Contingent payments from two-party signing and verification for abelian groups

Sergiu Bursuc
SnT, University of Luxembourg

Sjouke Mauw
SnT and DCS, University of Luxembourg

***Abstract*—The fair exchange problem has faced for a long time the bottleneck of a required trusted third party. The recent development of blockchains introduces a new type of party to this problem, whose trustworthiness relies on a public ledger and distributed computation. The challenge in this setting is to reconcile the minimalistic and public nature of blockchains with elaborate fair exchange requirements, from functionality to privacy. Zero-knowledge contingent payments (ZKCP) are a class of protocols that are promising in this direction, allowing the fair exchange of data for payment. We propose a new ZKCP protocol that, when compared to others, requires less computation from the blockchain and less interaction between parties. The protocol is based on two-party (weak) adaptor signatures, which we show how to instantiate from state of the art multiparty signing protocols. We improve the symbolic definition of ZKCP security and, for automated verification with Tamarin, we propose a general security reduction from the theory of abelian groups to the theory of exclusive or.**

1. Introduction

The computational guarantees and the distributed nature of blockchains, e.g. [59], [70], raise expectations that they can serve as trustworthy infrastructure for applications that usually require trusted third parties. A prominent such application is that of *fair exchange*, which should allow two parties to exchange data in a secure way. Indeed, since fair exchange was shown impossible in general without a trusted third party [26], [61], it has been confined to restricted functionalities or optimistic scenarios [5], [20], [27]. A natural question is whether the blockchain, guaranteeing *consistency* and *liveness* [41], [63], can contribute to more general solutions. The answer is not easy, since blockchain-based computation should be minimal, for efficiency and privacy reasons. As a consequence, most solutions incur significant overhead, relying on advanced cryptographic primitives and mediators [24], [36], [37], [44], trusted hardware [31], [69], or penalties [3], [4], [13].

A step towards more practical solutions is the functionality of zero-knowledge contingent payments. A *contingent payment* is executed between a sender and a receiver and allows the provision of certain type of data in exchange for payment. Furthermore, in a *zero-knowledge contingent payment* (ZKCP), the receiver should obtain no information about the data if the sender does not obtain payment. The data exchanged in ZKCP is called a *witness*, since its properties are formalised as a relation between an NP statement and a witness to it. The first protocol achieving ZKCP in Bitcoin was proposed in [71], and implemented in [55], based on the generic zk-SNARK framework for non-interactive zero-knowledge proofs [12]. The protocol relies on so-called *hashed timelock contracts*. Such a contract allows two parties to agree on a hash value and create a coin that can be claimed by one of the two parties by providing the inverse of that hash value. In the first step of the ZKCP protocol from [71], the sender $\mathcal{S}$ constructs a non-interactive zero-knowledge proof to convince the receiver $\mathcal{R}$ that the inverse of a hash value, together with additional encrypted data, reveals the witness. In the second step, $\mathcal{R}$ transfers a coin into a hashed timelock contract for that hash value. Finally, $\mathcal{S}$ redeems the payment by posting the inverse of the hash value on the Bitcoin ledger, where $\mathcal{R}$ can retrieve it and compute the witness. The *timelock* refers to the fact that the coin can be returned to its original owner if the party responsible for providing the inverse of the hash does not respond after a certain amount of time. The timelock mechanism can also be applied to standard coins, without a hashlock contract.

Security definitions for ZKCP and security proofs for this protocol have been proposed both in the symbolic [18] and in the computational model [60]. However, there are still several drawbacks in practice. The foremost is that hashed timelock contracts are not standard transactions in Bitcoin and add significant overhead. This limitation refers to the hash function required to be evaluated on the blockchain, but not to the timelock aspect which is negligible in comparison. Ideally we should have a protocol that uses only standard transactions that transfer funds between accounts based on signatures with associated keys, possibly with timelock constraints associated to certain coins. Even on blockchains like Ethereum that naturally support more general code, minimising the amount of on-chain computation is worthwhile. A second drawback of [71] is that the zero-knowledge proof has to be recomputed at every new session between a given sender and any receiver, even if it is for selling the same witness. The reason for this is that the inverse of the hash value giving access to the witness is linked to the zero-knowledge proof and is recorded on the blockchain after completing the exchange.

The problem of hashed timelock contracts in [71] has been discussed in [8], which proposed a second protocol based on standard Bitcoin. The protocol relies on a shared key between the sender and the receiver, which acts as an intermediary for payment between the two. Before making a payment towards the shared key, the receiver needs to be convinced that the followup signature transferring it to the sender will reveal enough information to compute the witness. For this, the parties jointly generate a signature with respect to the shared key and the sender

proves in zero-knowledge that this signature, together with additional encrypted data, reveals the witness. The fair-exchange happens when this signature, initially held by the sender, is posted on the blockchain, allowing the receiver to obtain the witness and the sender to obtain payment. The first drawback of this protocol is that, as in [71], the zero-knowledge proof cannot be reused, because it is linked to a particular signature. Second, the relation that needs to be proved in zero-knowledge is significantly more complex than in [71], as it needs to prove the validity of the signature, of the witness, and of the relation between them. That is why [8] aims to avoid generic zk proofs and designs a specific zk construction. It is based on the cut-and-choose technique, where many sessions are executed in parallel to derive security guarantees. Their construction expects that the underlying NP statement admits zero-knowledge proofs of a particular structure, which significantly limits the type of data to which the protocol can be applied. Finally, the zk proof construction of [8] is interactive, which adds more overhead compared to the non-interactive zero-knowledge proofs that are now standard in the context of blockchains [11], [12], [62].

Our contributions. We propose a new ZKCP protocol that has new interesting features when compared to the state of the art. We perform its formal specification and verification with the Tamarin prover. We make several additional contributions of independent interest described below.

New ZKCP protocol. The main improvement of our protocol with respect to [71] is that it works on standard Bitcoin, like [8]. It only assumes the ability of the ledger to return a coin to a previous owner after a timeout, but does not require hashlock transactions. An improvement over both protocols is that the same zero-knowledge proof can be reused for selling the same witness to different buyers across different sessions. This is important in a decentralised setting where one may generate many data elements for many potential buyers. The relation that we need to prove in zero-knowledge is much simpler than [8], allowing us to rely on generic non-interactive zero-knowledge proofs like [71]. This means we do not have a restriction on the class of statements: the protocol can be applied to exchange any data whose properties can be formalised as an NP relation.

Two-party weak adaptor signing. The notion of *adaptor signatures* has been recently introduced in the context of blockchains in order to improve their scalability and privacy [6], [38], [54], [66]. An adaptor signature scheme allows to construct a so-called pre-signature on a message, that can be adapted into a real signature on that message given a witness for a specified NP relation. Conversely, given a valid signature on that message, one can extract from the pre-signature a witness for that NP relation. A related notion of *lockable signatures* has been introduced in [67]. This notion is weaker than adaptor signatures, since it requires a party to provide a corresponding NP witness in order to construct a pre-signature and it does not allow the verification of a pre-signature. In this paper, we consider an intermediary notion between lockable signatures and adaptor signatures, that we call weak adaptor signatures. Like lockable signatures, this notion requires the NP witness in order to compute a pre-signature; like adaptor signatures, it allows to verify a pre-signature.

For our ZKCP protocol, the main tool that we need is a distributed protocol that allows two parties to construct a weak adaptor signature for a shared signing key. The signature scheme that is used in Bitcoin is ECDSA (elliptic curve DSA) [47]. Two recent multiparty ECDSA signing protocols with state-of-the-art efficiency are proposed in [42], [53]. We show that these protocols can be directly extended to compute ECDSA-based weak adaptor signatures in a distributed way, which is not the case for adaptor signatures. A pre-signature in this setting is simply a term $\beta + s$, where s is (part of) a fresh signature s and β is the witness for a suitably chosen NP relation. Our ZKCP protocol will ensure that the signature s allows the sender to obtain payment, while the mask β allows the receiver to obtain the witness. A zk proof will attest the relation between the public representation of β and a valid witness. The fair exchange happens when s reaches the blockchain.

Formal verification modulo abelian groups. To perform formal verification for our protocol, we need to model the abelian group (ag) where $\beta + s$ is performed. Unfortunately, the theory of ag is currently outside the scope of automated verification tools. The most advanced model is the one associated to the Diffie-Hellman theory in Tamarin [65], but therein the use of ag is restricted to the exponent, and protocol rules cannot directly use abelian group operations. ProVerif allows only a limited use of the Diffie-Hellman theory and, in order to handle associative-commutative operators, requires additional restrictions and encodings [49], [50]. There are several decidability results for security verification modulo abelian groups in the context of a passive intruder [51] or for a bounded number of sessions [32], [33]. To our knowledge they have not been implemented, and furthermore we are interested in automated verification for the general case of an active intruder and unbounded number of sessions.

In the family of equational theories that rely on an associative-commutative symbol, the state of the art is much better for the theory of exclusive or (xor). For xor, we have decidability results for a bounded number of sessions [23], [29], and also tools like Tamarin that work in practice for an unbounded number of sessions and no restrictions on the class of protocols [35]. Mathematically, the xor operation corresponds to an abelian group of order two, and its use in security protocols stems from the algebraic properties of such a group. A natural question is whether we can reduce security verification modulo ag (corresponding to any group) to security verification modulo xor (corresponding to a group of order two). For a significant class of properties, including all non-reachability properties and some reachability properties, we prove that this is indeed the case. We illustrate this approach by performing automated verification of our proposed protocol with Tamarin modulo xor, and then deriving the desired security guarantees modulo ag. The soundness of the reduction holds for any combination of the theory of abelian groups with another intruder theory, e.g. for modelling additional cryptographic primitives, and for any class of protocols. We prove our reduction in the more general setting of subsumed theories.

