules, in order to acquire some company shares. Alice is the prover; the IRS, the AML entity and The Bank are all issuers; and Bob Inc. is the verifier.

The different processes in the adaptation of the use-case are the following:

1. Three confidential credentials are issued to Alice which represent the rules that we apply on an entity to be an accredited investor[1]:

[1] We assume that the SEC generates the constraint system for the accreditation rules as the circuit used to generate the proving and verification keys. In the real scenario, here are the Federal Rules for accreditation.

    a. The IRS issues a tax credential, C_0, that testifies to the claim “from 1/1/2017 until 1/1/2018, Alice, with identifier X_0, ows 0$ to the IRS, with identifier Y” and holds two attributes: the net income of Alice, *$income,* and a bit *b* such that *b=1* if Alice has paid her taxes.
    b. The AML entity issues a KYC credential, C_1, that testifies to claim T_1:= “Alice, with identifier X_1, has NO relation to a (set of) blacklisted organization(s)”
    c. The Bank issues a net-worth credential, C_2, that testifies to claim T_2:= “Alice has a net worth of V_alice”
2. Alice then proves to Bob Inc. that:
    a. “Alice’s identifier, X_bob, is related to the identifiers, {X_i} for i = 0,1,2 that are connected to the confidential credentials {C_i}”
    b. “I know the credentials, which are the preimage of some commitment, {C_i}, were issued by the legitimate issuers”
    c. “The credentials, which are the preimage of some commitment, {C_i}, that exist in an accumulator, U, satisfy the three statements {T_i}”

Instantiation details

Based on the different options laid out in the table above, the following have been used:

- Holder identification: we instantiate the identifiers as a unique anonymous identifier, publicKey
- Issuance identification: the identity of the issuers is known to all the participants, who can publicly verify the signature on the credentials they issue[2].
- Credential issuance: credentials are issued by publishing a signed commitment to a positive accumulator and sharing the credential in the clear to Alice.
- Credential revocation: is done by removing the commitment of credential from a dynamic and positive accumulator. Alice must prove membership of commitment to show her credential was not revoked.
- Credential verification: Bob Inc. then verifies the cryptographic proof with the instance.

Note that the transfer of company shares as well as the issuance of company shares is outside of the scope of this use-case, but one could use the “Asset Transfer” section of this document to provide that functionality.

On another note, the fact that the proving and verification keys were validated by the SEC is an assurance to Bob Inc. that proof verification implies Alice is an accredited investor.

The Predicate

[2] With public signature verification keys that are hard coded into the circuit

Blue = publicly visible in protocol / statement
Red = secret witness, potentially shared between parties when proving

Definitions / Notation:

Public state: Accumulator, for issuance and revocation, which includes all the commitments to the credentials.

ConfCred = Commitment to Cred = { Revoke, certificateType, publicKey, Attribute(s) }

Where, again, the IRS, AML and Bank are authorities with well-known public keys. Alice's publicKey is her long term public key and one cannot create a new credential unless her long term ID has been endorsed. The goal of the scheme is for the holder to create a fresh **proof of confidential aggregated credentials to the claim of accredited investor**.

IRS issues a **$\text{ConfCred}_{\text{IRS}}$** = Commitment(open_{IRS}, revokeIRS, "IRS", myID, $Income, *b*), sigIRS
AML issues **$\text{ConfCred}_{\text{AML}}$**= Commitment(open_{AML}, revokeAML, "AML", myID, "OK"), sigAML

Holder generates a fresh public key freshCred to serve as an ephemeral blinded aggregate credential, and a ZKP of the following:

ZkPoK{ (witness: myID, $\text{ConfCred}_{\text{IRS}}$, $\text{ConfCred}_{\text{AML}}$, sigIRS, sigAML, $Income, , mySig, open_{IRS}, open_{AML} statement: freshCred, minIncomeAccredited) :

Predicate:
- $\text{ConfCred}_{\text{IRS}}$ is a commitment to the IRS credential (open_{IRS}, "IRS", myID, $Income)
- $\text{ConfCred}_{\text{AML}}$ is the AML crdential to ($\text{open}_{\text{AML,}}$ "AML", myID, "OK")
- $Income >= minIncomeAccredited
- *b* = 1 = "myID paid full taxes"
- mySig is a signature on freshCred for myID
- **ProveNonRevoke( )**

}

Present the credential to relying party: freshCred and ***zkp***.

