Legally binding anonymous multiparty commitments on a blockchain

Debasish Ray Chawdhuri
Research and Development
Talentica Software (I) Pvt. Ltd.
Pune, India

***Abstract*—In recent times, many blockchain solutions have been proposed for automatic electronic contracts on a blockchain. The most prominent example of it is Ethereum smart contracts. In many cases, it is necessary to be able to sign contracts involving things in the physical world that are not represented in the form of crypto-assets. Such a contract can be legally enforced after all the parties have signed them in case any party backs out after signing the contract. We propose a solution to advertise and sign anonymous multi-party contracts that are designed to be acceptable and enforceable in a court of law once they are signed.**

To enable this, we also provide a designated deanonymizer ring signature that allows some designated deanonymizers to be able to break the anonymity of the signer and also allows them to prove the true identity of the signer to any third party of their choice.

***Index Terms*—Anonymous contracts, Designated Deanonymizer Ring Signature, Blockchain**

I. Introduction

Since the popularity and momentum of Bitcoin [1] skyrocketed, there has been much interest in other applications of the underlying technology, now called a blockchain or more generally a distributed ledger. Among them, smart-contracts have come up as a groundbreaking idea. Instead of only validating the result of the transfer of coins, the validators are now responsible for validating the result of an arbitrary computation. The concept of a smart-contract has been a massive breakthrough since any agreements or arrangements can now be executed on a blockchain platform. One direct application was smart contracts to support custom tokens; for example, ERC20 tokens [2] on Ethereum blockchain [3].

Smart contracts directly allowed multiple parties to trade different assets on the blockchain automatically. However, the automatic execution of contracts can only be done if the assets being traded are somehow being represented on the blockchain with some form of a smart-contract. Either these assets are of purely cryptographic origin, for example, proofs of work or tokens in a different blockchain, which can be traded through a technique called atomic swap [4]; or these assets are some cryptographic representation of some physical assets, for example, fiat-backed cryptocurrencies. In the case of the later, there must be some third-party arbiter to make sure that the cryptographic representation correctly represents a physical asset. This dependency on a central party cannot be avoided since the existence and value of physical assets cannot be ascertained cryptographically.

However, it may be worthwhile to try to minimize the interaction with this arbiter and restrict our interaction to the only times when there is a dispute. In the real world, such a system exists in the form of the judiciary. The judiciary needs to intervene only in case of a dispute. There are also forums called alternative dispute resolution systems that can be used for this purpose. Some of these systems can use a decentralized mechanism for a conflict resolution with a majority vote among several judges. Such systems are introduced in section II.

The judiciary is dependent on the parties in conflict to be able to prove their claims using some form of evidence. In the case of trade, such evidence is usually some contract documents signed by all the parties involved. Contracts can be either signed physically or digitally. Once a contract is signed, it can be enforced by the legal system.

However, when working with an online trading platform, anonymity and privacy are of great importance. In the case of a centralized web-based system, anonymity is maintained by entrusting the central system not to disclose any private data. On a blockchain system, however, preferably, there are no privileged parties who can hide information from the system while still using that information to facilitate the trade.

In our proposed solution, we provide a way to sign legally enforceable contracts anonymously. Our solution ensures that the following properties hold -

- Any authorized party with a verified public key must be able to post an anonymous advertisement for the contract.
- Any party must be able to communicate with an advertiser anonymously.
- The system must ensure that all signatures are done only by validated users without being able to know the identities of the parties involved.
- The system must ensure that the parties involved in a contract know the identity of one another, while the contract remains anonymous to anyone else.
- In case any of the parties break the contract after it has been signed, any of the others involved must be able to prove the identity of all of the signers in the court of law or an alternative dispute resolution system.

978-1-7281-9390-8/20/$31.00 ©2020 IEEE
DOI 10.1109/CVCBT50464.2020.00009

- Since the public keys used for signing contracts must be mapped to some real parties in the physical world for the signatures to be legally enforceable, and we assume some authority for providing the mapping after doing validation and paperwork. This can be performed by a traditional certifying authority.

In the current system, we propose two different kinds of contracts - a seller-buyer contract and a peer-to-peer contract.

In a seller-buyer contract, a seller wants to get paid in cryptocurrency in exchange for some services or goods. Our proposed system allows the seller to post an advertisement anonymously and then sign a binding contract with any interested party. Signing the contract discloses the identity of the seller to the buyer while keeping the seller anonymous to the rest of the world.

In the case of a peer-to-peer system, one party advertises for a trade of two different physical assets. One example of such an asset transaction can be a promise to pay interest in exchange for a loan, or a promise to exchange the ownership of bonds of different companies. Our proposed system allows any verified user to create an advertisement that other verified users can respond to and strike a deal. When the signature is done, each party learns about the true identities of all the other parties, but everyone remains anonymous to non-participating parties.

It is well-known that in any multi-party communication protocol, the party that is supposed to send the last message can abandon the protocol while still having access to the data communicated by everyone else. In our system, such an abandonment would disclose the identities of the players who have already signed, to the party that abandons the trade. In such a case, our system proposes to resolve the problem of parties abandoning the deal after some other parties have already signed. It does so by locking cryptocurrency funds of each party beforehand until he/she signs the contract.

II. Related work

The concept of a ring signature was first suggested in [5]. A ring signature allows anonymity for the signer by allowing the signer to hide his/her public key within a set of other public keys of his/her choosing. A ring signature proves the fact that one of the given set of public keys belongs to the real signer, without giving out the identity of the signer. There is a fair amount of research on ring signatures with different properties and security definitions.

A verifiable ring signature [6]–[8] is a ring signature that allows the signer to prove he/she indeed is the real signer to a verifier if he/she wants to. An accountable ring signature introduced in [9] allows a designated set of deanonymizers to find out the real signer in the ring. However, it depends on the user being in possession of a certified smart card that holds the private key, which the user cannot have access to. It also does not provide any way for the deanonymizer to prove the identity of the real signer to anyone. [10] describes an accountable signature scheme which provides the same functionality. The signatured described in [10] has better assymptotic complexity, but ours perform better for smaller rings. Also, in case there is a need for the use of multiple deanonymisers, our construction provides better performance for that.

There has been some work in the area of having legal contracts on a blockchain. The Decentralized Arbitration and Mediation Network [11] is a project to provide a framework for arbitration using smart contracts. However, this is still a work-in-progress.

A significant amount of work on alternative dispute resolution on smart contracts has been done by the Juris team [12]. Juris defines a structure for legal contracts on a blockchain, and an elaborate process for dispute resolution. Juris allows the creation of smart contracts that fix the terms of a dispute resolution. Once a party asks for arbitration, that party can trigger the arbitration process. This immediately stops all further execution of the contract and freezes all funds in a special locked account. The Juris dispute resolution can happen in the following three manner -

1) **Self mediation:** This allows parties to resolve disputes using the contract to either proceed in the previously agreed-upon terms with mutual consent among all the involved parties or proceed with different terms agreed upon by the involved parties.
2) **SNAP Judgement:** If self mediation does not work, any party can escalate it to the Simple Neutral Arbitrator Poll or SNAP judgment. This process costs some inbuilt cryptocurrency called JRS. Some JRS is attached to a contract for this purpose when it is created. Some more can be attached by the parties. The JRS is paid to the Jurists who would be responsible for deciding the judgment. The jurists vote in an anonymous, decentralized poll. All the parties can see the result of the poll, i.e., the number of votes in favor of each possible outcome. They may choose to agree to a resolution suggested by the jurists.
3) **Binding PANEL Judgment:** If none of the above mechanisms work, the case would be escalated to the Peremptory Agreement for Neutral Expert Litigation or PANEL. This is a mechanism similar to the SNAP judgment in that it is made by a panel of Jurists. However, this judgment is binding on the parties, and the jurists are not anonymous. A presiding High Jurist would then be able to execute the decision and mark the dispute as closed.

The Juris system also has a mechanism to keep track of the reputation of every Jurist along with their area of expertise. The reputation is computed based on data both on blockchain and outside of the blockchain, like certification.

Kleros [13] defines a decentralized dispute resolution framework that is incentive-driven and works for cases where all the evidence can be evaluated electronically. Kleros allows smart contracts to make themselves arbitrable. Arbitrable contracts have the option to choose the courts under whose jurisdiction they would operate. Both the courts and the jurors are categorized per specialization in a tree structure. Jurors are compensated using cryptocurrency that the parties need to pay.

One difference between Kleros and Juris is that Kleros allows an unlimited number of appeals with the condition that the number of jurors is doubled in each appeal. This increases the cost proportionally. However, the option of multiple appeals curbs the incentive to bribe the jurors. Our work is, in some sense, complementary to Juris and Kleros.

There has been very little research on privacy-preserving contracts on a blockchain. A system called Hawk [14], privacy is achieved by having a trusted party computing and certifying the result of the computation dependent on private data. Such a system creates a dependency on the honesty of the central party. Lelantos [15] is a system that uses blockchain and onion routing to deliver packages anonymously.

III. Our contribution

We make the following contributions -

- We propose an optimization of the zero-knowledge proof of linear member tuple [16].
- We propose a different construction for a designated deanonymizer ring signature that allows the anonymity of the signer while allowing the deanonymizers of the signature to be able to know the true identity of the signer. The deanonymizers are also able to demonstrate the true identity of the signer to anyone.
- We propose a blockchain smart contract-based system to allow the signing of a contract anonymously by multiple parties. Such contracts can then be enforced by systems similar to [12] and [13] or a court in case of a dispute.