Better ZKCP definition. We improve the ZKCP symbolic security definition from [18], making it amenable to automation, since their property cannot be fully verified

with tools like Tamarin. We also make it more general, as the receiver property from [18] needs to be adapted according to the structure of the protocol. Finally, we also make it stronger. Indeed, the receiver property from [18] only ensures that the witness is deducible by the receiver, i.e. there exists a sequence of computation steps that allows to deduce the witness. However, a ZKCP protocol specifies a computation sequence that should allow the receiver to obtain the witness in a bounded number of steps. Our definition guarantees this stronger notion.

Paper structure. In Section 3 we present (weak) adaptor signing, ZKCP definitions and our corresponding constructions. Formal specifications for the verification of the protocol are in Section 4, our reduction results for intruder theories are in Section 5, and their application for the verification of the proposed protocol is in Section 6.

2. Preliminaries

We denote by $x \xleftarrow{\$} S$ the fact that x is uniformly sampled from S, by $x \leftarrow \mathsf{A}(y, \ldots)$ the fact that x is the result of a randomised algorithm A on inputs $y, \ldots$, and by $x := \mathsf{A}(y, \ldots)$ the result of a deterministic algorithm.

Basic cryptographic primitives. We consider a symmetric key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ and a key derivation function that generates encryption keys for Enc from elements of $\mathbb{Z}_q$. The key derivation function can be based on a hash function [48], and we denote it by H. The ECDSA signing scheme is represented by algorithms Sign and Verify. An ECDSA signature is composed by a pair of elements $(r, s) \in \mathbb{Z}_q \times \mathbb{Z}_q$. The element r is random and does not depend on the signed message nor on the secret key, while s is the principal element of the signature computed from the message, the secret key and r. Sometimes we will call s the signature, assuming that r is specified implicitly. We let g be the generator of the group of order q that underlies ECDSA. We note that computing the discrete logarithm in this group is hard. A secret-sharing scheme allows multiple parties to hold individual shares of a secret and use it when needed [10]. It is called additive when the secret is the sum of individual shares. A feature often used is that, from an additive secret-sharing of u and an additive secret-sharing of v, the parties can compute an additive secret-sharing of $u + v$ by locally adding their individual shares.

Zero-knowledge proofs allow a prover to convince a verifier that it holds a secret element w (also called witness) that is in a given relation with respect to a public element x (also called statement). The relation is formalised by a public function $\mathsf{f} \colon \{0,1\}^* \to \{0,1\}^*$ such that w and x are in the relation iff $x = \mathsf{f}(w)$. This is denoted by $\{(x; w) \mid x = \mathsf{f}(w)\}$. Of special interest are NP relations where f can be computed in polynomial time and finding a witness for a statement is hard. We consider non-interactive versions of zero-knowledge proofs, where the proof consists of a single term sent from the prover to the verifier [64]. In contingent payment protocols, we assume that the sender holds a witness w for a statement x in a relation f and the receiver wants to acquire it. In order to enable the fair exchange, the witness w is encoded in a way that makes the release of the witness contingent on the availability of other data. In this setting, we need to augment the statement x with the new elements that encode the witness and will allow the receiver to obtain it at the end of the protocol. The NP relation needs to be extended accordingly to formalise the correspondence between the desired witness and the new elements. For the protocol in [71], the new elements are a ciphertext c encrypting the witness and an element ρ that is the hash image of the corresponding decryption key. The corresponding relation, presented in the first line of Figure 1, shows that these elements are as expected. We denote by $\mathsf{Prove}_0(c, \rho; w, k)$ a non-interactive zero-knowledge proof for this relation. For our protocol, we will require a zero-knowledge proof for a slightly different relation. Namely, for new public elements c and ρ, the proof needs to show that c encrypts a valid witness w and that the decryption key is the hash of the discrete logarithm of ρ, as presented in the second line of Figure 1. We denote by $\mathsf{Prove}(c, \rho; w, \beta)$ a non-interactive zero-knowledge proof for this relation, and by Ver the corresponding verification algorithm.

Bitcoin. We consider a model of Bitcoin, presented in Figure 2, where the ownership of a coin with serial number sn by a party with public key pk is represented by a coin referred to by (sn, pk, ℓ) on the ledger. The label ℓ records whether a timeout constraint is associated to this coin: if $\ell = \top$, then the coin can be returned to the previous owner after a timeout; otherwise, the coin can only be spent by providing a signature. To transfer the coin owned by pk to another party pk', the ledger expects a valid ECDSA signature with respect to pk authorising the transfer. In that case a new coin (sn', pk', ℓ') is stored on the ledger, along with the signature that justifies the transfer from pk to pk'. We store all transaction data in a set $\mathcal{L}.\mathsf{Trans}$. An extension of this model are the so-called hashed timelock contracts. Such a contract creates a coin (sn, ρ, pk, ℓ) which can be transferred to the party pk if it provides the inverse of ρ with respect to a cryptographic hash function.

Two-party signing. We consider multiparty ECDSA signing protocols from [42], [53], instantiated to the case of two parties, allowing to generate signatures with respect to a shared secret key. These protocols do not guarantee fairness, but we will use them as a building block in order to obtain fairness in our protocol. In multiparty computation in general, and in particular [42], [53], the lack of fairness means that a malicious party may obtain the output and prevent the other parties from doing so [27]. However, the two parties can agree on which of them has advantage and can obtain the output first. We call this party the receiver, and the other party is the sender. The signature generation procedures from [42], [53] have two main parts: *i) Signing:* a fresh signature $\sigma = (r, s)$ is generated, r and g^s are published, and the parties obtain an additive secret-sharing of s; *ii) Opening:* the parties reveal their shares of s and the signature can be reconstructed by each party. We note that g^s is published during the signing operation so that the parties can verify (in the exponent) that (r, s) is indeed a valid signature before opening their shares of s in the clear.

Zero knowledge contingent payments. Informally, ZKCP security for the sender means that, whenever the adversary learns the witness, the sender can obtain a coin from the blockchain as payment. Symmetrically, whenever one of the receiver's coins is spent, the receiver should

Figure 1: Zero-knowledge proof relations for contingent payment protocols

$$\text{[71]:} \quad \Big\{(c,\rho,x;w,k) \;\mid\; c=\mathsf{Enc}_k(w) \;\wedge\; x=\mathsf{f}(w) \;\wedge\; \rho=\mathsf{H}(k)\Big\}$$

$$\text{Ours:} \quad \Big\{(c,\rho,x;w,\beta) \;\mid\; c=\mathsf{Enc}_k(w) \;\wedge\; x=\mathsf{f}(w) \;\wedge\; k=\mathsf{H}(\beta) \;\wedge\; \rho=g^{\beta}\Big\}$$

$$\text{[8]:} \quad \Big\{(c,\rho,x,pk;w,\sigma,\gamma) \;\mid\; c=\mathsf{Enc}_k(w) \;\wedge\; x=\mathsf{f}(w) \;\wedge\; k=\mathsf{H}(\sigma) \;\wedge\; \rho=\mathsf{Com}(\sigma,\gamma) \;\wedge\; \mathsf{Verify}(\sigma,pk)=\mathsf{ok}\Big\}$$

Figure 2: Ledger functionality $\mathcal{L}$

Setup	SignedTrans	TimedTrans
$\mathcal{L}.\mathsf{Coins} := \emptyset$	$\mathsf{In}(sn, pk', \ell', \sigma)$,	**if** $(sn, pk, \top) \in \mathcal{L}.\mathsf{Coins}$ **and**
$\mathcal{L}.\mathsf{Trans} := \emptyset$	**if** $(sn, pk, \ell) \in \mathcal{L}.\mathsf{Coins}$ **and**	$(sn_0, pk_0, \_, sn, pk) \in \mathcal{L}.\mathsf{Trans}$ **then** $\mathsf{Fr}(sn')$
Mine	$\mathsf{Verify}(\sigma, (sn, pk', \ell'), pk)$ **then** $\mathsf{Fr}(sn')$	$\mathcal{L}.\mathsf{Coins} := \mathcal{L}.\mathsf{Coins} \setminus (sn, pk, \top) \cup (sn', pk', \bot)$,
$\mathsf{Pk}(pk), \mathsf{Fr}(sn)$,	$\mathcal{L}.\mathsf{Coins} := \mathcal{L}.\mathsf{Coins} \setminus (sn, pk, \ell) \cup (sn', pk', \ell')$,	$\mathcal{L}.\mathsf{Trans} := \mathcal{L}.\mathsf{Trans} \cup (sn, pk, \mathtt{time}, sn', pk_0)$,
add $(sn, pk, \bot)$	$\mathcal{L}.\mathsf{Trans} := \mathcal{L}.\mathsf{Trans} \cup (sn, pk, \sigma, sn', pk')$	$\mathtt{event} : \mathsf{Spend}(sn, \mathtt{time}, pk_0)$
to $\mathcal{L}.\mathsf{Coins}$	$\mathtt{event} : \mathsf{Spend}(sn, \sigma, pk')$	

be able to learn a valid witness, typically by combining information received during the run of the protocol with information from the ledger. We give more formal specifications of security in the following sections. The first protocol for ZKCP was proposed in [71], and several follow-up works are devoted to its implementation and analysis [21], [39], [55], [60]. It relies on a zero-knowledge proof $\mathsf{Prove}_0(c, \rho, x; w, k)$ for the relation introduced in the previous section and Figure 1. The first step of the protocol is for the sender $\mathcal{S}$ to construct the public data c, ρ and send it to the receiver $\mathcal{R}$ along with the corresponding zero-knowledge proof. Upon successful verification, $\mathcal{R}$ is convinced that the preimage of ρ reveals the witness encrypted in c. It then transfers a coin into a hashed timelock contract on the ledger, with respect to the hash image ρ, recording the originator of the transaction and the party that is entitled to claim the coin by providing the inverse of ρ. Upon claiming this coin by $\mathcal{S}$, the inverse of ρ will be recorded on the ledger. Using this, the receiver can decrypt c and obtain the witness. If the sender aborts at any time, the receiver can obtain a refund after a timeout associated to the hashed timelock contract.