ProveNonRevoke(rhIRS, w_hrIRS, rhAML, w_hrAML, a_IRS

- revokeIRS: revocation handler from IRS. Can be embedded as an attribute in $\text{ConfCredt}_{\text{IRS}}$ and is used to handle revocations.
- $\text{wit}_{\text{rhIRS}}$: accumulator witness of revokeIRS.
- revokeAML: revocation handler from AML. Can be embedded as an attribute in $\text{ConfCredt}_{\text{AML}}$ and is used to handle revocations.
- $\text{wit}_{\text{rhAML}}$: accumulator witness of revokeAML.
- acc_{IRS}: accumulator for IRS.

- $CommRevoke_{IRS}$: commitment to revokeIRS. The holder generates a new commitment for each revocation to avoid linkability of proofs.
- acc_{AML}: accumulator for AML.
- $CommRevoke_{AML}$: commitment to revokeAML. The holder generates a new commitment for each revocation to avoid linkability of proofs.

ZkPoK{ (witness: rhIRS, $open_{rhIRS}$, w_{rhIRS}, rhAML, $open_{rhAML}$, w_{rhAML}|| statements: C_{IRS}, a_{IRS} , C_{AML} , a_{AML}):

Predicate:

- C_{IRS} is valid commitment to ($open_{rhIRS}$, rhIRS)
- rhIRS is part of accumulator a_{IRS}, under witness w_{rhIRS}
- rhIRS is an attribute in $Cert_{IRS}$
- C_{AML} is valid commitment to ($open_{rhAML}$, rhAML)
- rhAML is part of accumulator a_{AML}, under witness w_{rhAML}
- rhAML is an attribute in $Cert_{AML}$

}

- myCred is unassociated with myID, with sigIRS, sigAML etc.
- Withstands partial compromise: even if IRS leaks myID and sigIRS, it cannot be used to reveal the sigAML or associated myID with myCred

Asset Transfer

Privacy-preserving asset transfers and balance updates

In this section, we examine two use-cases involving using ZK Proofs (ZKPs) to facilitate private asset-transfer for transferring fungible or non-fungible digital assets. These use-cases are motivated by privacy-preserving cryptocurrencies, where users must prove that a transaction is valid, without revealing the underlying details of the transaction. We explore two different frameworks, and outline the technical details and proof systems necessary for each.

There are two dominant paradigms for tracking fungible digital assets, tracking ownership of assets individually, and tracking account balances. The Bitcoin system introduced a form of asset-tracking known as the UTXO model, where *Unspent Transaction Outputs* correspond roughly to single-use “coins”. Ethereum, on the other hand, uses the balance model, and each account has an associated balance, and transferring funds corresponds to decrementing the sender’s balance, and incrementing the receiver’s balance accordingly.

These two different models have different privacy implications for users, and have different rules for ensuring that a transaction is valid. Thus the requirements and architecture for building ZK

proof systems to facilitate privacy-preserving transactions are slightly different for each model, and we explore each model separately below.

In its simplest form, the asset-tracking model can be used to track non-fungible assets. In this scenario, a transaction is simply a transfer of ownership of the asset, and a transaction is valid if: the sender is the current owner of the asset. In the balance model (for fungible assets), each account has a balance, and a transaction decrements the sender's account balance while simultaneously incrementing the receivers. In a "balance" model, a transaction is valid if 1) The amount the sender's balance is decremented is equal to the amount the receiver's balance is incremented, 2) The sender's balance remains non-negative 3) The transaction is signed using the sender's key.