IV. Notations

We use the following notations and conventions.

- We use $i \in \{1..m\}$ as the index of a component of a tuple, and $j \in \{1..n\}$ to index a tuple in a set. In the case of the designated deanonymizer ring signature, i is used as the index of a deanonymizer, and j is used as the index of a signer in the ring.
- The counting variables are written as the superscript, and the bound ones are written as a subscript to avoid confusion. If j is written as a superscript, that means its a sequence with an element for each value for j. For example, $X^j = a^j Y^j$ means for each j, $X_j = a_j Y_j$ and $X = \{a^j Y^j\}$, means $X = \{a_1 Y_1, a_2 Y_2, ...\}$ for all values of j. If we use parentheses instead of curly braces, it means an ordered set. So, $X = (a^j Y^j)$, means $X = (a_1 Y_1, a_2 Y_2, ...)$. On the other hand, in the case of $X = sA_j$, the j refers to either a fixed value or a value known from the context. For example, $\forall j, A_j = s_j B_j$ is the same as $A^j = s^j B^j$. Some conditional expressions are also used. The expression $A^{j \neq k} = s^j B^j$ represents $\forall j \in \{1..n\} \setminus \{k\}, A_j = s_j B_j$. Notice that the condition is used on only one of the occurances of j for brevity.
- $\mathbb{G}$ is a DDH elliptic curve group of prime order q. G is a system-wide fixed generator of $\mathbb{G}$. $\mathbb{F}_q$ is the field integer modulo q. $\mathbb{F}_q^*$ is $\mathbb{F}_q \setminus \{0\}$. H_q is a hash function with output in $\mathbb{F}_q^*$.
- If X is a set, then $x \leftarrow X$ denotes a random sample x being picked from a uniform distribution over X. If X is a probabilistic polynomial-time algorithm or PPTA, then $x \leftarrow X$ represents that x is the output of one run of the algorithm X.
- A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is negligible if $\forall \alpha \in \mathbb{N}[\exists x_0 \in \mathbb{N}[\forall x \in \mathbb{N}[x > x_0 \Rightarrow |f(x)| < \frac{1}{x^\alpha}]]]$. For our system, we use the security parameter λ which represents the number of bits required to represent the order q of the group $\mathbb{G}$. We say $f(\lambda) \approx \phi(\lambda)$ if the function $f(\lambda) - \phi(\lambda)$ is negligible in λ. We assume that q is the function of the parameter λ and $1/q(\lambda) \approx 0$.

V. Designated deanonymizer Ring Signature

We propose a designated deanonymizer ring signature to facilitate our trade of commitments. A designated ring signature allows a set of designated public keys such that the owners of the corresponding private keys can deanonymize the signer. In addition, a designated deanonymizer ring signature gives the deanonymizers the ability to prove the identity of the real signer to everyone.

A designated deanonymizer ring signature is a multi-party protocol - a signer, a set of deanonymizers, a verifier, and an authority.

- The signer must have permanent key-pair verified by the system.
- The deanonymizer key-pairs are ephemeral and can be used for exactly one signature. We support multiple deanonymizers in our signature.
- The signer is the party making a commitment to the deanonymizer.
- Since it is a ring signature, the signer hides his/her permanent public key in a random collection of other decoy public keys.
- However, a designated deanonymizer can discover the true identity of the signer using his/her ephemeral public key.
- A deanonymizer can prove the identity of the true signer to an authority or a judge using an interactive protocol.
- A verifier does not need a key-pair, which means anyone can verify a signature.
- A verifier checks that the deanonymizer can know and prove the identity of the true signer to any authority during verification. An authority does not need a key-pair.

The following describes a designated deanonymizer ring signature formally.

- **Key Generation:** The j^{th} signer generates a permanent key-pair $(sk_{sj}, pk_{sj}) \leftarrow Gen()$. The i^{th} deanonymizer generate his/her own key-pair $(sk_{di}, pk_{di}) \leftarrow Gen()$.
- **Signing:** For a message M, a set of public keys of possible signers $\{pk_s^j\}$, k being the index of the public key of the current signer, secret key sk_s for the public key $\{pk_{sk}\}$, deanonymizer keys $\{pk_d^i\}$, the signer generates a signature $\Pi \leftarrow Sign_{sk_s}(M, (pk_s^j), (pk_d^i))$

- **Verification:** For a message M, a set of public keys of possible signers $\{pk_s^j\}$, deanonymizer keys $\{pk_d^i\}$, and the signature Π, the verifier checks the validity using $validity \leftarrow Verify(\Pi, M, (pk_s^j), (pk_d^i))$, where $validity \in \{0, 1\}$.
- **Deanonymization:** For a message M, a set of public keys of possible signers $\{pk_s^j\}$, k the index of the public key of the current signer, secret key sk_s for the public key pk_{sk} , deanonymizer keys $\{pk_d^i\}$,,the signature Π, and the private key sk_{dl} for the l^{th} deanonymizer, the deanonymzer can compute the public key of the true signer using $pk_{sk} = Deanonymize_{sk_{dl}}(\Pi, M, (pk_s^j))$.
- **Demonstration of the signer by the deanonymizer:** For a message M, a set of public keys of possible signers $\{pk_s^j\}$, k the index of the public key of the current signer, secret key sk_s for the public key pk_{sk}, deanonymizer keys $\{pk_d^i\}$, the signature Π, and the private key sk_{dl} for the l^{th} deanonymizer, the deanonymzer can execute an zero-knowledge interactive proof protocol to any authority to demonastrate the identity of the correct identity of a signer to any authority. The protocol between the prover P and the authority A would be represented as $(T, pk_{sk}, s) = Demonstrate(\mathcal{P}(sk_{dl}), \mathcal{V}, \Pi, M, (pk_s^j), (pk_d^i), pk_{dl})$, where T is the transcript of the proof pk_{sk} is the true signer, and s is the result of the verification. $s = 1$ if the verification succeeds and $s = 0$ otherwise.

The following are the properties of a designated deanonymizer ring signature -

- **Completeness:** The completeness property states that if the signature is correctly created by a owner of one of the private keys, the public keys of which are listed in the ring of signers, then the verification is guranteed to be successful.

$$Pr\left[\begin{array}{l|l} v = 1 & \begin{array}{l}(sk_d^i, pk_d^i) \leftarrow Gen();\\ (sk_s^j, pk_s^j) \leftarrow Gen();\\ k \leftarrow \{1..n\};\\ \Pi \leftarrow Sign_{sk_{sk}}(M, (pk_s^j), (pk_d^i));\\ v \leftarrow Verify(\Pi, M, (pk_s^j), (pk_d^i));\end{array}\end{array}\right] = 1$$

- **Existential unforgeability:** Existential unforgeability guarantees that an adversary cannot produce a signature without a private key when provided with a signing oracle, with the condition that it is not allowed to query for the input for which it outputs a signature. For any PPTA adversary $\mathcal{A}$, and an oracle $\mathcal{O}$ that returns a valid signature for any $M', (pk_s^j), (pk_d^i)$. Let θ be the set of all the queries made by $\mathcal{A}$ to $\mathcal{O}$.

$$Pr\left[\begin{array}{l|l} v = 1 \wedge \\ \tau \notin \theta & \begin{array}{l}(sk_s^j, pk_s^j) \leftarrow Gen();\\ (M, \Pi, (pk_d^i)) \leftarrow \mathcal{A}_\theta^{\mathcal{O}}((pk_s^j));\\ \tau := (M, (pk_s^j), (pk_d^i));\\ v \leftarrow Verify(\Pi, M, (pk_s^j), (pk_d^i));\end{array}\end{array}\right] \approx 0$$

- **Anonymity:** The anonymity property guarantees that a PPTA adversary cannot find out the real signer in the ring signature even when provided with a signing oracle.

$$Pr\left[b' = b \;\middle|\; \begin{array}{l}(sk_d^i, (pk_d^i)) \leftarrow Gen();\\ (sk_s^j, (pk_s^j)) \leftarrow Gen();\\ k_1 \leftarrow \{1..n\};\\ k_2 \leftarrow \{1..n\} \setminus \{k_1\};\\ b \leftarrow \{1, 2\}\\ k := k_b;\\ \Pi \leftarrow Sign_{sk_{sk}}(M, (pk_s^j), (pk_d^i));\\ b' \leftarrow \mathcal{A}^{\mathcal{O}}(\Pi, k_1, k_2, M, (pk_s^j),\\ (pk_d^i));\end{array}\right] \approx \frac{1}{2}$$

- **Nonrepudiation:** The nonrepudiation property states that an adversary cannot create a signature that is valid for either two different messages or different signer rings or deanonymizer rings or both. The adversary is allowed to create all the public keys of its choice, thus allowing it to know all the private keys. This property guarantees that a signer cannot later claim that the signature was for a different message. In the following, $i' \in \{1..m'\}$ and $j' \in \{1..n'\}$ where (m', n') are possibly different from (m, n).