A second ZKCP protocol is proposed in [8], aiming to run on standard Bitcoin. It assumes a shared ECDSA key between $\mathcal{S}$ and $\mathcal{R}$. A basic step of the protocol allows $\mathcal{S}$ to obtain a fresh signature on a message with respect to this shared key. The message signed in this way is a Bitcoin transaction that transfers a particular coin from the shared key to $\mathcal{S}$. Before paying into the shared key, $\mathcal{R}$ needs to ensure that the follow-up signature will reveal a valid witness. For this $\mathcal{R}$ and $\mathcal{S}$ engage in two phases of cut-and-choose proofs: multiple instances of the protocol are run and some of them are opened for verification. First, $\mathcal{R}$ ensures that a sequence of commitments contains valid signatures of the agreed-upon transaction. Second, $\mathcal{R}$ ensures that they can be mapped to decryption keys that give access to zk proof transcripts allowing to extract a valid witness. This technique relies on an underlying zk proof system for the NP relation of interest, and it requires a witness extractor associated to zk proof transcripts that has special properties.

To ease the comparison between ZKCP proof relations, we give an abstract representation of the relation ensured by [8] in line 3 of Figure 1. We note, however, that in reality the relation for [8] involves many more terms and signatures depending on a security parameter, and zk proof transcripts are encrypted instead of the witness. The term pk in the relation represents the public part of the shared signing key between the sender and the receiver. Security of [8] relies on an assumption about ECDSA called strong unforgeability [2], [17]: an adversary against the signing scheme cannot construct a fresh signature σ' on a message m, even if it has previously seen a signature σ on the same message m. This forces the sender in the protocol described above to claim payment through one of the signatures linked to the cut-and-choose proofs. If the sender aborts and does not publish such a signature, the receiver can obtain a refund after a timeout associated to the shared key coin (or relying on timed commitments of sender's secret key shares [16]).

3. Two-party signing and contingent payment

Our protocol relies on a cryptographic building block that allows to combine an ECDSA signature σ with a secret element y into a term σ' such that: i) from σ' and y, one can obtain σ; ii) from σ' and σ, one can obtain y; iii) from σ' alone, one cannot derive σ or y. These three properties are satisfied by the recently introduced notion of *adaptor signatures* [6], [38], which has been applied to improve the scalability and privacy of blockchain-based payments [54], [66]. While we could directly use an adaptor signature in our protocol, it turns our that a weaker notion is sufficient for our purposes. We show how this weaker notion can be constructed for ECDSA and how multi-party ECDSA signing protocols from [42], [53] can be applied to distribute our construction between a sender and a receiver.

3.1. Two-party weak adaptor signatures

An adaptor signing scheme, as introduced in [6], consists in four algorithms $(\mathsf{pSign}, \mathsf{pVerify}, \mathsf{Adapt}, \mathsf{Extract})$ and is relative to a hard NP relation $\mathcal{R}$. The witnesses y for the relation $\mathcal{R}$ plays the role of a secret element to be combined with a signature, while the statement Y gives

public access to this element. The role of these algorithms is as follows:

- $\mathsf{pSign}(m, sk, Y)$ allows to construct a pre-signature on message m, that allows to obtain real signature given a witness corresponding to Y;
- $\mathsf{pVerify}(\sigma', m, pk, Y)$ allows to verify the correctness of a pre-signature σ';
- $\mathsf{Adapt}(\sigma', y)$ allows to obtain a signature σ from a pre-signature σ' given the corresponding witness y:
- $\mathsf{Extract}(\sigma', \sigma)$ allows to obtain a witness y for a pre-signature σ' given a corresponding signature σ.

We weaken this notion in the following sense: i) the pSign algorithm takes as input a witness y instead of a statement Y; ii) the statement Y is not sufficient to verify a pre-signature: we require in addition a (helper) statement Z that may be produced by a trusted party having access to the signing key or a corresponding signature. The resulting notion is similar to *lockable signatures* introduced in [67]. The difference is that the definition in [67] does not allow to extract the witness, does not allow verification with $\mathsf{pVerify}$, and only considers a particular NP relation. Therefore the notion that we propose is somewhere between lockable signatures and adaptor signatures.

Definition 1. A *weak adaptor signature scheme* consists in a signing scheme $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ and:

- an NP relation $\mathcal{R}_a$, called the adaptor relation;
- an NP relation $\mathcal{R}_v$, called the verification relation;

and a tuple of algorithms $(\mathsf{pSign}, \mathsf{pVerify}, \mathsf{Adapt}, \mathsf{Extract})$. These algorithms are the same as for an adaptor signature, except:

- $\mathsf{pSign}(m, sk, y)$ takes as third argument a witness y instead of a statement Y for the relation $\mathcal{R}_a$;
- $\mathsf{pVerify}(\sigma', m, pk, Y, Z)$ takes as an additional argument Z, for which (m, pk, Z) is expected to be a statement from the relation $\mathcal{R}_v$.

Following [6], the security properties for a (weak) adaptor signature scheme are: *unforgeability*: given any number of signatures and pre-signatures, an adversary without access to the secret key is not able to forge any signature or pre-signature on a new message; furthermore, *witness extractability* requires that, if such an adversary successfully adapted a pre-signature σ' into a signature σ, an honest party will be able to extract a witness from σ' and σ; *pre-signature adaptability* requires that, if a pre-signature σ' was successfully verified, then a valid signature can be obtained $\mathsf{Adapt}(\sigma', y)$, where y is a witness for Y. These properties are formally defined in a computational model in [6]. In this paper, we consider an abstract, Dolev-Yao style model of the adversary where these properties are enforced by the underlying term algebra. In particular, the formal security proof of our protocol implies that the Dolev-Yao adversary cannot break the security of the weak adaptor signature scheme that we consider as a building block in the protocol.

In the following, we consider the weak adaptor signing scheme defined by the ECDSA algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$, the relations $(\mathcal{R}_a, \mathcal{R}_v)$ and the algorithms $(\mathsf{pSign}, \mathsf{pVerify}, \mathsf{Adapt}, \mathsf{Extract})$ from Figure 3. The relation $\mathcal{R}_a$ states that $(Y, y) \in \mathcal{R}_a$ iff $y \in \mathbb{Z}_q$ is the discrete logarithm of Y. A pre-signature is defined in Figure 3 as simply the sum between the witness and (the second component of) a valid signature, while the adaptation and extraction algorithms perform the subtraction of the corresponding element (a witness or a signature). The verification relation states that $(m, pk, Z) \in \mathcal{R}_v$ iff Z is equal to (r, g^s) for a valid ECDSA signature (r, s) for message m and key pk. It can be easily checked that the scheme defined in this way is correct, i.e. for any valid pre-signature, the algorithms of verification, adaptation and extraction can be successfully performed.

Two-party weak adaptor signing. In order to apply a weak adaptor signature scheme in our protocol, we need a sub-protocol that allows to compute a pre-signature in a distributed manner. First, we define the expected properties of this sub-protocol.

Definition 2. A two-party weak adaptor signing protocol $\mathcal{AS}$ allows two parties to perform the following functions:

- $\mathcal{AS}.\mathsf{SharedPk}(\{\mathsf{id}_1, \mathsf{id}_2\})$: if a shared signing key for $\{\mathsf{id}_1, \mathsf{id}_2\}$ is not already stored, generate and store the secret key; return the corresponding public key.
- $\mathcal{AS}.\mathsf{Sender}(m, pk, y)$: a party $\mathcal{S}$ playing the role of the sender inputs a message m, a shared public key pk, and a witness y to be used for a pre-signature. The tuple $(\mathcal{S}, m, pk, y, \mathcal{R})$ is stored, where $\mathcal{R}$ is the expected receiver.
- $\mathcal{AS}.\mathsf{Receiver}(m, pk, Y)$: a party $\mathcal{R}$ playing the role of the receiver inputs a message m, a shared public key pk and a statement Y to be used for a pre-signature. When a tuple $(\mathcal{S}, m, pk, y, \mathcal{R})$ is available, check if $(Y; y) \in \mathcal{R}$, fetch the sk corresponding to pk, compute $\sigma' \leftarrow \mathsf{pSign}(m, sk, y)$, and return σ' to the receiver.

In Figure 4, we show how to extend the protocols for ECDSA two-party signing from [42], [53], presented in Section 2, in order to obtain a protocol for two-party weak adaptor for the scheme introduced in Figure 3: we show how to compute a pre-signature corresponding to a signature (r, s), i.e. $\sigma' = (r, y+s)$, where the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ have agreed on the public element $Y = g^y$ beforehand. We call $\mathcal{F}_{\mathsf{Sign}}$ the functionality implemented by the protocols from [42], [53]. The main idea for the construction in Figure 4 is that the signing operation implemented by $\mathcal{F}_{\mathsf{Sign}}$ provides each party with r, s_i, g^s, where (r, s) is a fresh signature for the desired message m and key sk,while and (s_1, s_2) is an additive secret-sharing of s. We note that (m, pk, g^s) is now a valid statement for the verification relation $\mathcal{R}_v$ in Figure 3. Based on this, the functions in $\mathcal{AS}$ are implemented as follows:

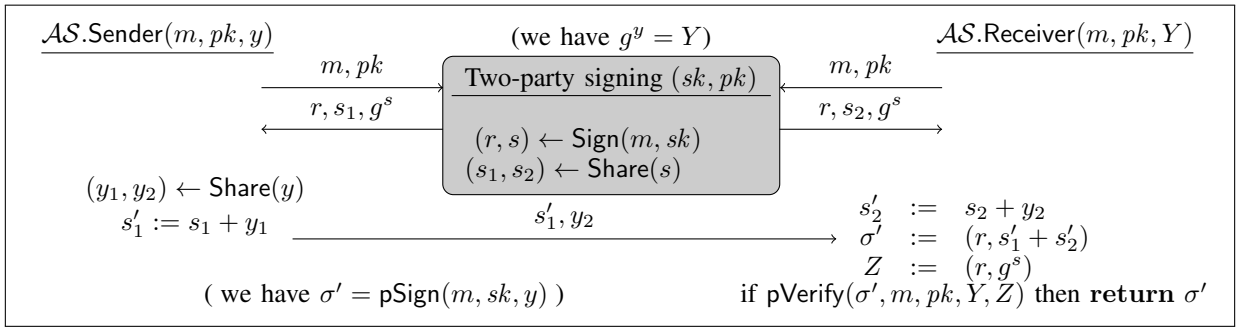
- $\mathcal{AS}.\mathsf{SharedPk}(\{\mathsf{id}_1, \mathsf{id}_2\})$: the parties generate a shared secret key as in $\mathcal{F}_{\mathsf{Sign}}$.
- $\mathcal{AS}.\mathsf{Sender}(m, pk, y)$: first phase: perform the signing operation from $\mathcal{F}_{\mathsf{Sign}}$ interactively with $\mathcal{R}$; second phase: generates an additive secret-sharing of y and send to $\mathcal{R}$ its respective share, along with the sum between $\mathcal{S}$'s shares of the signature s and of y.
- $\mathcal{AS}.\mathsf{Receiver}(m, pk, Y)$: first phase: perform signing operation from $\mathcal{F}_{\mathsf{Sign}}$ interactively with $\mathcal{S}$; second phase: after receiving the information from $\mathcal{S}$, $\mathcal{R}$ can compute $s' = y + s$ by summing up all shares. Finally, it performs the verification operation for the obtained pre-signature (r, s'). Note that the statement (m, pk, r, g^s) that is required to run $\mathsf{pVerify}$ is provided by the functionality from $\mathcal{F}_{\mathsf{Sign}}$.

Figure 3: ECDSA-based weak adaptor signature

1) Adaptor relation : $\mathcal{R}_a = \{(Y; y) \mid Y = g^y \ \}$
2) Verification relation : $\mathcal{R}_v = \{(m, pk, Z; \sigma) \mid \mathsf{Verify}(\sigma, m, pk) = \mathsf{true} \ \wedge \ \sigma = (r, s) \ \wedge \ Z = (r, g^s)\}$
3) Algorithms :

$\mathsf{pSign}(m, sk, y)$	$\mathsf{pVerify}(\sigma', m, pk, Y, Z)$	$\mathsf{Adapt}(\sigma', y)$	$\mathsf{Extract}(\sigma', \sigma)$
$(r, s) \leftarrow \mathsf{Sign}(m, sk)$	$(r', s') := \sigma', (r, W) := Z$	$(r, s') := \sigma'$	$(r, s') := \sigma', (r, s) := \sigma$
return $(r, s + y)$	**if** $r' = r \ \wedge \ Y \cdot W = g^{s'}$ **return** true	**return** $(r, s' - y)$	**return** $s' - s$

Figure 4: In gray: two-party ECDSA signing provided by [42], [53]. In clear: extension for weak adaptor signing



3.2. Contingent payment syntax and security

The ZKCP protocols from [8], [71], as well as our protocol, follow a common structure: 1) the sender and the receiver agree on an encoding of the witness and funds are committed by the receiver on the ledger; 2) data is exchanged and verified among the two parties; 3) the parties finalise the protocol by interacting with the ledger, either to obtain payment or to extract the witness. We introduce a syntax to make explicit the algorithms for each phase and clarify the specification of ZKCP protocols.

Definition 3. A *zero-knowledge contingent payment protocol* consists of two roles $\{\mathcal{S}, \mathcal{R}\}$ and, for each role $\mathcal{X} \in \{\mathcal{S}, \mathcal{R}\}$, a triple of (interactive) algorithms:

- $\mathsf{Setup}_{\mathcal{S}}(w)$ takes as input a witnes and outputs a state for the sender, while $\mathsf{Setup}_{\mathcal{R}}(x, sk_{\mathcal{R}}, pk_{\mathcal{S}})$ takes as input a statement, the secret key of the receiver and the public key of the sender;
- $\mathsf{Exchange}_{\mathcal{S}}(\mathsf{state}_0, pk_{\mathcal{S}}, pk_{\mathcal{R}})$ - the input of this algorithm is a state returned by $\mathsf{Setup}_{\mathcal{S}}$, and the public keys of the sender and receiver;
- $\mathsf{Settle}_{\mathcal{X}}(\mathsf{state}_1)$ - the input of this algorithm is a state returned by $\mathsf{Exchange}_{\mathcal{X}}$.

For a sender, we have $\mathsf{Setup}_{\mathcal{S}}(w, sk_{\mathcal{S}}, pk_{\mathcal{R}})$, where w is the witness to be transferred. Note that the secret key of the sender is not required for the protocol execution, reflecting the fact that the sender only uses the blockchain infrastructure to receive money. For a receiver, we have $\mathsf{Setup}_{\mathcal{R}}(x, sk_{\mathcal{R}}, pk_{\mathcal{S}})$, where x is the corresponding statement for the desired witness. In this section, we only provide an informal definition of security, that we instantiate formally in Section 4. For this, we assume a notion of events that can be added at certain points in the protocol specification, recording that certain actions have occurred during the execution of a protocol. Events can be defined both in computational models, e.g. in CryptoVerif [14], and in formal models, e.g. in ProVerif [15] or Tamarin [56], where they are called action facts. For ZKCP security, we assume the following events:

- $\mathsf{Trans}(sn, \ell, pk')$ - recording that a coin sn is transferred to pk' on the ledger; the label ℓ records either a signature on the corresponding transaction, or we have $\ell = \mathtt{time}$ to mark the lapse of a timeout associated to sn.
- $\mathsf{Setup}(w)$ - recording that an honest sender has performed the setup for a given witness.
- $\mathsf{Claim}(pk, w, sn)$ - recording that an honest sender with public key pk claims a coin with serial number sn as payment for a given witness w; this event will typically occur in the $\mathsf{Settle}_{\mathcal{S}}$ algorithm.
- $\mathsf{Pay}(pk, sn, x)$ - recording that an honest receiver with public key pk has paid for a witness for the statement x and that this has resulted in a coin with serial number sn on the blockchain ledger; this event will typically occur in the $\mathsf{Exchange}_{\mathcal{R}}$ algorithm.
- $\mathsf{End}(pk, sn, w)$ - recording a finished session for a receiver obtaining a purported witness w when the coin sn is spent on the ledger. This event should occur in the $\mathsf{Settle}_{\mathcal{R}}$ algorithm.

We refer to Figure 5 for an example ZKCP protocol specification and the specification of corresponding events for security. Informally, a ZKCP protocol is secure if:

- Sender security: whenever the adversary learns a new witness w, an honest sender obtains payment for w, i.e. the events $\mathsf{Claim}(pk_{\mathcal{S}}, w, sn)$ and $\mathsf{Trans}(sn, \sigma, pk_{\mathcal{S}})$ are executed. Due to the fact that the attacker controls the network, we need to relax this security definition to account for the fact that the claim of the sender may not reach the blockchain ledger in time, in which case $\mathsf{Trans}(sn, \mathtt{time}, pk')$ may be executed for a key pk' controlled by the adversary.

- Receiver security: by the end of a protocol session corresponding to a statement x, the receiver can either obtain a witness for x or be refunded for any coin it has paid, i.e. we have $\mathsf{Pay}(pk, sn, x) \ \wedge \ \mathsf{Trans}(sn, \ell, pk') \wedge \mathsf{End}(pk, sn, w) \Rightarrow (x, w) \in \mathcal{R} \vee pk' = pk$.

Formal instances for these two notions of security in the symbolic model are given in Definition 4 from Section 4.3. A computational definition for ZKCP security

has been proposed in [60]. Their definition is focused on the security of cryptographic components and is targeted for the particular components used in the protocol from [71]. It also does not consider the blockchain environment where the protocol is executed.

3.3. Proposed contingent payment protocol

We assume $\mathcal{S}$ and $\mathcal{R}$ have established a shared signing, e.g. by running the key generation protocol from [42], [53]. Our proposed protocol is described in Figure 5. The main idea is for the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ to jointly compute a pre-signature encoding a signature σ and a secret element β, where β allows $\mathcal{R}$ to obtain the desired witness, while σ is allows $\mathcal{S}$ to obtain payment. To protect the witness and the secret key, σ' need to be computed in a distributed way, which we do relying on the protocol from Figure 4. We also use zk proof relation Prove introduced in Section 2 in order to convince $\mathcal{R}$ that the discrete logarithm of $\rho = g^{\beta}$ allows to obtain the desired witness. In our protocol, we can let $\mathcal{R}$ have advantage in the pre-signing functionality, obtain σ' first, and forward it to $\mathcal{S}$. The blockchain will ensure fairness: $\mathcal{R}$ can obtain β iff $\mathcal{S}$ can obtain payment.

Setup. $\mathcal{S}$ generates $\beta \in \mathbb{Z}_q$, derives the key $k \leftarrow \mathsf{H}(\beta)$ and publishes an encryption of the witness with this key, g^{β} and a zk proof showing that these encode a valid witness. $\mathcal{R}$ verifies the zk proof and if it is valid transfers a coin to the public key shared with $\mathcal{S}$.

Exchange. The parties compute $\sigma' := \mathsf{pSign}(\tau, pk, \beta)$, where g^{β} is provided by $\mathcal{S}$ at setup, while τ is a transaction that transfers the shared coin created at the previous step to $\mathcal{S}$. $\mathcal{R}$ obtains σ' and forwards it to $\mathcal{S}$.

Settle. $\mathcal{S}$ adapts σ' to compute σ and posts it to the blockchain to obtain payment. From the blockchain ledger, $\mathcal{R}$ can obtain σ and then extract β from σ', finally computing the witness. We provide formal security proofs in a symbolic Dolev-Yao model in the next section. In the following, we argue informally why security holds for each party.