Zero-Knowledge Proofs in the asset-tracking model

In this section, we describe a simple ZK proof system for privacy-preserving transactions in the asset-tracking (UTXO) model. The architecture we outline is essentially a simplification of the ZCash system. The primary simplification is that we assume that each asset ("coin") is indivisible. In other words, each asset has an owner, but there is no associated value, and a transaction is simply a transfer of ownership of the asset.

Motivation: Allow stakeholders to transfer non-fungible assets, without revealing the ownership of the assets publicly, while ensuring that assets are never created or destroyed.

Parties: There are three types of parties in this system: a Sender, a Receiver and a distributed set of validators. The sender generates a transactions and a proof of validity. The (distributed) validators act as verifiers and check the validity of the transaction. The receiver has no direct role, although the sender must include the receiver's public-key in the transaction.
What is being proved: At high level, the sender must prove three things to convince the validators that a transaction is valid.

- The asset (or "note") being transferred is owned by the sender. (Each asset is represented by a unique string)
- The sender proves that they have the private spending keys of the input notes, giving them the authority to send asset.
- The private spending keys of the input assets are cryptographically linked to a signature over the whole transaction, in such a way that the transaction cannot be modified by a party who did not know these private keys.

What information is needed by the verifier:

- The verifiers need access to the CRS used by the proof system
- The validators need access to the entire history of transactions (this includes all UTXOs, commitments and nullifiers as described later). This history can be stored on a distributed ledger (e.g. the Bitcoin blockchain)

Possible attacks:

- CRS compromise: If an attacker learns the private randomness used to generate the CRS, the attacker can forge proofs in the underlying system
- Ledger attacks: validating a transaction requires reading the entire history of transactions, and thus a verifier with an incorrect view of the transaction history may be convinced to accept an incorrect transaction as valid.
- Re-identification attacks: The purpose of incorporating ZKPs into this system is to facilitate transactions without revealing the identities of the sender and receiver. If anonymity is not required, ZKPs can be avoided altogether, as in Bitcoin. Although this system hides the sender and receiver of each transaction, the fact that a transaction occurred (and the time of its occurrence) is publicly recorded, and thus may be used to re-identify individual users.
- IP-level attacks: by monitoring network traffic, an attacker could link transactions to specific senders or receivers (each transaction requires communication between the sender and receiver) or link public-keys (pseudonyms) to real-world identities
- Man-it-the-Middle attacks: An attacker could convince a sender to transfer an asset to an "incorrect" public-key

Setup scenario: This system is essentially a simplified version of Zcash proof system, modified for indivisible assets. Each asset is represented by a unique AssetID, and for simplicity we assume that the entire set of assets has been distributed, and no assets are ever created or destroyed.

At any given time, the public state of the system consists of a collection of "asset notes". These notes are stored as leaves in a Merkle Tree, and each leaf represents a single indivisible asset represented by unique assetID. In more detail, a "note" is a commitment to { Nullifier, publicKey, assetID }, indicating that publicKey "owns" assetID.

Main transaction type: Sending an asset from Current Owner A to New Owner B

Security goals:

- Only the current owner can transfer the asset
- Assets are never created or destroyed

Privacy goals*: Ideally, the system should hide all information about the ownership and transaction patterns of the users. The system sketched below does not attain that such a high-level of privacy, but instead achieves the following privacy-preserving features*

- Transactions are publicly visible, i.e., anyone can see that a transaction occurred
- Transactions do not reveal which asset is being transferred
- Transactions do not reveal the identities (public-keys) of the sender or receiver.
    - Limitation: Previous owner can tell when the asset is transferred. (Mitigation: after receiving asset, send it to yourself)

Details of a transfer:

Each transaction is intended to transfer ownership of an asset from a Current Owner to a New Owner. In this section, we outline the proofs used to ensure the validity of a transaction. Throughout this description, we use Blue to denote information that is globally and publicly visible in the protocol / statement. We use Red to denote private information, e.g. a secret witness held by the prover or information shared between the Current Owner and New Owner.