$$Pr\left[\begin{array}{l|l} v = 1 \wedge \\ v' = 1 & \begin{array}{l}(M, M', \Pi, (pk_d^i), (pk'^{i'}_d),\\ (pk_s^j), (pk'^{j'}_s)) \leftarrow \mathcal{A};\\ v \leftarrow Verify(\Pi, M, (pk_s^j),\\ (pk_d^i));\\ v' \leftarrow Verify(\Pi, M',\\ (pk'^{j'}_s), (pk'^{i'}_d));\end{array}\end{array}\right] \approx 0$$

- **Designated deanonymization (sanity):** This property guarantees that it is not possible for a PPTA to create a signature such that it deanonymizes to a public key outside of the list of public keys in the ring of signers. The adversary is allowed to create all keys other than the key-pair of that particular deanonymizer.

$$Pr\left[\begin{array}{l|l} v = 1 \wedge \\ pk \notin \{pk_s^j\} \wedge \\ \tau \notin \theta & \begin{array}{l}l \leftarrow \{1..m\};\\ (sk_{dl}, pk_{dl}) \leftarrow Gen();\\ (M, \Pi, (pk_s^j), (pk_d^{i \neq l}))\\ \leftarrow \mathcal{A}_\theta^{\mathcal{O}}(sk_{dl});\\ \tau := (M, (pk_s^j), (pk_d^i))\\ v \leftarrow Verify\\ (\Pi, M, (pk_s^j), (pk_d^i));\\ pk \leftarrow Deanonymize_{sk_{dl}}\\ (\Pi, M, (pk_s^j));\end{array}\end{array}\right] \approx 0$$

- **Designated deanonymization (unforgeability):** The unforgeability property of the designated deanonymization guarantees that even when an adversary knows all the private keys of the signer ring except one, it cannot make it look to the deanonymizer like the one signer that it does

not have the private key of, actually was the real signer.

$$Pr\left[\begin{array}{l|l} & l \leftarrow \{1..m\}; \\ & k \leftarrow \{1..n\}; \\ & (sk_{dl}, pk_{dl}) \leftarrow Gen(); \\ & (sk_{sk}, pk_{sk}) \leftarrow Gen(); \\ v = 1 \wedge & (M, \Pi, (pk_s^{j \neq k}), (pk_d^{i \neq l})) \\ pk = pk_{sk} \wedge & \leftarrow \mathcal{A}_\theta^{\mathcal{O}}(sk_{dl}); \\ \tau \notin \theta & \tau := (M, (pk_s^j), (pk_d^i)) \\ & v \leftarrow Verify \\ & (\Pi, M, (pk_s^j), (pk_d^i)); \\ & pk \leftarrow Deanonymize_{sk_{dl}} \\ & (\Pi, M, (pk_s^j)); \end{array}\right] \approx 0$$

- **Completeness of the demonstration of the signer by the deanonymizer:** The completeness of the demonstration means that an honest deanonymizer can prove the result of the deanonymization to an honest verifier.

$$Pr\left[\begin{array}{l|l} & l \leftarrow \{1..m\}; \\ & (sk_{dl}, pk_{dl}) \leftarrow Gen(); \\ & (M, \Pi, (pk_s^j), (pk_d^{i \neq l})) \\ & \leftarrow \mathcal{A}_\theta^{\mathcal{O}}(sk_{dl}); \\ (v = 1 \wedge & \tau := (M, (pk_s^j), (pk_d^i)) \\ \tau \notin \theta) \Rightarrow & v \leftarrow Verify \\ (s = 1 \wedge & (\Pi, M, (pk_s^j), (pk_d^i)); \\ pk = pk_{sk}) & pk \leftarrow Deanonymize_{sk_{dl}} \\ & (\Pi, M, (pk_s^j)); \\ & (T, pk_{sk}, s) \leftarrow \\ & Demonstrate(\mathcal{P}(sk_{dl}), \mathcal{V}, \\ & \Pi, M, (pk_s^j), (pk_d^i), pk_{dl}); \end{array}\right] \approx 1$$

- **Soundness of the demonstration of the signer by the deanonymizer:** The soundness property of the demonstration protocol proves that it is not possible for a dishonest PPTA $\mathcal{A}$ to create a signature collaborating with a different PPTA $\mathcal{A}_2$ to demonstrate a different signer to the authority than what the $Deanonymize$ function would return for the same signature.

$$Pr\left[\begin{array}{l|l} & l \leftarrow \{1..m\}; \\ & (sk_{dl}, pk_{dl}) \leftarrow Gen(); \\ & (M, \Pi, (pk_s^j), (pk_d^{i \neq l}), x) \\ & \leftarrow \mathcal{A}_\theta^{\mathcal{O}}(sk_{dl}); \\ (v = 1 \wedge & \tau := (M, (pk_s^j), (pk_d^i)) \\ \tau \notin \theta) \Rightarrow & v \leftarrow Verify \\ (s = 1 \wedge & (\Pi, M, (pk_s^j), (pk_d^i)); \\ pk \neq pk_{sk}) & pk \leftarrow Deanonymize_{sk_{dl}} \\ & (\Pi, M, (pk_s^j)); \\ & (T, pk_{sk}, s) \leftarrow \\ & Demonstrate(\mathcal{A}_2(x, sk_{dl}), \\ & \mathcal{V}, \Pi, M, (pk_s^j), (pk_d^i), pk_{dl}); \end{array}\right] \approx 0$$

- **Zero-knowledge property of the demonstration of the signer by the deanonymizer:** The zero-knowledge property of the demonstration shows that that for any PPTA $\mathcal{V}^*$ acting as a verifier (possibly malicious), the transcript of the demonstration protocol does not provide any information to any PPTA adversary that is also given access to $\mathcal{V}^*$.

$$Pr\left[\begin{array}{l|l} & (sk_d^i, pk_d^i) \leftarrow Gen(); \\ & (sk_s^j, pk_s^j) \leftarrow Gen(); \\ & k \leftarrow \{1..n\}; \\ & \Pi \leftarrow Sign_{sk_{sk}}(M, (pk_s^j), (pk_d^i)); \\ & (T_1, pk_{sk}, s) \leftarrow \\ b = b' & Demonstrate(\mathcal{P}(sk_{dl}), \\ & \mathcal{V}^*, \Pi, M, (pk_s^j), (pk_d^i), pk_{dl}); \\ & T_2 = \mathcal{S}(\mathcal{V}^*, \\ & \Pi, M, (pk_s^j), (pk_d^i)); \\ & b \leftarrow \{1, 2\}; \\ & b' = \mathcal{A}(\mathcal{V}^*, \\ & \Pi, M, (pk_s^j), (pk_d^i), T_b); \end{array}\right] \approx \frac{1}{2}$$

VI. Blockchain model

We assume a blockchain with a UTXO model. The UTXOs in our system supports some level of scripting like Bitcoin [1]. We assume the existence of UTXOs with an internal state. The states can be set or altered through smart contract calls. We assume the functionality of being able to call methods on a UTXO that do not cause the use of the UTXO but instead, change its internal state based on the custom logic written in it. This is not very different from Ethereum smart contract, except that in the case of UTXO, either the entire amount is spent or nothing is spent.

To spend the UTXO, one must call the $spend(spender, signature, DDRSig)$ method with appropriate arguments. If the method returns $true$, the transaction is valid; otherwise, it is invalid. We also assume that calling methods on spent transaction outputs or non-existent transaction outputs have no effect.

We assume the existence of two system-defined methods to be usable in the conditional payment code. The method $verifySchnorr(signer, signature)$ verifies a Schorr signature [17] $signature$ of the transaction body against a public key $signer$. The method $verifyDDRSig(message, DDRSig, peerList)$ verifies a designated deanonymizer ring signature $DDRSig$ for the message $message$, and it also verifies that the owner of every public key in the list $peerList$ can both discover the permanent key of the actual signer and the fact that each of them can prove the identity of the signer to an authority. For this purpose, the function also checks that every signer public key used for the ring signature has been KYC validated.

The protocol requires some broadcast of messages. Since blockchain systems also depend on peer-to-peer broadcasting of transactions, the same mechanism can be used to broadcast encrypted messages that can be decrypted only by the intended parties. We use the Diffie-Hellman key exchange to establish symmetric key channels between parties over this broadcasting mechanism.

VII. Multi-party commitments

A multi-party commitment in our system is potentially a legally binding agreement among multiple participants. We

propose two different kinds of multi-party transactions -

- Seller-buyer commitment
- Peer-to-peer multiparty commitment

A. Seller-buyer commitment

A seller-buyer commitment is a more straightforward case where the seller advertises it's products. The seller wants to get paid in cryptocurrency in exchange for something written in a legal contract, which is not a smart contract, but preferably structured. Although the seller does advertise publicly, the seller does not want to disclose when a sale has happened. The following describes the protocol.