Sender security. Assuming the security of the encryption scheme, the only way the adversary can obtain the witness is by decrypting the corresponding ciphertext. Assuming the random oracle property for H, the only way of obtaining the decryption key is by applying H to β. From the hardness of computing discrete logarithms, computing β before the settling phase requires extracting it from σ', for which a valid signature σ is required. Finally, computing σ is impossible by the unforgeability of ECDSA and the security of the protocol in [42], [53].

Receiver security. The only way in which $\mathcal{R}$ can lose a coin as a result of the protocol is if a valid signature for pk is posted on the ledger. From the strong unforgeability assumption for ECDSA, discussed in Section 2, and from the security of [42], [53], the adversary can obtain such a signature σ only if a corresponding pre-signature σ' was obtained by $\mathcal{R}$ from the (adaptor) signing functionality. Then, relying on σ from the ledger, $\mathcal{R}$ can extract β and compute the witness. The zero-knowledge proof relation ensures that the witness obtained in this way is valid.

Instance from adaptor signatures. We conclude this section by noting that the protocol in Figure 5 could also be instantiated based on the stronger notion of adaptor signatures from [6], relying on the same zero-knowledge proof relation Prove to connect the witness in the adaptor signature with the witness in the desired functionality. What would be different is the way of computing a pre-signature in a distributed way. In our instantiation of weak adaptor signatures in Figure 3, a pre-signature is simply a masked version of a fresh signature, which does not depend on the witness for the adaptor relation. As a consequence, we can have a direct, generic extension of the protocols in [42], [53] in order to compute a pre-signature in a distributed way. On the other hand, in current constructions of adaptor signatures the randomness of the signature is determined by the secret witness of the adaptor relation [6], [38]. As a consequence, the corresponding distributed protocols in [54], [58] require additional zk proofs to be plugged into a two-party signing protocol. In our construction from Figure 4, no zk proof is required, except those used for implementing the distributed signing functionality in [42], [53].

Comparison with previous ZKCP protocols. Some notable features of our proposed protocol are the following: i) it does not require the application of hash functions on the blockchain; ii) the zero-knowledge proof is generic and non-interactive; iii) the zero-knowledge proof and the secret β can be reused across different protocol sessions to sell the same witness to different receivers. The protocol from [71] satisfies point ii), but not points i) and iii). The protocol from [8] satisfies the point i), but not points ii) and iii).

4. Formal specifications

We first introduce the symbolic model used in Tamarin for the verification of security protocols, referring to [56], [65] for more details. Then we present our formal models for the cryptographic primitives, the ideal functionalities, the security properties and the proposed protocol.

4.1. Formal specification preliminaries

The cryptographic primitives are represented by an equational theory, i.e. a set of function symbols, constants and equations. A term is constructed from function symbols applied recursively to variables from a set denoted by $\mathcal{X}$, to constants and to names - typically representing atomic or fresh data. Considering the function symbols $\{\circ, \mathtt{i}, 1\}$, the theories for abelian groups (ag) and exclusive or (xor) are defined as in Figure 6. The last two equations in $\mathcal{E}_{\mathsf{ag}}$ and $\mathcal{E}_{\mathsf{xor}}$ represent the associativity and commutativity of $\circ$, denoted by AC. We note that the equation $\mathtt{i}(x) = x$ is not usually part of the xor theory, but adding it allows simpler statements and proofs for our reduction between theories, and it does not pose any challenge for Tamarin. This equation can be justified computationally by taking $\mathtt{i}(.)$ as being the identity function, and also noting that the inverse of an element in the group of order two that underlies xor is itself. Sometimes we need to reason about several symbols that satisfy ag or xor, e.g. $+$ and $*$. In that case, we annotate the corresponding theory with a superscript, i.e. $\mathcal{E}^{+}_{\mathsf{ag}}, \mathcal{E}^{+}_{\mathsf{xor}}, \mathcal{E}^{\star}_{\mathsf{ag}}, \ldots$ and we replace $\circ$ and $\mathtt{i}$ with corresponding symbols in the equations for $\mathcal{E}_{\mathsf{ag}}$. All equalities in the following are implicitly modulo the underlying equational theory. For protocol

Figure 5: Algorithms defining our protocol for ledger $\mathcal{L}$ and two-party weak adaptor signing scheme $\mathcal{AS}$

$\mathsf{Setup}_{\mathcal{S}}(w)$

$\beta \xleftarrow{\$} \mathbb{Z}_q^\star, \rho := g^\beta, k := \mathsf{H}(\beta)$,
$c \leftarrow \mathsf{Enc}(w,k), \pi \leftarrow \mathsf{Prove}(c,\rho;w,\beta)$,
$\mathtt{event}:\mathsf{Setup}(w)$,
$\mathsf{Out}(c,\rho,\pi)$, **return** state_0

$\mathsf{Exchange}_{\mathcal{S}}(\mathsf{state}_0, pk_{\mathcal{S}}, pk_{\mathcal{R}})$

$pk \leftarrow \mathcal{AS}.\mathsf{SharedPk}(\{pk_{\mathcal{S}}, pk_{\mathcal{R}}\})$,
if $(sn, pk, \top) \in \mathcal{L}.\mathsf{Coins}$ **and** $sn \notin \mathsf{Used}$
then $\mathsf{Used} := \mathsf{Used} \cup \{sn\}$, $\tau := (sn, pk_{\mathcal{S}}, \bot)$
$\mathcal{AS}.\mathsf{Sender}(\tau, pk, \beta)$, **return** state_1

$\mathsf{Settle}_{\mathcal{S}}(\mathsf{state}_1)$

$\mathsf{In}(\sigma'), \sigma := \mathsf{Adapt}(\sigma', \beta)$
if $\mathsf{Verify}(\sigma, \tau, pk) = \mathsf{true}$
then $\mathsf{Out}(sn, pk_{\mathcal{S}}, \sigma)$
$\mathtt{event}:\mathsf{Claim}(pk_{\mathcal{S}}, w, sn)$

$\mathsf{Setup}_{\mathcal{R}}(x, sk_{\mathcal{R}}, pk_{\mathcal{S}})$

$\mathsf{In}(\pi, c, \rho), pk_{\mathcal{R}} := \mathsf{pub}(sk_{\mathcal{R}})$
if $\mathsf{Ver}(\pi, c, \rho, x)$ **and**
$(sn_0, pk_{\mathcal{R}}, \ell_0) \in \mathcal{L}.\mathsf{Coins}$ **then**
$pk \leftarrow \mathcal{AS}.\mathsf{SharedPk}(\{pk_{\mathcal{S}}, pk_{\mathcal{R}}\})$
$\sigma_0 \leftarrow \mathsf{Sign}((sn_0, pk, \top), sk_{\mathcal{R}})$
$\mathsf{Out}(sn_0, pk, \top, \sigma_0)$, **return** state_0

$\mathsf{Exchange}_{\mathcal{R}}(\mathsf{state}_0)$

if $(sn_0, pk_{\mathcal{R}}, \sigma_0, sn, pk) \in \mathcal{L}.\mathsf{Trans}$
then $\mathtt{event}:\mathsf{Pay}(pk_{\mathcal{R}}, sn, x)$
$\tau := (sn, pk_{\mathcal{S}})$,
$\sigma' := \mathcal{AS}.\mathsf{Receiver}(\tau, pk, \rho)$,
$\mathsf{Out}(\sigma')$, **return** state_1

$\mathsf{Settle}_{\mathcal{R}}(\mathsf{state}_1)$

if $(sn, \_, \sigma, \_, \_) \in \mathcal{L}.\mathsf{Trans}$ **then**
$\beta \leftarrow \mathsf{Extract}(\sigma', \sigma), k := \mathsf{H}(\beta)$,
$w := \mathsf{Dec}(c, k)$,
$\mathtt{event}:\mathsf{End}(pk_{\mathcal{R}}, sn, w)$

Corresponding Tamarin specifications $\mathcal{P}_{\mathcal{S}}$ and $\mathcal{P}_{\mathcal{R}}$ using $\mathcal{P}_{\mathcal{NP}}, \mathcal{P}_{\mathcal{PK}}, \mathcal{P}_{\mathcal{L}}, \mathcal{P}_{\mathcal{AS}}$ from Figure 8. See full code in [1].

Rule $\mathsf{Setup}_{\mathcal{S}}$

$[\ \mathsf{Witn}(w),$
$\mathsf{Fr}(\beta), \rho = \mathsf{g}^\beta, k = \mathsf{h}(\beta), c = \mathsf{enc}(w,k),$
$\pi = \mathsf{prove}(c, \rho, w, \beta)\ ]$
$-[\ \mathsf{Setup}(w)\ ]\rightarrow[\ \mathsf{Out}(c,\rho,\pi), \mathsf{state}_0\ ]$

Rule $\mathsf{Exchange}_{\mathcal{S}}$

$[\ \mathsf{state}_0,\ \mathsf{Pk}(pk_{\mathcal{S}}), \mathsf{Pk}(pk_{\mathcal{R}}),$
$\mathcal{AS}.\mathsf{SharedPk}(\{pk_{\mathcal{S}}, pk_{\mathcal{R}}\}, pk),$
$\mathcal{L}.\mathsf{Coin}(sn, pk, \ell), \tau = \langle sn, pk_{\mathcal{S}}, \mathtt{notime}\rangle\ ]$
$-[\ \mathsf{Once}(\langle \mathsf{used}, sn\rangle), \mathsf{Honest}(pk_{\mathcal{S}})\ ]\rightarrow$
$[\ \mathcal{AS}.\mathsf{Sender}(pk_{\mathcal{S}}, \tau, pk, \beta), \mathsf{state}_1\ ]$