The Current Owner, A, has the following information

- A publicKey and corresponding secretKey
- An assetID corresponding to the asset being transferred
- A note in the MerkleTree corresponding to the asset
- Knows how to open the commitment (Nullifier, assetID, publicKey) publicKeyOut of the new Owner B

The Current Owner, A, generates

- A new NullifierOut
- A new commitment commitment (NullifierOut, assetID, publicKey)

The Current owner, A, sends

- Privately to B: NullifierOut, publicKeyOut, assetID
- Publicly to the blockchain: Nullifier, comOut, ZKProof (the structure of ZKProof is outlined below)

If Nullifier does not exist in MerkleTree and and ZKProof validates, then comOut is added to the merkleTree.

The structure of the Zero-Knowledge Proof:

We use a modification of Camenisch-Stadler notation to describe the describe the structure of the proof.

Public state: MerkleTree of Notes:
Note = Commitment to { Nullifier, publicKey, assetID }

ZKProof = $ZkPoK_{pp}${
(witness: publicKey, publicKeyOut, merkleProof, NullifierOut, com, assetID, sig
statement: MerkleTree, Nullifier, comOut) :
predicate:
- com is included in MerkleTree (using merkleProof)
- com is a commitment to (Nullifier, publicKey, assetID)
- comOut is a commitment to (NullifierOut, publicKeyOut, assetID)
- sig is a signature on comOut for publicKey
}

Zero-Knowledge proofs in the balance model

In this section, we outline a simple system for privately transferring fungible assets, in the "balance model." This system is essentially a simplified version of zkLedger. The state of the system is an (encrypted) account balance for each user. Each account balance is encrypted using an additively homomorphic cryptosystem, under the account-holder's key. A transaction decrements the sender's account balance, while incrementing the receiver's account by a corresponding amount. If the number of users is fixed, and known in advance, then a transaction can hide all information about the sender and receiver by simultaneously updating all account balances. This provides a high-degree of privacy, and is the approach taken by zkLedger. If the set of users is extremely large, dynamically changing, or unknown to the sender, the sender must choose an "anonymity set" and the transaction will reveal that it involved members of the anonymity set, but not the amount of the transaction or which members of the set were involved. For simplicity of presentation, we assume a model like zkLedger's where the set of parties in the system is fixed, and known in advance, but this assumption does not affect the details of the zero-knowledge proofs involved.

Motivation: Each entity maintains a private account balance, and a transaction decrements the sender's balance and increments the receiver's balance by a corresponding amount. We assume that every transaction updates every account balance, thus all information the origin, destination and value of a transaction will be completely hidden. The only information revealed by the protocol is the fact that a transaction occurred.

Parties:

- A set of n stakeholders who wish to transfer fungible assets anonymously
- The stakeholder who initiates the transaction is called the "prover" or the "sender"
- The receiver, or receivers do not have a distinguished role in a transaction
- A set of validators who maintain the (public) state of the system (e.g. using a blockchain or other DLT).

What is being proved: The sender must convince the validators that a proposed transaction is "valid" and the state of the system should be updated to reflect the new transaction. A transaction consists of a set of n ciphertexts, (c_1,...,c_n), and where c_i = Enc_{pk}(x_i), and a transaction is valid if:

- The sum of all committed values is 0 (i.e., x_1 + … + x_n = 0)
- The sender owns the private key corresponding to all negative x_i
- After the update, all account balances remain positive

What information is needed by the verifier:

- The verifiers need access to the CRS used by the proof system
- The verifiers need access to the current state of the system (i.e., the current vector of n encrypted account balances). This state can be stored on a distributed ledger

Possible attacks:

- CRS compromise: If an attacker learns the private randomness used to generate the CRS, the attacker can forge proofs in the underlying system
- Ledger attacks: validating a transaction requires knowing the current state of the system (encrypted account balances), thus a validator with an incorrect view of the current state may be convinced to accept an incorrect transaction as valid.
- Re-identification attacks: The purpose of incorporating ZKPs into this system is to facilitate transactions without revealing the identities of the sender and receiver. If anonymity is not required, ZKPs can be avoided altogether, as in Bitcoin. Although this system hides the sender and receiver of each transaction, the fact that a transaction occurred (and the time of its occurrence) is publicly recorded, and thus may be used to re-identify individual users.
- IP-level attacks: by monitoring network traffic, an attacker could link transactions to specific senders or receivers (each transaction requires communication between the sender and the validators) or link public-keys (pseudonyms) to real-world identities
- Man-it-the-Middle attacks: An attacker could convince a sender to transfer an asset to an "incorrect" public-key. This is perhaps less of a concern in the situation where the user-base is static, and all public-keys are known in advance.

Setup scenario: There are fixed number of users, n. User i has a known public-key, pk_i Each user has an account balance, maintained as an additively homomorphic encryption of their current balance under their pk. Each transaction is a list of n encryptions, corresponding to the amount each balance should be incremented or decremented by the transaction. To ensure money is never created or destroyed, the plaintexts in an encrypted transaction must sum to 0. We assume that all account balance are initialized to non-negative values.

Main transaction type: Transferring funds from user i to user j

Security goals:

- An account balance can only be decremented by the owner of that account
- Account balances always remain non-negative
- The total amount of money in the system remains constant

Privacy goals*:* Ideally, the system should hide all information about the ownership and transaction patterns of the users. The system sketched below does not attain that such a high-level of privacy, but instead achieves the following privacy-preserving features

- Transactions are publicly visible, i.e., anyone can see that a transaction occurred
- Transactions do not reveal which asset is being transferred
- Transactions do not reveal the identities (public-keys) of the sender or receiver. Limitation: transaction times are leaked

Details of a transfer:

Each transaction is intended to update the current account balances in the system. In this section, we outline the proofs used to ensure the validity of a transaction. Throughout this description, we use Blue to denote information that is globally and publicly visible in the protocol / statement. We use Red to denote private information, e.g. a secret witness held by the prover.

The Sender, A, has the following information

- Public keys pk_1,...,pk_n
- secretKey_i corresponding to publicKey_i, and a values x_j, to transfer to user j
- The sender's own current account balance, y_i

The Sender, A, generates

- a vector of ciphertexts, C_1,...,C_n with C_t = Enc_{pk_t}(x_t)

The Sender, A, sends

- The vector of ciphertexts C_1,...,C_n and ZKProof (described below) to the blockchain

ZK Circuit:

Public state: The current state of the system, i.e., a vector of (encrypted) account balances, B_1,...,B_n.

ZKProof = ZkPoK_{pp}\{ (witness: i, x_1,...,x_n, sk statement: C_1,...,C_n) :
predicate:
-C_t is an encryption to x_t under public key pk_t for t=1...n
-x_1 + … + x_n = 0
-x_t >= 0 OR sk corresponds to pk_t for t = 1...n
-x_t >= 0 OR current balance B_t encrypts a value no smaller than |x_t| for t = 1...n
}

Regulation Compliance

Overview

An important pattern of applications in which zero-knowledge protocols are useful is within settings in which a regulator wishes to monitor, or assess the risk related to some item managed by a regulated party. One such example can be whether or not taxes are being paid correctly by an account holder, or is a bank or some other financial entity solvent, or even stable.

The regulator in such cases is interested in learning "the bottom line", which is typically derived from some aggregate measure on more detailed underlying data, but does not necessarily need to know all the details. For example, the answer to the question of "did the bank take on too many loans?" Is eventually answered by a single bit (Yes/No) and can be answered without

detailing every single loan provided by the bank and revealing recipients, their income, and other related data.