Smart Contract 1: Conditional payment for seller-buyer commitment

```
Globals: timestamp, creator, committer, contract
function init(cr, com,contr) :
    timestamp = currentTimestamp();
    creator = cr; committer = com; contract = contr;
end
function spend(spender,signature,DDRSig) :
    if spender == creator then
        if currentTimestamp() > timestamp+ΔT ∧
          verifySchnorr(creator, signature) then
            return true;
        else
            return false;
        end
    else if spender==committer then
        if verifySchnorr(committer, signature) then
            if verifyDDRSig(contract, DDRSig,
              {creator}) then
                return true;
            else
                return false;
            end
        else
            return false;
        end
end
```

1) The seller generates a random secret key $s \leftarrow \mathbb{F}_q^*$.
2) The seller computes an ephemeral public key $S = sG$.
3) The seller broadcasts an advertisement (M, S), where M is the body of the contract.
4) An interested buyer, generates an ephemeral key-pair $p \leftarrow \mathbb{F}_q^*, P = pG$ and broadcasts (M, S, P) as an interest to this offer.
5) At this point, a Diffie-Hellman key exchange has been achieved as both sides can compute $sP = pS$, establishing a private communication channel between the seller and the buyer, which they can use to communicate further. These channels can be used to negotiate over the contract and decide on a final contract M' which may or may not be the same as M.
6) When both sides are satisfied with a possibly modified contract M' and a price c, the buyer creates a conditional payment as shown in smart contract 1 using call $init(P, S, M')$.
7) The seller checks that the conditional payment is correctly created and then spends the UTXO immediately within ΔT time using a correct demonstrable signature. Notice that the seller must sign the UTXO using the ephemeral key S, but also must provide the $DDRSig$ using his/her permanent key Q hidden within a random list of other permanent public keys.
8) In case the seller does not spend the UTXO, the buyer can get his/her money back after the time limit has expired.

The above protocol makes sure of the following properties.

- If the buyer does not correctly lock his/her funds, the seller does not provide the commitment.
- If the seller does not claim the funds by disclosing the correct demonstrable signature, the buyer can reclaim the funds after ΔT time, thus canceling the trade.

B. Peer-to-peer multiparty commitment

In this case, the nature of the commitment is peer-to-peer. The participants can either negotiate offline or respond to an anonymous advertisement. In the following, we describe the case of an anonymous advertisement. The offline negotiation would only take a subset of the steps.

We would like to highlight the problem of information asymmetry in signing such peer-to-peer contracts. Even though the contract is not valid until everyone signs it, the act of signing discloses the identity of the signer to the rest of the participants. This allows other participants to abandon the contract once they come to know of the identity of the people who already signed. To resolve this problem, we propose the use of locked cryptocurrency funds with conditional payments. The person who refuses abandons the contract would lose all the locked funds. The way to unlock the funds is to disclose the appropriate demonstrable signature. The following describes the protocol.

1) The advertiser generates a random key $p_1 \leftarrow \mathbb{F}_q^*$.
2) The advertiser computes an ephemeral public key $P_1 = p_1 G$.
3) The advertiser broadcasts an advertisement (M, P_1), where M is the body of the contract.
4) i^{th} interested peer generates an ephemeral key-pair $p_i \leftarrow \mathbb{F}_q^*, P_i = p_i G$ and broadcasts (M, P_i) as an interest to this offer.
5) Once the interests are shared, peer-to-peer Diffie-Hellman channels open up using the key $p_i P_l = p_l P_i$ between the i^{th} and the l^{th} peer.
6) The peers then can negotiate a modified contract M' and also decide on the order of signing the contract. The peers also decide on the transaction amount to be locked for each participant. Let i be the index in the order of signing, and there is a total of m participants.

Smart Contract 2: Conditional payment for multi-party commitment

```
Globals: timestamp, creator, contract, prev, state,
 nextContract, peerList
function init(cr,contr,pr, next, pl) :
    timestamp=currentTimestamp();
    creator = cr; contract = contr; prev = pr; state =
     prelock; nextContract = next; peerList = pl;
end
function lock(signer, signature, DDRSig) :
    if signer == prev ∧ verifySchnorr(prev, signature)
     ∧ verifyDDRSig(contract++prev, DDRSig,
     peerList) then
        state = locked;
    end
end
function spend(spender,signature,DDRSig) :
    if spender==creator ∧ verifySchnorr(creator,
     signature) then
        if state == prelock then
            if currentTimestamp() > timestamp+ΔT
             then
                return true;
            else
                return false;
            end
        else if state==locked then
            if verifyDDRSig(contract++creator,
             DDRSig, peerList) then
                if nextContract≠NULL then
                    nextContract.lock(creator, signature,
                     DDRSig);
                end
                return true;
            else
                return false;
            end
        else
            return false;
        end
    else
        return false;
    end
end
```

7) The peers must lock their funds in a pre-lock position in the reverse order of signing with the $init$ function in smart contract 2. The last peer locks his/her fund by creation of an UTXO with the agreed-upon amount by calling $init(P_m, M', P_{m-1}, NULL, \{P^i\})$. The other peers except the first peer then lock their own funds using $init(P_i, M', P_{i-1}, next\_contract, \{P^i\})$ where $next\_contract$ is the pointer to the conditional UTXO created by the $(i+1)^{th}$ peer. A peer must verify the $(i+1)^{th}$ peer's contract to be following the protocol before locking his/her funds.
8) Now, the first peer verifies all the contracts are created correctly. Then he/she calls $contract_2.lock(P_1, Sig_1, DSig_1)$, where Sig_1 is his signature on the transaction, and $DSig_1$ is the appropriate designated deanonymizer ring signature. This locks the funds of P_2 forever until P_2 discloses his/her designated deanonymizer ring signature.
9) Every other peer with index i then unlocks and spends his/her own funds in sequence using the appropriate designated deanonymizer ring signature $DSig_i$ and signature Sig_i using $contract_i.spend(P_i, Sig_i, DSig_i)$. The designated deanonymizer ring signature needs to be done on the message $M' + +P_i$, where $++$ means the concatenation of the bits. Notice that other than the last signer, every signer locks the funds of the next signer permanently by using his/her funds.

The above procedure makes sure of the following properties -

- If any of the peers refuses to lock his/her funds in the appropriate amount, everyone can abandon the contract, and no identity is disclosed in the process. Every peer who locked his/her fund can recover the same using the $spend$ function after ΔT time.
- After the first peer discloses the designated deanonymizer ring signature revealing his/her identity to the other peers, any player that abandons the process would lose all his/her funds permanently while others would be able to recover their funds. The ones who did sign before that would get their funds immediately after signing, the ones who did not sign yet can recover their funds after ΔT time.

VIII. Zero-knowledge Proof of Linear Member Tuple

The Zero-knowledge Proof of Linear Member Tuple is a generalization of the Cryptonote signature. We use this many times in our protocol. All our constructions requie an elliptic curve for which the DDH problem is assumed to be hard. The following describes the protocol.

Definition 1. ***Linear Tuple:*** *We describe a linear t-tuple as an ordered set of t pairs of elliptic curve points $\mathbf{X} = ((X_1, Y_1), (X_2, Y_2), (X_3, Y_3), ..., (X_t, Y_t))$ such that $\exists w[\forall i \in \{1...t\}[Y_i = wX_i]]$. We use i as the index of the vector in a tuple of vectors.*

A. Zero-knowledge Proof of Linear Member Tuple

Given a set of t-tuples $\mathbf{S} = \{\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, ..., \mathbf{X}_k\}$ of pairs of curve points, it may be the case that one of the member t-tuples is a linear tuple. We assume this t-tuple is $\mathbf{X} = ((X_1, Y_1), (X_2, Y_2), (X_3, Y_3), ..., (X_t, Y_t))$ where $\exists w[\forall i \in \{1...t\}[Y_i = wX_i]]$. We assume that the index of $\mathbf{X}$ in $\mathbf{S}$ is k . We describe a protocol so that a party with access to the proportionality constant w can provide a non-interactive zero-knowledge proof that one of the member t-tuple in the given set of t-tuples is a linear tuple. A bit-string called a token or a message $m \in 1^*$ is included in the proof to be able to use it as a signature.

The proof involves probabilistic polynomial-time algorithms - a prover $\mathbf{P}$ and a verifier $\mathbf{V}$.

1) Construction of ZkPLMT:

- **Prover** $\mathbf{P}(m, \mathbf{S}, k, w)$ **:** The prover works in the following manner.
  - Choose $r \leftarrow \mathbb{F}_q^*$
  - $\forall j \neq k$, choose $c_j, d_j \leftarrow \mathbb{F}_q^*$
  - Compute $h_2 = 1$, $h_i = H_q(h_{i-1}, \mathbf{S}) \forall 3 \leq i \leq t$
  - Compute $A_j = X_{1j}$, $B_j = Y_{1j}$, $C_j = \sum_{i=2}^{t} h_i X_{ij}$, $D_j = \sum_{i=2}^{t} h_i Y_{ij}$
  - $\forall j \neq k$, set $E_j = c_j A_j + d_j B_j$, $F_j = c_j C_j + d_j D_j$
  - Set $E_k = rA_k, F_k = rC_k$
  - Compute $h = H_q(m, \mathbf{S}, (E^j, F^j))$
  - Set $d_k = h - \sum_{j \neq k} d_j$
  - Set $c_k = r - d_k w$
  - Output $\pi = ((c^j, d^j))$

  When a random index k is implied, we denote the call to the prover by $\mathbf{P}(m, \mathbf{S}, \mathbf{X}, w)$ where $\mathbf{X}$ is the linear tuple at the index k.
- **Verifier** $\mathbf{V}(\pi, m, \mathbf{S})$ **:** Given the token m and the proof π, the verifier does the following.
  - Compute $h_2 = 1$, $h_i = H_q(h_{i-1}, \mathbf{S}) \forall 3 \leq i \leq t$
  - Compute $A_j = X_{1j}$, $B_j = Y_{1j}$, $C_j = \sum_{i=2}^{t} h_i X_{ij}$, $D_j = \sum_{i=2}^{t} h_i Y_{ij}$
  - set $E^j = c^j A^j + d^j B^j$, $F^j = c^j C^j + d^j D^j$
  - Compute $h = H_q(m, \mathbf{S}, (E^j, F^j))$
  - Check $h \stackrel{?}{=} \sum_j d_j$. If the check works, return 1, else return 0

2) Properties of ZkPLMT: We now consider the properties of the ZkPLMT protocol. These properties are adapted from [18], [19]. All of these properties are proved under the DDH-hardness assumption. The proofs of the theorems are provided in the Appendix section.