Rule $\mathsf{Settle}_{\mathcal{S}}$

$[\ \mathsf{state}_1, \mathsf{In}(s'), s = s' - \beta\ ]$
$-[\ \mathsf{verify}(s, \tau, pk) = \mathsf{ok},$
$\mathsf{Claim}(pk_{\mathcal{S}}, w, sn)\ ]\rightarrow$
$[\ \mathsf{Out}(\tau, s)\ ]$

Rule $\mathsf{Setup}_{\mathcal{R}}$

$[\ \mathsf{Sk}(sk_{\mathcal{R}}), pk_{\mathcal{R}} = \mathsf{pub}(sk_{\mathcal{R}}), \mathsf{Pk}(pk_{\mathcal{R}}),$
$\mathcal{AS}.\mathsf{SharedPk}(\{pk_{\mathcal{R}}, pk_{\mathcal{S}}\}, pk),$
$\mathsf{Stm}(x), \mathsf{In}(c,\rho,\pi), \mathcal{L}.\mathsf{Coin}(sn_0, pk_{\mathcal{R}}, \ell_0),$
$\tau_0 = \langle sn_0, pk, \mathtt{time}\rangle, \mathsf{Fr}(r),$
$s_0 = \mathsf{sign}(\tau_0, sk_{\mathcal{R}}, r)\ ]$
$-[\ \mathsf{ver}(\pi, c, \rho, x) = \mathsf{ok}\ ]\rightarrow$
$[\ \mathsf{Out}(\tau_0, s_0), \mathsf{state}_0\ ]$

Rules $\mathsf{Exchange}^1_{\mathcal{R}}$ and $\mathsf{Exchange}^2_{\mathcal{R}}$

$[\ \mathsf{state}_0, \mathcal{L}.\mathsf{Trans}(sn_0, pk_{\mathcal{R}}, s_0, sn, pk),$
$\mathcal{L}.\mathsf{Coin}(sn, pk, \ell), \tau = \langle sn, pk_{\mathcal{S}}, \mathtt{notime}\rangle]$
$-[\ \mathsf{Pay}(pk_{\mathcal{R}}, sn, x)\ ]\rightarrow$
$[\ \mathcal{AS}.\mathsf{Receiver}(sk_{\mathcal{R}}, \tau, pk, \rho), \mathsf{state}'_1\ ]$
$[\ \mathsf{state}'_1,$
$\mathcal{AS}.\mathsf{ReceiverOutput}(sk_{\mathcal{R}}, \tau, pk, \rho, s')\ ] \Rightarrow$
$[\ \mathsf{Out}(s'), \mathsf{state}_1\ ]$

Rule $\mathsf{Settle}_{\mathcal{R}}$

$[\mathsf{state}_1,$
$\mathcal{L}.\mathsf{Trans}(sn, \_, s, \_, \_),$
$\beta = s' - s,$
$k = \mathsf{h}(\beta), w = \mathsf{dec}(c, k)\ ]$
$-[\ \mathsf{End}(pk_{\mathcal{R}}, sn, w)\ ]\rightarrow[\ ]$

Figure 6: Equational theories for ag and xor

$\mathcal{E}_{\mathsf{ag}}$	$\mathcal{E}_{\mathsf{xor}}$
$x \circ \mathtt{i}(x) = \mathtt{1}$	$x \circ x = \mathtt{1}$
$x \circ \mathtt{1} = x$	$x \circ \mathtt{1} = x$
$\mathtt{i}(\mathtt{i}(x)) = x$	$\mathtt{i}(x) = x$
$x \circ y = y \circ x$	$x \circ y = y \circ x$
$(x \circ y) \circ z = x \circ (y \circ z)$	$(x \circ y) \circ z = x \circ (y \circ z)$

verification, equational theories are usually represented by an equivalent AC-convergent rewrite system, where each term has a unique normal form modulo AC [34]. Such a system is also called an *intruder theory* [9], [30], [68]. We denote by $t{\downarrow}$ the normal form of a term t. When it is not clear from the context, we annotate $\downarrow$ with a symbol representing the corresponding intruder theory. A term that is in normal form wrt to $\mathcal{I}$ is called in $\mathcal{I}$-normal form. To express security protocols, the language is extended with fact symbols for adversarial knowledge, protocol state, freshness information, etc. A fact is represented by $F(t_1, \ldots, t_k)$, where F is a fact symbol and $t_1, \ldots, t_k$ are terms. There are the following special fact symbols: K - for attacker knowledge; Fr - for fresh data; In and Out - for protocol inputs and outputs. Other symbols may be added as required, e.g. for representing the protocol state. Facts can be persistent (consumed any number of times) or linear (consumed at most once). The notation $!F$ is used in Tamarin to distinguish persistent facts, but most of the facts in our models will be persistent, so we do not use this notation in the paper to avoid clutter.

A multiset rewriting rule is defined by $[L]-[\ M\ ]\rightarrow[N]$, where L, M, N are multisets of facts called respectively premises, actions, and conclusions. We denote such a rule by $[L] \Rightarrow [N]$ when M is empty. To ease protocol specification, the syntax of multiset rules is extended with variable assignments and equality constraints, i.e. we can write rules of the form $[L]-[\ E, M\ ]\rightarrow[N]$ where L may contain expressions $x = t$ to define local variables and E is a set of equations of the form $u = v$. Equations are not directly supported in Tamarin, but can be easily encoded within its language. For two multisets of facts M_0, M_1 and rule $P = [L]-[\ E, M\ ]\rightarrow[N]$ we say that M_1 can be obtained from M_0 by applying the rule P, instantiated with θ if: (1) every equality in $E\theta$ is true; (2) every fact in $L\theta{\downarrow}$ is included in M_0 (counting multiplicities for linear facts); (3) M_1 is obtained from M_0 by removing linear facts included in $L\theta{\downarrow}$ and adding all facts from $N\theta{\downarrow}$.

A special set of *message deduction rules* defines how the attacker can derive new knowledge and make use of existing knowledge to interact with the protocol. Within this set, we distinguish *network deduction rules* and *intruder deduction rules*. Network deduction rules are fixed: they define outputs, inputs, public and fresh data.

$$[\mathsf{Out}(x)] \Rightarrow [\mathsf{K}(x)] \qquad [\mathsf{K}(x)] \Rightarrow [\mathsf{In}(x)]$$
$$[\_] \Rightarrow [\mathsf{K}(y)] \qquad [\_] \Rightarrow [\mathsf{Fr}(z)] \qquad [\mathsf{Fr}(x)] \Rightarrow [\mathsf{K}(x)]$$

The semantics ensures that y and z in the rules above are instantiated to public, resp. fresh names. A fact of the form $\mathsf{K}(t)$ represents that the intruder knows the message t. Intruder deduction rules define operations on messages and are of the form $[\mathsf{K}(x_1), \ldots, \mathsf{K}(x_k)] \Rightarrow [\mathsf{K}(f(x_1, \ldots, x_k))]$, for any symbol f from the equational theory. Another class of multiset rules are *protocol rules*, which model the execution of the protocol by honest parties. For a rule R, we let $lhs(R)$, $rhs(R)$, $act(R)$ be respectively the set of its left-hand side facts (i.e. premisses), of its right-hand side facts (i.e. conclusions) and of its action facts. We use the following notation:

- $M_0 \xrightarrow{R,\sigma} M_1$ if M_1 can be obtained from M_0 by applying a rule R instantiated with substitution σ (this is called a transition);
- for a set of rules $\mathcal{P}$, we let $M_0 \stackrel{\mathcal{P}}{\Rightarrow} M_1$ if M_1 can be obtained from M_0 by applying a sequence of transitions using rules from $\mathcal{P}$. Such a sequence of transitions is called a trace of $\mathcal{P}$.

We let $traces(\mathcal{P})$ be the set of all traces of a set of rules $\mathcal{P}$. All terms that occur in a trace can be assumed to be in normal form with respect to the intruder theory. Consider a trace τ obtained by applying n multiset rules that ends in a multiset of facts M. We say that such a trace has length n, and denote the length of τ by $|\tau|$. Elements of the set $\{1, \ldots, n\}$ are called the timepoints of τ. For every $i \in \{1, \ldots, n\}$, we let R_i be the rule applied at step i and σ_i be the corresponding substitution. We define:

- $facts(\tau, i) = act(R_i)\sigma_i{\downarrow}$ if R_i is a protocol or network deduction rule;
- $facts(\tau, i) = \{\mathsf{K}(v\sigma_i{\downarrow})\}$ if R_i is a message deduction rule with $rhs(R_i) = \{\mathsf{K}(v)\}$
- $facts(\tau) = (\ facts(\tau, 1); \ldots; facts(\tau, n); M\ )$

We consider a set of timepoint variables, denoted by $i, j, l, \ldots$, which will be interpreted over rational numbers. A *trace atom* is either $\bot$, or a timepoint ordering $i < j$, or a timepoint equality $i = j$, or a term equality $t_1 = t_2$ or an action fact $\mathsf{F}@i$ for a fact F and a timepoint variable i. Atoms involving only timepoints are called *timepoint atoms*, and all others are called *term atoms*. A *trace formula* is a first-order logic formula obtained from trace atoms by applying the usual quantification and logical connectives. A *trace property* is a trace formula where all variables are bounded by a quantifier. When quantifiers are missing in a formula $\Phi_0 \Rightarrow \Phi_1$, we assume all variables in $\mathsf{var}(\Phi_0)$ are universally quantified, and all variables in $\mathsf{var}(\Phi_1) \setminus \mathsf{var}(\Phi_0)$ are existentially quantified. If a formula does not contain timepoint atoms, we omit the notation $@i$ from its action facts. For example, the formulas $\forall i, x. F(x)@i \Rightarrow \exists j, y. G(x, y)@i$ and $F(x) \Rightarrow G(x, y)$ are the same according to notations above.