Additional examples of such scenarios include:

- Checking that taxes have been properly paid by some company or person.
- Checking that a given loan is not too risky.
- Checking that data is retained by some record keeper (without revealing or transmitting the data)
- Checking that an airplane has been properly maintained and is fit to fly

The use of Zero knowledge proofs can then allow the generation of a proof that demonstrate the correctness of the aggregate result. The idea is to show something like the following statement: There is a commitment (possibly on a blockchain) to records that show that the result is correct.

Trusting data fed into the computation:
In order for a computation on hidden data to prove valuable, the data that is fed in must be grounded as well. Otherwise, proving the correctness of the computation would be meaningless. To make this point concrete: A credit score that was computed from some hidden data can be correctly computed from some financial records, but when these records are not exposed to the recipient of the proof, how can the recipient trust that they are not fabricated?

Data that is used for proofs should then generally be committed to by parties that are separate from the prover, and that are not likely to be colluding with the prover. To continue our example from before: an individual can prove that she has a high credit score based on data commitments that were produced by her previous lenders (one might wonder if we can indeed trust previous lenders to accurately report in this manner, but this is in fact an assumption implicitly made in traditional credit scoring as well).

The need to accumulate commitments regarding the operation and management of the processes that are later audited using zero-knowledge often fits well together with blockchain systems, in which commitments can be placed in an irreversible manner. Since commitments are hiding, such publicly shared data does not breach privacy, but can be used to anchor trust in the veracity of the data.

An example in depth: Proof of compliance for aircraft

An operator is flying an aircraft, and holds a log of maintenance operations on the aircraft.
These records are on different parts that might be produced by different companies.
Maintenance and flight records are attested to by engineers at various locations around the world (who we assume do not collude with the operator).
The regulator wants to know that the aircraft is allowed to fly according to a certain set of rules. (Think of the Volkswagen emissions cheating story.)

The problem: Today, the regulator looks at th5e records (or has an auditor do so) only once in a while. We would like to move to a system where compliance is enforced in “real time”, however, this reveals the real-time operation of the aircraft if done naively.

Why is zero-knowledge needed? We would like to prove that regulation is upheld, without revealing the underlying operational data of the aircraft which is sensitive business operations. Regulators themselves prefer not to hold the data (liability and risk from loss of records), prefer to have companies self-regulate to the extent possible.

What is the threat model beyond the engineers/operator not colluding? What about the parts manufacturers? Regulators? Is there an antagonistic relationship between the parts manufacturers?

This scheme will work on regulation that isn't vague, such as aviation regulation. In some cases, the rules are vague on purpose and leave room for interpretation.

Protocol high level

Parties:

- **Operator / Party under regulation:** performs operations that need to comply to a regulation. For example an airline operator that operates aircrafts
- **Risk bearer / Regulator :** verifies that all regulated parties conform to the rules; updates the rules when risks evolve. For example, the FAA regulates and enforces that all aircrafts to be airworthy at all times. For an aircraft owner leasing their assets, they want to know that operation and maintenance does not degrade their asset. Same for a bank that financed an aircraft, where the aircraft is the collateral for the financing.
- **Issuer / 3rd party attesting to data:** Technicians having examined parts, flight controllers attesting to plane arriving at various locations, embarked equipment providing signed readings of sensors.

What is being proved:

- The operator proves to the regulator that the latest maintenance data indicates the aircraft is airworthy
- The operator proves to the bank that the aircraft maintenance status means it is worth a given value, according to a formula provided by that bank

What are the privacy requirements?

- An operator does not want to reveal the details of his operations and assets maintenance status to competition
- The aircraft identity must be kept anonymous from all parties except the regulators and the technicians.

- The technician's identity must be kept anonymous from the regulator but if needed the operator can be asked to open the commitments for the regulator to validate the reports

The proof predicate: "The operator is the owner of the aircraft, and knows some signed data attesting to the compliance with regulation rules: all the components are safe to fly".