Theorem 1. ***Completeness:*** *Given a set of t-tuples $\mathbf{S} = \{\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, ..., \mathbf{X}_n\}$ of pairs of curve points, containing a member t-tuple $\mathbf{X} = ((X_1, Y_1), (X_2, Y_2), (X_3, Y_3), ..., (X_t, Y_t))$ such that $\exists w[\forall i \in \{1...t\}[Y_i = wX_i]]$, if prover $\mathbf{P}$ generates π in the prescribed manner, then $\mathbf{V}(\pi, m, \mathbf{S}) = 1$, where $\mathbf{V}$ is the verifier working in the prescribed manner.*

Theorem 2. ***Soundness:*** *Given an oracle $\mathcal{O}_P$ that provides a valid proof π' given any message m' and any set of tuples $\mathbf{S}'$ containing a linear member tuple, for any probabilistic polynomial time algorithm $\mathbf{P}'$, $Pr[(\pi, m, \mathbf{S}') \leftarrow \mathbf{P}'^{\mathcal{O}_P}_{\theta}(); \mathbf{V}(\pi, m, \mathbf{S}') = 1]$ is negligible if $\mathbf{S}'$ contains no linear member tuple, where θ is the set of all queries to $\mathcal{O}_P$.*

Theorem 3. ***Proof of knowledge:*** *If there exists a PPTA $\mathbf{P}'$ such that given an oracle $\mathcal{O}_P$ that provides a valid proof π' given any message m' and any set of tuples $\mathbf{S}'$ containing a linear member tuple, and $Pr[\mathbf{V}(\pi, m, \mathbf{S}) = 1 \wedge (m, \mathbf{S}, \pi) \notin \theta | (m, \mathbf{S}, \pi) \leftarrow \mathbf{P}'^{\mathcal{O}_P}_{\theta}();] = \delta$, where θ is the set of all queries to $\mathcal{O}_P$, and P' calls the random oracle Q times; then there exists an extractor $\mathbf{E}$ such that $Pr[w = w' | w' \leftarrow \mathbf{E}(\mathbf{P}');] \geq \frac{1}{Q}\delta(\delta - 1/q)$ where $\mathbf{S}$ and w are as defined in theorem 1.*

Theorem 4. ***Mixing:*** *Under the DDH assumption, for any probabilistic polynomial time algorithm $\boldsymbol{A}$, $Pr[k = k' | \pi \leftarrow \mathbf{P}(m, \mathbf{S}, k, w); k' \leftarrow \mathbf{A}(m, \mathbf{S}, \pi)] - 1/||\mathbf{S}||$ is negligible where $\mathbf{S}$ and w are as defined in theorem 1 and k is the index of linear tuple in $\mathbf{S}$.*

Theorem 5. ***Non-interactive Zero-knowledge [20]:*** *We rewrite $\mathbf{P}(m, \mathbf{S}, k, w) = \mathbf{P}'^{\mathcal{O}_H}(m, \mathbf{S}, k, w)$, where $\mathcal{O}_H$ is the random oracle. There exists a simulator $\mathcal{S}$, such that for any $(m, \mathbf{S}, k, w)$, $Pr[b' = b | (\pi_1, \mathcal{O}_S) \leftarrow \mathcal{S}(m, \mathbf{S}); \pi_2 \leftarrow \mathbf{P}'^{\mathcal{O}_S}(m, \mathbf{S}, k, w); b \leftarrow \{1, 2\}; b' \leftarrow \mathcal{A}^{\mathcal{O}_S}(m, \mathbf{S}, \pi_b)] = \frac{1}{2}$.*

Theorem 6. ***Token Stickiness:*** *For any probabilistic polynomial time algorithm $\mathbf{P}'$, $Pr[(\pi, \mathbf{S}', \mathbf{S}'', m', m'') \leftarrow \mathbf{P}'; m' \neq m'' \wedge \mathbf{V}(\pi, m', \mathbf{S}') = 1 \wedge \mathbf{V}(\pi, m'', \mathbf{S}'') = 1]$ is negligible.*

Our version of the protocol is faster than the original version in [16] because it is faster to compute the sum of scalar products of curve points than to individually compute the scalar products and then to sum them as mentioned in [21].

IX. Construction of the Designated Deanonymizer Ring Signature

The signature has four roles involved - a signer, several deanonymizers, a verifier, and an authority.

A. Key Generation:

The keys used are all elliptic curve Schnorr signature keys. For a system-wide fixed generator G, the private key is a randomly generated scalar $x \leftarrow \mathbb{F}_q$ and the public key is $X = xG$.

B. Signing

The signer creates a signature to make a commitment towards the deanonymizer. For the message M, signer private key $\mathfrak{q}$, the signer public key $Q = \mathfrak{q}G$, the deanonymizer public keys (P^i), the signer creates the signature as follows.

- The signer chooses public keys $\{Q_1, Q_2, ..., Q_m\}$ from the system containing Q. We assume that the index of Q in this array is k, i.e. $Q = Q_k$.
- The signer computes $R^i = \mathfrak{q}P^i$.
- The signer computes the linear tuple $\mathbf{X} = ((G, Q), (P^i, R^i))$.
- The signer computes non-linear tuples $\mathbf{Y}^j = ((G, Q^j), (P^i, R^i))$ for all $j \neq k$. Assign $\mathbf{Y}_k = \mathbf{X}$.

- The signer computes the proof $\pi = \mathbf{P}(M, \mathbf{Y^j}, k, \mathfrak{q})$
- The signer shares the signature (π, M, R^i, P^i, Q^j)

C. Verification

The verifier verifies the signature in the following manner.

- The verifier computes $\mathbf{Y}^j = ((G, Q^j), (P^i, R^i))$
- The verifier returns the result of the verification $\mathbf{V}(\pi, M, (\mathbf{Y}^j))$

D. Deanonymization

The l^{th} deanonymizer can discover the true signer by simply computing $Q = p_l^{-1}(p_l Q) = p_l^{-1}(p_l \mathfrak{q} G) = p_l^{-1}(\mathfrak{q} P_l) = p_l^{-1} R_l$ where p_l is the private key of the l^{th} deanonymizer and $P = P_l = p_l G$ is his public key.

E. demonstration of the signer by the deanonymizer

A deanonymizer can prove the true identity to any authority (identity verifier) using generalized zero-knowledge proof of linear dependence described in [16], which is a generalization of Chaum's undeniable signature [22] to multiple linear tuples.

- The deanonymizer sends the signature (π, M, R^i, P^i, Q^j), his own public key P and Q to the authority. First, the authority checks that P is one of the deanonymizer public keys in the signature. The authority runs the verification algorithm on the signature to check if it is valid.
- The authority chooses a random scalars $t, u \leftarrow \mathbb{F}_q$,
- The authority computes $C = tG + uQ$ and shares it to the deanonymizer.
- The deanonymizer computes $D = pC$ and shares $h = H_q(D)$ to the authority.
- The authority shares t, u to the deanonymizer.
- The deanonymizer checks if $C \stackrel{?}{=} tG + uQ$. If the check succeeds, the verifier discloses D to the authority.
- The authority checks if $h \stackrel{?}{=} H_q(D)$ and $D \stackrel{?}{=} tP + uR$. If both the check succeeds, the authority accepts the proof, otherwise rejects it.

Theorem 7. *The designated deanonymizer ring signature is complete.*

Theorem 8. *The designated deanonymizer signature is existentially unforgeable.*

Theorem 9. *The designated deanonymizer signature is anonymous*

Theorem 10. *The designated deanonymizer ring signature has nonrepudiation property.*

Theorem 11. *The designated deanonymization algorithm of the signer in the designated deanonymizer ring signature both sane and unforgeable.*

Theorem 12. *The demonstration of the signer algorithm for the designated deanonymizer ring signature is complete.*

Theorem 13. *The demonstration of the signer algorithm for the designated deanonymizer ring signature is sound.*

Theorem 14. *The demonstration of the signer algorithm for the designated deanonymizer ring signature is a black box computational zero-knowledge.*

X. Comparison against earlier construction of designated deanonymizer ring signature

For a ring size of N, and only one deanonymizer, our signature consists of $2N$ field elements and 1 group element. For every additional deanonymizer, the signature requires one additional group element. In case of [10], the signature requires $log_2 N + 12$ group elements and $\frac{1}{2}(3log_2 N + 12)$ field elements. We can reasonably assume that the group elements have the same size as the field elements. In such a case, our construction has a size of $2N + 1$, and the $\frac{5}{2} log_2 N + 18$. In the case of smaller ring sizes up to 13, our construction has a smaller size. Larger ring size does increase the size of the transaction proportionally as all the public keys do need to be included in the transaction. This limits the ring size in practice. For example, Monero has a fixed ring size of 11. Under this condition, our construction is a little bit smaller in size. Our construction needs only one extra group element per additional deanonymizer.