The satisfaction relation $\tau \models \Phi$, for a trace τ and a trace property Φ is defined recursively starting from atoms and applying the usual semantics of logical connectives. For an action fact, the semantics is: $\tau \models F@i$ iff $F{\downarrow} \in facts(\tau, i)$. For other atoms the semantics is as expected from the syntax. Then we have $\mathcal{P} \models \Phi$ iff $\forall \tau \in traces(\mathcal{P}).\ \tau \models \Phi$. When the equational theory $\mathcal{E}$ is not clear from the context, we use the notation $(\mathcal{P}, \mathcal{E}) \models \Phi$. For protocol specification, it is often convenient to consider restrictions in addition to rules. Restrictions are modeled by a trace property Ψ and we use the notation $(\mathcal{P}, \Psi, \mathcal{E}) \models \Phi$ for $(\mathcal{P}, \mathcal{E}) \models \Psi \Rightarrow \Phi$, i.e. the property Φ holds assuming the restrictions Ψ.

Figure 7: Equational theory $\mathcal{E}_{\mathsf{zkcp}}$ for our protocol model

$$\begin{aligned}
\mathcal{E}_{\mathsf{zkcp}} &: \quad \mathcal{E}^{+}_{\mathsf{ag}} \cup \mathcal{E}^{*}_{\mathsf{ag}} \cup \mathcal{E}_{\mathsf{exp}} \cup \mathcal{E}_{\mathsf{enc}} \cup \mathcal{E}_{\mathsf{sign}} \cup \mathcal{E}_{\mathsf{zk}} \\
\mathcal{E}^{+}_{\mathsf{ag}} &: \quad \text{abelian group theory for the symbol } + \\
\mathcal{E}^{*}_{\mathsf{ag}} &: \quad \text{abelian group theory for the symbol } * \\
\mathcal{E}_{\mathsf{exp}} &: (x^y)^z = x^{(y*z)} \qquad \mathcal{E}_{\mathsf{enc}} : \mathsf{dec}(\mathsf{enc}(x, y), y) = x \\
\mathcal{E}_{\mathsf{sign}} &: \quad \mathsf{verify}(\mathsf{sign}(x, y, z), x, \mathsf{pub}(y)) = \mathsf{ok} \\
\mathcal{E}_{\mathsf{zk}} &: \quad \mathsf{ver}(\mathsf{prove}(\mathsf{enc}(x, \mathsf{h}(y)), \mathsf{g}^y, x, y), \\
&\qquad\qquad \mathsf{enc}(x, \mathsf{h}(y)), \mathsf{g}^y, \mathsf{f}(x))) = \mathsf{ok}
\end{aligned}$$

4.2. Cryptographic primitives and functionalities

The cryptographic primitives are modelled by an equational theory presented in Figure 7 while the higher level functionalities are modelled by rules presented in Figure 8.

Cryptographic primitives. For representing exponentiations, we use the same notation as in Tamarin, where term x^y represents exponentiation of x to power y and g is a constant playing the role of the generator for the exponentiation group. In addition to $\mathcal{E}^{+}_{\mathsf{ag}}$ used in our protocol, we consider $\mathcal{E}^{*}_{\mathsf{ag}}$ modelling the abelian group property for the operation $*$ used in the exponent. The theory $\mathcal{E}_{\mathsf{exp}} \cup \mathcal{E}^{*}_{\mathsf{ag}}$ is the so-called Diffie-Hellman theory used in the original Tamarin paper [65] and Tamarin disallows the use of the theory $\mathcal{E}^{*}_{\mathsf{ag}}$ in protocol rules. We have no restrictions on the use of $\mathcal{E}^{+}_{\mathsf{ag}}$ or $\mathcal{E}^{*}_{\mathsf{ag}}$ for our reduction. $\mathcal{E}_{\mathsf{enc}}$ and $\mathcal{E}_{\mathsf{sign}}$ represent the standard theories for symmetric encryption and randomised signatures.The theory $\mathcal{E}_{\mathsf{zk}}$ models the zero-knowledge proof functionality Prove introduced in Section 2 and used in our protocol. A term of the form $\mathsf{prove}(\mathsf{enc}(x, h(y)), g^y, x, y)$ represents a zero-knowledge proof that a given ciphertext $\mathsf{enc}(x, h(y))$ encrypts a valid witness for a statement $\mathsf{f}(x)$ and that the encryption key can be obtained by applying a hash function to the discrete logarithm of g^y. This model of zero-knowledge proofs is similar to the general symbolic model introduced in [7].

NP relations and PK infrastructure. As in [18], we consider a function symbol f to represent an NP relation, where $\mathsf{f}(w)$ represents a valid statement, while w is the corresponding witness. The adversary cannot obtain a witness w from a statement $x = \mathsf{f}(w)$, while it is efficient to verify that a candidate witness w' is indeed valid, by checking whether $\mathsf{f}(w') = x$. The rule $\mathcal{R}_{\mathsf{witn}}$ from $\mathcal{P}_{\mathcal{NP}}$ allows an honest sender to obtain a witness w, while the adversary learns the corresponding public element $\mathsf{f}(w)$. The rule $\mathcal{R}_{\mathsf{stm}}$ allows an honest receiver to obtain a desired statement $\mathsf{f}(w)$, while the adversary knows the witness w. We rely on the rules $\mathcal{P}_{\mathcal{PK}}$ to manage public and secret keys of parties, where the first rule generates and stores

Figure 8: Specifications for cryptographic functionalities

Rules $\mathcal{P}_{\mathcal{NP}}$ for NP relations

$\mathcal{R}_{\mathsf{witn}} : [\mathsf{Fr}(w)] \Rightarrow [\mathsf{Witn}(w), \mathsf{Out}(\mathsf{f}(w))]$
$\mathcal{R}_{\mathsf{stm}} : [\mathsf{In}(w)] \Rightarrow [\mathsf{Stm}(\mathsf{f}(w))]$

Rules $\mathcal{P}_{\mathcal{PK}}$ for key registration

$\mathcal{R}_{\mathsf{hon}} : [\ \mathsf{Fr}(sk)\ ] \multimap [\ \mathsf{Honest}(\mathsf{pub}(sk))\ ] \mapsto$
$[\ \mathsf{Sk}(sk), \mathsf{Pk}(\mathsf{pub}(sk)), \mathsf{Out}(\mathsf{pub}(sk))\ ]$
$\mathcal{R}_{\mathsf{cor}} : [\ \mathsf{In}(sk)\ ] \multimap [\ \mathsf{Corrupt}(\mathsf{pub}(sk))\ ] \mapsto [\ \mathsf{Pk}(\mathsf{pub}(sk))\ ]$

Rules $\mathcal{P}_{\mathcal{L}}$ for ledger

$\mathcal{R}_{\mathsf{mine}} : [\ \mathsf{Pk}(x)\ ] \Rightarrow [\ \mathcal{L}.\mathsf{Coin}(sn, x, \mathtt{notime})\ ]$
$\mathcal{R}_{\mathsf{sign}} : [\ \mathcal{L}.\mathsf{Coin}(sn, pk, \ell), \mathsf{In}(\tau, s), \tau = \langle sn, pk', \ell' \rangle\ ]$
$\multimap [\ \mathsf{verify}(s, \tau, pk) = \mathsf{ok}, \mathsf{Spend}(sn, s, pk'),$
$\mathsf{Once}(\langle \mathtt{spend}, sn \rangle), \mathsf{Once}(\langle \mathtt{coin}, sn' \rangle)\ ] \mapsto$
$[\ \mathcal{L}.\mathsf{Coin}(sn', pk', \ell'), \mathcal{L}.\mathsf{Trans}(sn, pk_0, s, sn', pk')\ ]$

$\mathcal{R}_{\mathsf{time}} : [\ \mathcal{L}.\mathsf{Coin}(sn, pk, \mathtt{time}),$
$\mathcal{L}.\mathsf{Trans}(sn_0, pk_0, \_, sn, pk)\ ]$
$\multimap [\ \mathsf{Spend}(sn, \mathtt{time}, pk_0), (\text{and Once facts as above})\ ] \mapsto$
$[\ \mathcal{L}.\mathsf{Coin}(sn', pk_0, \mathtt{notime}),$
$\mathcal{L}.\mathsf{Trans}(sn, pk, \mathtt{time}, sn', pk_0)\ ]$

Rules $\mathcal{P}_{\mathcal{AS}}$ for two-party weak adaptor signing

$\mathcal{R}_{\mathsf{kgen}} : [\ \mathsf{Sk}(sk_1), \mathsf{In}(sk_2), \mathsf{Fr}(sk),$
$pk_1 = \mathsf{pub}(sk_1), pk_2 = \mathsf{pub}(sk_2), pk = \mathsf{pub}(sk)\ ]$
$\multimap [\ \mathsf{Honest}(pk)\ ] \mapsto [\ \mathsf{SharedSk}(\{pk_1, pk_2\}, sk),$
$\mathcal{AS}.\mathsf{SharedPk}(\{pk_1, pk_2\}, pk),\ \mathsf{Out}(pk)\ ]$

$\mathcal{R}_{\mathsf{psign}} : [\ \mathcal{AS}.\mathsf{Sender}(pk_{\mathcal{S}}, m, \mathsf{pub}(sk), y),$
$\mathcal{AS}.\mathsf{Receiver}(pk_{\mathcal{R}}, m, \mathsf{pub}(sk), Y),$
$\mathsf{SharedSk}(\{pk_{\mathcal{S}}, pk_{\mathcal{R}}\}, sk),$
$\mathsf{Fr}(r), s' = y + \mathsf{sign}(m, sk, r)\ ]$
$\multimap [\ \mathsf{if}\ Y = g^y\ \mathsf{then}\ ] \mapsto$
$[\ \mathcal{AS}.\mathsf{ReceiverOutput}(pk_{\mathcal{R}}, m, pk, Y, s')\ ]$

- Plus rules for inputs and outputs of corrupt parties

the keys for honest parties, while the second rule allows the adversary to chose any key for corrupt parties.