- The plane is made up of the components x_1,...,x_n and for each of the components:
  - There is an *legitimate* attestation by an engineer who checked the component, and signed it's OK
  - The latest attestation by a technician is *recent*: the timestamp of the check was done before date D

What is the public / private data:

- **Private:**
  - Identity of the operator
  - Airplane record
  - Examination report of the technicians
  - Identity of the technician who signed the report
- **Public:**
  - Commitment to airplane record
  -

There is a record for the airplane that is committed to a public ledger, which includes miles flown.
There are records that attest to repairs / inspections by mechanics that are also committed to the ledger. The decommitment is communicated to the operator. These records reference the identifier of the plane.

Whenever the plane flies, the old plane record needs to be invalidated, and a new on committed with extra mileage.
When a proof of "airworthiness" is required, the operator proves that for each part, the mileage is below what requires replacement, or that an engineer replaced the part (pointing to a record committed by a technician).

At the gadget level:

- The prover proves knowledge of a de-commitment of an airplane record (decommitment)
- The record is in the set of records on the blockchain (set membership)
- and knowledge of de-commitments for records for the parts (decommitment) that are also in the set of commitments on the ledger (set membership)
- The airplane record is not revoked (i.e., it is the most recent one), (requires set non-membership for the set of published nullifiers)
- The id of the plane noted in the parts is the same as the id of the plane in the plane record. (equality)

- The mileage of the plane is lower than the mileage needed to replace each part (range proofs) OTHERWISE
- There exists a record (set membership)that says that the part was replaced by a technician (validate signature of the technician (maybe use ring signature outside of ZK?))

Conclusions

- The asset transfer and regulation can be used in the identity framework in a way that the additions complete the framework.
- External oracles such as blockchain used for storing reference to data commitments

References

[FHEStandards] - http://homomorphicencryption.org/white_papers/applications_homomorphic_encryption_white_paper.pdf

ZERO CASH - http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf
Baby-zoe - https://github.com/zcash-hackworks/babyzoe
HAWK -
ZKledger - https://eprint.iacr.org/2018/241.pdf

Other identity references (ref [4] is mentioned in table on page 11):

[1] Sovrin™: A Protocol and Token for Self-Sovereign Identity and Decentralized Trust, https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf

[2] D2.2 - Architecture for Attribute-based Credential Technologies - Final Version, https://abc4trust.eu/download/Deliverable_D2.2.pdf

[3] Jan Camenisch, Manu Drijvers, Maria Dubovitskaya. Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain. *ACM Conference on Computer and Communications Security*, 2017.

[4] Foteini Baldimtsi and Jan Camenisch and Maria Dubovitskaya and Anna Lysyanskaya and Leonid Reyzin and Kai Samelin and Sophia Yakoubov. Accumulators with Applications to Anonymity-Preserving Revocation. *IEEE European Symposium on Security and Privacy, EuroS&P 2017*, IEEE.

[5] Camenisch, Jan; Kohlweiss, Markulf; Soriente, Claudio. Solving Revocation with Efficient Update of Anonymous Credentials. Security and Cryptography for Networks, 454--471, 2010.

External resources

- ZKProof repository: https://github.com/zkpstandard/
- ZKProof Security Track and ZKProof Implementation Track documents on https://zkproof.org/documents.html
- zkp.science - a curated and annotated list of references
- Zcon0 ZKProof Workshop breakout notes: https://zkproof.org/zcon0_notes.pdf

Acknowledges

The workshops underlying these proceedings were sponsored by QED-it, Zcash Foundation, CheckPoint Institute for Information Security, Accenture, Danhua Capital, R3, Stratumn, Thundertoken, UR Ventures and Vmware.

Change Log

2018-08-01: Initial version. Summarizes the deliberations at 1st ZKProof Standards Workshop, and subsequent major contributions.