The verification of our construction requires $4N$ group exponentiations and 2 additional exponentiations per additional deanonymizer. In the construction in [10], it takes $N + 2mn + 2m + 15$ group exponentiations where $N = n^m$. Assuming $n = 2$, our construction performs about better up to a ring size of 10. The proving requires the same number of groups exponentiation as verification in our construction. In case of [10], the proving time is $mN + 3mn + 2m + 12$. The proving time is always better in our construction.

Therefore, it would be advisable to use our construction for smaller ring sizes, and the construction in [10] for larger ring sizes.

XI. Summary

In this paper, we tackle the problem of trading on the blockchain that involves physical assets. The trade is simply a commitment to conduct the actual trade in due time. However, the commitment can be proven in the court of law if any of the parties back off. However, we also provide a solution to keep every trade anonymous for everyone else.

We provide a new construction of a designated deanonymizer ring signature to facilitate signing contracts in a way so that only the deanonymizer can know the signer's identity, and everyone can verify that the deanonymizer can indeed know the signer's identity and prove it to someone if the need be.

In most of the contracts, the parties involved would stick to their promise, and hence, there is no need to involve the court. In this case, every party remains anonymous to everyone not involved in the contracts. On the other hand, if any of the parties refuse to fulfill his/her promise, any other member can take him/her to court.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf."

[2] "Eip 20: Erc-20 token standard, https://eips.ethereum.org/eips/eip-20," 2015.

[3] D. Wood, "Ethereum: a secure decentralised generalised transaction ledger," 2014.

[4] M. Herlihy, "Atomic cross-chain swaps," *CoRR*, vol. abs/1801.09515, 2018. [Online]. Available: http://arxiv.org/abs/1801.09515

[5] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[6] J. Lv and X. Wang, "Verifiable ring signature," 2003.

[7] C.-H. Wang and C.-Y. Liu, "A new ring signature scheme with signer-admission property," *Inf. Sci.*, vol. 177, pp. 747–754, 2007.

[8] Z. Changlun, L. Yun, and H. Dequan, "A new verifiable ring signature scheme based on nyberg-rueppel scheme," in *2006 8th international Conference on Signal Processing*, vol. 4, 2006.

[9] S. Xu and M. Yung, "Accountable ring signatures: A smart card approach," in *CARDIS*, 2004.

[10] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit, "Short accountable ring signatures based on ddh," *IACR Cryptology ePrint Archive*, vol. 2015, p. 643, 2015.

[11] "Decentralized arbitration and mediation network (damn), https://github.com/thirdkey-solutions/damn/blob/master/proposal.asciidoc."

[12] "Juris, https://drive.google.com/file/d/1318klgeyl4g02vudl-c-bcnvpkujtnbf/view," 2018.

[13] W. G. Clement Lesaege and F. Ast.

[14] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 839–858, 2015.

[15] R. AlTawy, M. ElSheikh, A. M. Youssef, and G. Gong, "Lelantos: A blockchain-based anonymous physical delivery system," *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pp. 15–1509, 2017.

[16] D. Ray Chawdhuri, "Patient privacy and ownership of electronic health records on a blockchain," in *Blockchain – ICBC 2019*, J. Joshi, S. Nepal, Q. Zhang, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 95–111.

[17] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO*, 1989.

[18] E. Fujisaki and K. Suzuki, "Traceable ring signature."

[19] N. van Saberhagen, "Cryptonote v 2.0, https://cryptonote.org/whitepaper.pdf."

[20] U. Feige, D. Lapidot, and A. Shamir, "Multiple non-interactive zero knowledge proofs based on a single random string," *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pp. 308–317 vol.1, 1990.

[21] B. Bnz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 315–334.

[22] D. Chaum, *Zero-Knowledge Undeniable Signatures (extended abstract)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 458–464. [Online]. Available: https://doi.org/10.1007/3-540-46877-3_41

APPENDIX

Proof of theorem 1

Proof. Since $c_k = r - d_k w$ and $Y_k^i = wX_k^i$, we have $c_k A_k + d_k B_k = c_k A_k + d_k w A_k = (c_k + d_k w)A_k = (r - d_k w + d_k w)A_k = rA_k = E_k$.

Also, $D_k = \sum_{i=2}^{t} h_i Y_{ik} = \sum_{i=2}^{t} h_i (wX_{ik}) = w\sum_{i=2}^{t} h_i X_{ik} = wC_k$. So, we have, $c_k C_k + d_k D_k = c_k C_k + d_k w C_k = (c_k + d_k w)C_k = (r - d_k w + d_k w)C_k = rC_k = F_k$. Now, $\forall j \neq k, E_j = c_j A_j + d_j B_j$ and $F_j = c_j C_j + d_j D_j$, because that is how they are computed by the prover. Hence, the verifier would compute the same hash h and since $d_k = h - \sum_{j \neq k} d_j$, we have $h = d_k + \sum_{j \neq k} d_j = \sum_j d_j$. Hence, the verifier would return 1. □

Proof of theorem 2

Proof. We define the following interactive protocol for proving this fact.

- The prover sends $(\mathbf{S})$ to the verifier.
- The verifier chooses t random $h^i \leftarrow \mathbb{F}_q^*$ where $h_2 = 1$.
- The verifier sends (h^i) to the prover.
- The prover chooses $r \leftarrow \mathbb{F}_q^*$
- $\forall j \neq k$, the prover chooses $c_j, d_j \leftarrow \mathbb{F}_q^*$
- The prover computes
  $A_j = X_{1j}$,
  $B_j = Y_{1j}$,
  $C_j = \sum_{i=2}^{t} h_i X_{ij}$,
  $D_j = \sum_{i=2}^{t} h_i Y_{ij}$
- $\forall j \neq k$, the prover computes $E_j = c_j A_j + d_j B_j$,
  $F_j = c_j C_j + d_j D_j$
- The prover computes $E_k = rA_k, F_k = rC_k$
- The provers sends (E^j, F^j) to the verifier.
- The verifier chooses $h \leftarrow \mathbb{F}_q^*$ and sends h to the prover.
- The prover computes $d_k = h - \sum_{j \neq k} d_j$
- The prover computes $c_k = r - d_k w$
- The prover sends $\pi = ((c^j, d^j))$ to the verifier.
- The verifier computes
  $A_j = X_{1j}$,
  $B_j = Y_{1j}$,
  $C_j = \sum_{i=2}^{t} h_i X_{ij}$,
  $D_j = \sum_{i=2}^{t} h_i Y_{ij}$
- The verifier checks if $E^j = c^j A^j + d^j B^j$,
  $F^j = c^j C^j + d^j D^j$. If not, the verifier rejects the proof and exits.
- The verifier checks if $h \stackrel{?}{=} \sum_j d_j$. If not, the verifier rejects the proof, otherwise, the verifier accepts the proof.

We now prove the following lemma -

Lemma 1. *If* $\mathbf{S}$ *does not have any linear tuple, the probability of a successful verification is* $(1 - (1 - 1/q)^n) + \frac{(1-1/q)^n}{q} \simeq (n+1)/q$.

Proof. There can be two cases to consider.

1) There exists some k such that $\exists w[B_k = wA_k \wedge D_k = wC_k]$. Since S does not contain any linear tuple, there is some i such that $Y_{ik} \neq wX_{ik}$. Since the choices $h^i \forall i > 2$ are completely random, the probability of this case happening for any particular k is $1/q$ considering the tuple $((X_{k2}, Y_{k2}), (X_{k3}, Y_{k3}), ..., (X_{kt}, Y_{kt}))$ is not a linear tuple. Otherwise the probability is 0. Hence, the probability of this happening for at least one tuple in n tuples is at most $1 - (1 - 1/q)^n$.
2) Otherwise $\nexists w[B_k = wA_k \wedge D_k = wC_k]$. By the time the have of h is available, value of (E^j, F^j) are already committed, so in this case, the probability of successful verification is $1/q$.

Hence, the maximum probability of a successful verification is $(1-(1-1/q)^n)+\frac{(1-1/q)^n}{q}$. For a large q, this is almost equal to $(n+1)/q$.

□

Now we consider our main theorem. For simplicity, we can consider all the outputs h^i to be coming from a single random oracle call that returns a tuple of values in $(\mathbb{F}_q)^t$. Let us assume that, for an $\mathbf{S}$ containing no linear tuple, a PPTA $\mathcal{A}$ can produce a valid proof with a probability ϵ using Q_1 queries to the first oracle and Q_2 queries to the second oracle. We break the interactive version using $\mathcal{A}$.

We simulate the proof oracle $\mathcal{S}$ in the following manner -

- $\forall j$, choose $c_j, d_j \leftarrow \mathbb{F}_q^*$
- Compute $h_2 = 1$, $h_i = H_q(h_{i-1}, \mathbf{S}')\forall 3 \leq i \leq t$
- Compute $A'_j = X_{1j}$, $B'_j = Y_{1j}$, $C'_j = \sum_{i=2}^{t} h_i X_{ij}$, $D'_j = \sum_{i=2}^{t} h_i Y_{ij}$
- $\forall j$, set $E'_j = c_j A'_j + d_j B'_j$, $F'_j = c_j C'_j + d_j D'_j$
- Compute $h = \sum_j d_j$
- Output $\pi = ((c^j, d^j))$

Now, an inner random oracle simulator $\mathcal{O}_S$ is constructed in the following manner -

- If an input was seen before, return the same output as before.
- Else if the input is $(m', \mathbf{S}', (E'^j, F'^j))$ for any of the queries made in proof oracle, return $\sum_j d_j$.
- Else for any other input x, generate $h \leftarrow \mathbb{F}_q^*$ and return h.

We simulate the first hash function the following way -

- Select random numbers $a \leftarrow \{1...Q_1\}$.
- Else if the input is any $(m, \mathbf{S}, (E'^j, F'^j))$ used in the proof oracle , return $\sum_j d_j$ for the same.
- Else if the call is a^{th} call to the random oracle, send the input to the interactive verifier and receive (h^i) from the verifier and return it.
- Else generate $h^i \leftarrow \mathbb{F}_q^*$ and return (h^i).

Similarly, we simulate the second hash function the following way -

- Select random numbers $b \leftarrow \{1...Q_2\}$.
- If the current input has been seen before, return the stored output for the given input.
- Else if the call is b^{th} call to the random oracle, send the input to the interactive verifier and receive h from the verifier and return it.
- Else for input x, return $\mathcal{O}_S(x)$.

We run $\mathcal{A}$ with the simulated oracle. Now, there is a $1/Q_1$ chance that $\mathcal{A}$ uses a^{th} call to the oracle to use the h^i values and there is a $1/Q_2$ chance that $\mathcal{A}$ uses b^{th} call to the oracle to use the h value. If it did so, it has a probability of ϵ of successful break. Hence the total probability of a successful break of the interactive protocol is $\frac{\epsilon}{Q_1 Q_2}$. Hence, $\frac{\epsilon}{Q_1 Q_2} \leq (n+1)/q \Rightarrow \epsilon \leq \frac{(n+1)Q_1 Q_2}{q}$.

□

Proof of theorem 3

Proof. Suppose there is a $\mathbf{P}'$ that does produce some valid π for the commitment $(m, \mathbf{S}, (E^j, F^j))$ passed to the random oracle with a probability δ with the help of maximum Q oracle queries to the second oracle. We provide the definition for the extractor $\mathbf{E}$. $\mathbf{E}$ uses the algorithm P' $(\|\mathbf{S}\|+1)$ times with different simulators for the random oracle. $\mathbf{E}$ produces a random oracle simulator and the proof oracle simulator $\mathcal{O}_P$ in the following manner -

1) If a query to any of the oracles was seen before, it returns the same value as before.
2) If a query $(m', \mathbf{S}')$ is made to the oracle $\mathcal{O}_P$, the oracle $\mathcal{O}_P$ are simulated in the following manner -
   - $\forall j$, choose $c_j, d_j \leftarrow \mathbb{F}_q^*$
   - Compute $h_2 = 1$, $h_i = H_q(h_{i-1}, \mathbf{S}')\forall 3 \leq i \leq t$
   - Compute $A'_j = X_{1j}$, $B'_j = Y_{1j}$, $C'_j = \sum_{i=2}^{t} h_i X_{ij}$, $D'_j = \sum_{i=2}^{t} h_i Y_{ij}$
   - $\forall j$, set $E'_j = c_j A'_j + d_j B'_j$, $F'_j = c_j C'_j + d_j D'_j$
   - Compute $h = \sum_j d_j$
   - Output $\pi = ((c^j, d^j))$

   and the random oracle is simulated to return h when queried for $(m', \mathbf{S}', (E'^j, F'^j))$ for any query $(m', \mathbf{S}')$ made to $\mathcal{O}_P$. It is easy to check that this would make the proofs look valid under the simulated random oracle.
3) Otherwise, for any query x to the random oracle simulator, the oracle simulator returns a random value chosen uniformly from $\mathbb{F}_q^*$.

We rewind $\mathbf{P}'$ to right after the point it has chosen the hash argument $(m, \mathbf{S}, (E^j, F^j))$ repeated $\|\mathbf{S}\|$ times so that at least two of the runs are for the same linear tuple in the set $\mathbf{S}$. When we rewind the program $\mathbf{P}'$, we also rewind the oracle simulators upto the same point. We run the rest of $\mathbf{P}'$ with the oracle simulator so that the simulators now return different values. The probability of $\mathbf{P}'$ using the index of query in the second time is at least $1/Q$. Also, the minimum probability of passing the validation for two different challenges h and h' is at least $\delta(\delta - 1/q)$. Since the validation passes, it must be true that there is some index k so that the proofs π and π' for the challenges h and h' respectively are d_k and d'_k respectively such that $d_k \neq d'_k$. Also, by proof of lemma 1, we know that this k is the index of the linear member tuple. So, we must have $E_k = c_k A_k + d_k B_k = c'_k A_k + d'_k B_k \Rightarrow c_k + d_k w = c'_k + d'_k w$. This implies $w = \frac{c'_k - c_k}{d_k - d'_k}$ since $d_k \neq d'_k$. The computatation can be repeated for all possible value for k and the correct one can be verified by checking whether

$F_k = wE_k$. Hence the extractor can extract the knowledge of w with a probability of at least $\frac{1}{Q}\delta(\delta - 1/q)$.

□

Proof of theorem 4

Proof. Suppose there exists some $\mathbf{A}$ such that $|Pr[\pi \leftarrow \mathbf{P}(m, \mathbf{S}, k, w); \mathbf{A}(m, \mathbf{S}, \pi) = k] - 1/||\mathbf{S}||| = \epsilon$. We will use this to create a solver for the DDH problem. Given the DDH problem A, B, C, D, we design the following algorithm with a simulated random oracle.

- Generate $k \leftarrow \{1..t\}$.
- Choose $r_1, r_2, ..., r_t \leftarrow \mathbb{F}_q^*$. We compute $X^i = r^i A$ and $Y^i = r^i B$ for each $i \in \{1..t-1\}$. This forms $t-1$ elements of our linear t-tuple $\mathbf{X}$. The last element is (C, D).
- Choose random points for the other t-tuples to for the set $\mathbf{S}$. Keep the $\mathbf{X}$ at index k.
- Generate $m \leftarrow 1^*$.
- Compute $h_2 = 1$, $h_i = H_q(h_{i-1}, \mathbf{S}) \forall 3 \leq i \leq t$
- Compute $A_j = X_{1j}$, $B_j = Y_{1j}$, $C_j = \sum_{i=2}^{t} h_i X_{ij}$, $D_j = \sum_{i=2}^{t} h_i Y_{ij}$
- set $E^j = c^j A^j + d^j B^j$, $F^j = c^j C^j + d^j D^j$
- Design the simulated random oracle in the following way
  - If the input is $(m, S, (E^j, F^j))$, return $\sum_j d_j$.
  - Else return $H_q(input)$
- Compute $k' = \mathbf{A}(m, \mathbf{S}, \pi)$ by passing the simulated random oracle.
- If $k \stackrel{?}{=} k'$ return 1, else 0.

Now, if D indeed is a DDH point, i.e. $\exists(b, c)[B = bA \wedge C = cA \wedge D = bcA]$, the signature really is a valid signature w.r.t. the simulated oracle and there really is a linear member tuple, so $Pr[k' = k] = 1/||S|| \pm \epsilon$. On the other hand, when D is not really a DDH point, $Pr[k' = k] = 1/||S||$. Hence, the advantage of our algorithm $= |Pr[k' = k|\exists(b, c)[B = bA \wedge C = cA \wedge D = bcA]] - Pr[k' = k|\neg\exists(b, c)[B = bA \wedge C = cA \wedge D = bcA]] = 1/||S|| \pm \epsilon - 1/||S|| = \pm\epsilon$. So, the advantage for our DDH solver is also ϵ. □

Proof of theorem 5

Proof. We create the following simulator $\mathcal{S}$.

- $\forall j$, choose $c_j, d_j \leftarrow \mathbb{F}_q^*$
- Compute $h_2 = 1$, $h_i = H_q(h_{i-1}, \mathbf{S}) \forall 3 \leq i \leq t$
- Compute $A'_j = X_{1j}$, $B'_j = Y_{1j}$, $C'_j = \sum_{i=2}^{t} h_i X_{ij}$, $D'_j = \sum_{i=2}^{t} h_i Y_{ij}$
- $\forall j$, set $E'_j = c_j A'_j + d_j B'_j$, $F'_j = c_j C'_j + d_j D'_j$
- Compute $h = \sum_j d_j$
- Output $\pi = ((c^j, d^j))$

Now, the random oracle simulator $\mathcal{O}_S$ is constructed in the following manner -

- If an input was seen before, return the same output as before.
- Else if the input is $(m, \mathbf{S}, (E'^j, F'^j))$, return $\sum_j d_j$.
- Else for any other input x, return $H_q(x)$.

Now, since $c^j, d^j \leftarrow \mathbb{F}_q^*$, both π_1 and π_2 have the same statistical distribution. Hence, the probablity of predicting the correct value of b is $\frac{1}{2}$ for any PPTA.

□

Proof of theorem 6

Proof. Suppose there is a PPTA $\mathbf{P'}$ that is capable of creating π such that $m' \neq m'' \wedge \mathbf{V}(\pi, m', \mathbf{S'}) = 1 \wedge \mathbf{V}(\pi, m'', \mathbf{S''}) = 1$ with a probability ϵ with Q queries to the random oracle. Now, the algorithm $\mathbf{P'}$ must have had invoked the random oracle with values $(m', S', (E^j, F^j))$ and $(m'', S'', (E^j, F^j))$ (The value of (E^j, F^j) don't change as c^j, d^j are same). Since the values $\pi = (c^j, d^j)$ are same and $\sum_j d_j$ has to be equal to the output of the oracle, it must be so that the oracle has returned the same value for both the inputs. The probability of the two oracle value being equal among Q invocations is $1 - \frac{^qP_Q}{q^Q} = 1 - \frac{q(q-1)(q-2)...(q-Q+1)}{q^Q} < 1 - \frac{(q-Q+1)^Q}{q^Q}$ which is negligible. □

Proof of theorem 7

Proof. The completeness theorem is trivially true since the verification protocol is deterministically computed, and the computations listed can be checked to be correct. □

Proof of theorem 8

Proof. A signature oracle must contain a proof oracle since the proof is part of the signature. The verification of the signature requires a valid computation of the ZkPLMT π. Let us assume that the adversary $\mathcal{A}$ can produce this proof with a probability δ. Theorem 3 proves that if an adversary is able to create π with a probability δ in the presence of the proof oracle, then it should be able to extract the discrete logarithm for some Y_k with a probability $\epsilon = \frac{1}{Q}\delta(\delta - 1/q)$ where Q is the number of queries to the proof oracle. The discrete log is the private key for this signature. We now consider theorem 5 and construct an advarsary that distinguishes the simulator from the true random oracle and prover. Since the simulator $\mathcal{S}$ in theorem 5 does not use w and only takes $(m, \mathbf{S})$, any extractor can only extract w from the output of $\mathcal{S}$ with a negligible probability (say σ). However, for the real prover, the probability of extraction is ϵ. Hence, the distinguisher can be implemented by extracting w using the extractor, and trying out all of the tuples in $\mathcal{S}$ to check whether it is correct. If w is correct, output 2, else output 1. So, $\eta = Pr[b = b'] = Pr[b' = 1|b = 1]Pr[b = 1] + Pr[b' = 2|b = 2]Pr[b = 2] \geq \frac{1}{2}(1 - \sigma) + \frac{1}{2}\epsilon = \frac{1}{2} + \frac{\epsilon - \sigma}{2}$. Since σ is negligible and theorem 5 proves that $\eta - \frac{1}{2}$ is negligible, so is ϵ. Hence, δ is also negligible. □

Proof of theorem 9

Proof. Suppose there exists an adversary $\mathcal{A}^{\mathcal{O}}$ that can distinguish b from b' with a non-negligible probability δ. We use it to break the zero-knowledge property of the underlying ZkPLMT. Let $\mathcal{S}$ be the simulator for the zero-knowledge property of the ZkPLMT. We construct the following distinguisher for distinguishing between the simulator and the true prover.

- Choose public keys $\{Q_1, Q_2, ..., Q_m\}$ randomly containing Q. We assume that the index of Q in this array is k, i.e. $Q = Q_k$.
- Compute $R^i = \mathfrak{q}P^i$.
- Generate $k_1 \leftarrow \{0..n\}, k_2 =\leftarrow \{0..n\} \setminus \{k\}$
- Genetate $b_1 \leftarrow \{1, 2\}$. Assign $k := k_{b_1}$.
- Compute the linear tuple $\mathbf{X} = ((G, Q), (P^i, R^i))$.
- Compute non-linear tuples $\mathbf{Y}^j = ((G, Q^j), (P^i, R^i))$ for all $j \neq k$. Assign $\mathbf{Y}_k := \mathbf{X}$.
- Provide $(M, \mathbf{Y^j}, k, \mathfrak{q})$ to the ZkPMLT zero-knowledge challenger
- The Challenger generates random $b \leftarrow \{1, 2\}$.
- The Challenger computes the simulation $(\pi_1, \mathcal{O}_S) = \mathcal{S}(M, \mathbf{Y^j})$
- The Challenger computes the proof $\pi_2 = \mathbf{P'}^{\mathcal{O}_S}(M, \mathbf{Y^j}, k, \mathfrak{q})$
- The Challenger shares the signature $(\pi_b, \mathcal{O}_S)$
- Compute $b'_1 = \mathcal{A}^{\mathcal{O}}(\Pi, k_1, k_2, M, pk^j_s, pk^i_d)$.
- If $b'_1 = b_1$, return $b' = 2$, else return $b' = 1$.

Since the ZkPLMT simulator does not use k, the probability of $b'_1 = b_1$ if the ZkPLMT challenger used a simulator would be $\frac{1}{2}$. Otherwise, if the ZkPLMT challenger used the true signing algorithm, the probability of $b'_1 = b_1$ would be $\frac{1}{2} \pm \delta$. Hence, the probability of $b = b'$ is $Pr[b = 1] \times Pr[b' = 1|b = 1] + Pr[b = 2] \times Pr[b' = 2|b = 2] = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times (\frac{1}{2} \pm \delta) = \frac{1}{2} \pm \frac{\delta}{2}$. Since δ is non-negligible, so is $\delta/2$.

□

Proof of theorem 10

Proof. It follows directly from theorem 6. □

Proof of theorem 11

Proof. By theorem 2, it must be that there exist some k such that $((G, Q_k), (P^i, R^i))$ is a linear tuple with a overwhelming probability. Also, by theorem 3, the signer must know $q_k \in \mathbb{F}_q$ such that $Q_k = q_k G$ with a overwhelming probability. Since $((G, Q_k), (P^i, R^i))$ is a linear tuple, $R^i = q_k P^i = q_k p^i G = p^i q_k G = p^i Q_k$. Hence, the l^{th} deanonymizer can find out $Q_k = p_l^{-1} R_l$ with a overwhelming probability. □

Proof of theorem 12

Proof. By theorem 2, it is proven that there exist some k such that $((G, Q_k), (P^i, R^i))$ is a linear tuple. Also, by theorem 11, we know that the l^{th} deanonymizer can find out $Q_k = p_l^{-1} R_l$. Hence the theorem is trivially true as the computation is deterministically correct given fixed t, u, p_l. □

Proof of theorem 13

Proof. By theorem 2, it is proven that there exist some k such that $((G, Q_k), (P^i, R^i))$ is a linear tuple. Also, by theorem 11, we know that the l^{th} deanonymizer can find out $Q_k = p_l^{-1} R_l$. Suppose an advarsary can provide a valid proof even when $Q_k \neq p_l^{-1} R_l$.

The prover must either have computed $D = pC$ correctly, or H_q produced the same output for two different values. Let us assume that (G, Q_k) and (P_l, R_l) are not linearly dependent. Suppose $P_l = pG$. Now, it could happen that the verifier had sent the value $C' = t'G + u'Q$ where $t' = t - ks$, $u' = u + s$ for some s. This means $C' = (t - ks)G + (u + s)Q = tG + uQ = C$. Hence the prover would have no way of distinguishing the two cases.

However, given $C = tG + uQ$ and $D = tP_l + uR_l$ as the verifier checked, there is only one value of t that would satisfy the verifier checks unless (G, Q) and (P_l, R_l) are linearly dependent (otherwise one can solve for t and u given the other values with infinite computational power). Since the prover could only see C, it cannot distinguish between u and u'; so the probability of it computing D correctly is at most $1/q$. The probability of a different D having the same hash is $1/q$. Hence, the probability of a successful verification for a non-linear set of pairs of points is at most $1/q + 1/q(1 - 1/q)$. □

Proof of theorem 14

Proof. To prove it, we must construct a deterministic algorithm $\mathcal{M}$ such that given a black box access to any arbitrary program acting as a verifier $\mathcal{V}^*$(possibly malicious), $\mathcal{M}$ is able to generate transcript that is computationally indistinguishable from an actual transcript of the interaction, i.e. (C, D, h, t, u), by any PPTA $\mathbf{A}$.

Our simulator $\mathcal{M}$ works as follows with black-box access to $\mathcal{V}^*$.

1) $\mathcal{M}$ receives C from $\mathbf{V}^*$.
2) $\mathcal{M}$ generates $h' \leftarrow \mathbb{F}_q^*$ and sends to $\mathcal{V}^*$.
3) $\mathcal{M}$ receives (t, u) from $\mathcal{V}^*$.
4) $\mathcal{M}$ checks $C = tG + uQ$, and computes $D = tP_l + uR_l$. If the check fails, the simulator outputs (C, D, h, t, u)
5) $\mathcal{M}$ computes $h = H_q(D)$
6) $\mathcal{M}$ again rewinds the verifier $\mathcal{V}^*$ to the point after the verifier returned C, but passes h instead of h' in this run.
7) If $\mathcal{V}^*$ outputs the same values for (t, u), the simulator outputs the transcript (C, D, h, t, u). Otherwise it outputs (C, D, h', t, u).

If the check $C = tG + uQ$ succeeds and $\mathbf{V}^*$ outputs the same values for u, t for both runs, (C, D, h, t, u) pass all tests for correctness. Hence, the generated transcript is indeed indistinguishable from an actual transcript by any PPTA. Otherwise, the failure scripts are indistinguishable.

□