Bitcoin ledger. The specification $\mathcal{P}_{\mathcal{L}}$ for the bitcoin ledger is presented in Figure 8, following ledger functionality presented in Figure 2. A fact $\mathcal{L}.\mathsf{Coin}(sn, pk, \ell)$ represents a coin with serial number sn, belonging to a user with public key pk. The label $\ell \in \{\mathtt{time}, \mathtt{notime}\}$ determines whether the coin can be redeemed after a timeout or not. The first rule abstracts the coin mining process. A fact $\mathcal{L}.\mathsf{Trans}(sn, pk, s, sn', pk')$ represents the action that a coin with serial number sn has been spent towards a fresh coin sn' with new owner pk'. If the coin was spend after receiving a valid signature, s records that signature. Otherwise, $s = \mathtt{time}$ to mark that the coin was spent after a timeout. We have a restriction Ψ_{once} specifying that all the new coins on the ledger have fresh serial numbers and that there is no double spending of any coin.

Two-party weak adaptor signing. The specification $\mathcal{P}_{\mathcal{AS}}$ allows a sender and a receiver to create a shared signing key and subsequently create adaptor signatures for it, for the relation (y, g^y). To obtain an adaptor signature, both parties provide the message to be signed, the sender provides the witness y, while the receiver provides the

Figure 9: Specifications for ZKCP security properties

$\Phi^1_{\mathcal{S}}$: $\mathsf{Setup}(w) \ \wedge\ \mathsf{K}(w) \Rightarrow \mathsf{Claim}(pk, w, sn)$

$\Phi^2_{\mathcal{S}}$: $\mathsf{Claim}(pk_1, w_1, sn)@i_1 \ \wedge\ \mathsf{Claim}(pk_2, w_2, sn)@i_2$
$\Rightarrow i_1 = i_2$

$\Phi^3_{\mathcal{S}}$: $\mathsf{Claim}(pk, w, sn) \wedge\ \mathsf{Spend}(sn, \ell, pk')$
$\Rightarrow pk' = pk \vee \ell = \mathtt{time}$

$\Phi_{\mathcal{R}}$: $\mathsf{Pay}(pk, sn, x) \ \wedge\ \mathsf{Spend}(sn, \ell, pk')$
$\wedge\ \mathsf{End}(pk, sn, w) \Rightarrow \mathsf{f}(w) = x \vee pk' = pk$

statement Y. The receiver obtains the output and the adversary can control inputs and outputs for corrupt parties. We use a term $s + y$ to represent an adaptor signature for signature s and witness y. This reflects our construction of such signatures for ECDSA as shown in Figure 3 and Figure 4.

4.3. Contingent payment security properties

The definition of security properties, formalised in Definition 4, relies on trace properties from Figure 9. For the sender, our definition is the same as in [18], relying on the action fact $\mathsf{K}(w)$, recording that the adversary knows w, and the action facts $\mathsf{Setup}(w)$, $\mathsf{Claim}(pk, w, sn)$ and $\mathsf{Spend}(sn, \ell, pk')$, modeling the events introduced in Section 3.2. Security is expressed as a conjunction of three formulas $\Phi^1_{\mathcal{S}} \wedge \Phi^2_{\mathcal{S}} \wedge \Phi^3_{\mathcal{S}}$, ensuring that: 1) if the adversary learns a witness, then an honest sender can claim a coin from the blockchain as payment for that witness; 2) each coin is claimed at most once; 3) each claim is satisfied, i.e. the respective coin is transferred to the claimant, unless there is a timeout in the delivery of messages to the ledger.

For the receiver, we propose a new formalisation of security that improves upon the definition from [18]. First, our definition is protocol independent and only depends on generic events $\mathsf{Pay}, \mathsf{Spend}, \mathsf{End}$, whereas the definition from [18] requires in addition two formulas Ψ_1, Ψ_2 that depend on the protocol. Second, our definition can be given directly as input to automated verification tools like Tamarin and ProVerif, whereas part of the definition in [18] requires proof by hand or a special encoding. Finally, we also offer stronger security: while [18] ensures that the witness is deducible in general, our definition ensures that the receiver can deduce the witness by applying the specified protocol steps. Our formula relies on the action facts $\mathsf{Pay}(pk, sn, x)$, Spend and $\mathsf{End}(pk, sn, w)$ modelling the corresponding events introduced in Section 3.2. Intuitively, $\Phi_{\mathcal{R}}$ captures the same property as the conjunction of properties in [18]: if the receiver's coin is transferred on the ledger, then $\mathcal{R}$ can obtain a valid witness or it obtains a refund. Our novelty consists in a simpler formalisation relying on the action fact $\mathsf{End}(pk, sn, w)$. In any contingent payment protocol, there should be such an event recording the computed outcome.

Definition 4. For a ZKCP protocol $\mathcal{P}$, let

- $\mathcal{E}$ be an intruder theory;
- $(\mathcal{P}_{\mathcal{S}}, \Psi_{\mathcal{S}})$ be a specification for the sender role;
- $(\mathcal{P}_{\mathcal{R}}, \Psi_{\mathcal{R}})$ be a specification for the receiver role.
- $(\mathcal{P}_{\mathcal{I}}, \Psi_{\mathcal{I}})$ be a specification for the (adversarial) environment where $\mathcal{P}$ is executed.

Consider the trace properties defined in Figure 9. We say that $\mathcal{P}$ is *secure* if

$$\begin{array}{lrcl} \mathsf{Sec}_{\mathcal{S}}: & (\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{I}}, \Psi_{\mathcal{S}} \wedge \Psi_{\mathcal{I}}, \mathcal{E}) & \models & \Phi^1_{\mathcal{S}} \wedge \Phi^2_{\mathcal{S}} \wedge \Phi^3_{\mathcal{S}} \\ \mathsf{Sec}_{\mathcal{R}}: & (\mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\mathcal{I}}, \Psi_{\mathcal{R}} \wedge \Psi_{\mathcal{I}}, \mathcal{E}) & \models & \Phi_{\mathcal{R}} \end{array}$$

We say that $\mathcal{P}$ is *concurrently secure* if

$$(\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\mathcal{I}}, \Psi_{\mathcal{S}} \wedge \Psi_{\mathcal{R}} \wedge \Psi_{\mathcal{I}}, \mathcal{E}) \models \Phi^1_{\mathcal{S}} \wedge \Phi^2_{\mathcal{S}} \wedge \Phi^3_{\mathcal{S}} \wedge \Phi_{\mathcal{R}}.$$

In the above definition, we note that, if the specification $(\mathcal{P}_{\mathcal{I}}, \Psi_{\mathcal{I}})$ allows the adversary to run any number of instances of any role (as we do in our specifications), and if we assume that honest parties use different keys for different roles, then it can be shown that concurrent security is implied by simple security, see e.g. [25], [45], [46]. Formally, we will only prove simple security for our protocol, since Tamarin takes too long to run for verifying concurrent security: while simple security terminates within 3 minutes for each party, concurrent security does not terminate within 30 minutes. Since the abovecited composition results do not directly apply to our setting, we leave as future work the proof of the following proposition, which allows to reduce concurrent to simple security.

Proposition 1. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two sets of rules that do not share any fact symbol and $\mathcal{E}$ be an intruder theory that does not contain private function symbols. Assume that $\mathcal{P}$ is a set of rules that includes the message deduction rules. Then, for any trace properties Ψ, Φ_1, Φ_2, we have:*

$$\begin{array}{c} (\mathcal{P}_1 \cup \mathcal{P}, \Psi, \mathcal{E}) \models \Phi_1 \wedge (\mathcal{P}_2 \cup \mathcal{P}, \Psi, \mathcal{E}) \models \Phi_2 \\ \Longrightarrow (\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}, \Psi, \mathcal{E}) \models \Phi_1 \wedge \Phi_2 \end{array}$$

Note that the set of restrictions Ψ is the same in the two statements above, which will be the case for our case study. The idea for the proof is that, in any trace of $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}$, any instance of a rule from $\mathcal{P}_2$ can be simulated by adversarial message deduction rules from $\mathcal{P}$, by replacing any protocol facts with intruder facts, and any fresh secret data with fresh public data.

4.4. Contingent payment protocol

The formal specification for our proposed protocol is represented by the sender and receiver rules from the bottom part of Figure 5, following closely the protocol description given in the top part of that figure. Putting together the rules and restrictions from Figure 5 and Figure 8, the specification of the sender is $(\mathcal{P}_{\mathcal{S}}, \Psi_{\mathsf{once}})$, that of the receiver is $(\mathcal{P}_{\mathcal{R}}, \mathsf{true})$, and that of the environment is $(\mathcal{P}_{\mathcal{I}}, \Psi_{\mathsf{once}})$, where

$$\begin{array}{rcl} \mathcal{P}_{\mathcal{S}} & = & \{\mathsf{Setup}_{\mathcal{S}}, \mathsf{Exchange}_{\mathcal{R}}, \mathsf{Settle}_{\mathcal{S}}\} \\ \mathcal{P}_{\mathcal{R}} & = & \{\mathsf{Setup}_{\mathcal{R}}, \mathsf{Exchange}^1_{\mathcal{R}}, \mathsf{Exchange}^2_{\mathcal{R}}, \mathsf{Settle}_{\mathcal{R}}\} \\ \mathcal{P}_{\mathcal{I}} & = & \mathcal{P}_{\mathcal{L}} \cup \mathcal{P}_{\mathcal{AS}} \cup \mathcal{P}_{\mathcal{PK}} \cup \mathcal{P}_{\mathcal{NP}} \cup \mathcal{P}_{\mathcal{D}} \\ \Psi_{\mathsf{once}} & = & \mathsf{Once}(x) \mathbin{@} i \wedge \mathsf{Once}(x) \mathbin{@} j \Rightarrow i = j \end{array}$$

and $\mathcal{P}_{\mathcal{D}}$ is the set of Tamarin message deduction rules. For the specification of the sender, we have an associated restriction Ψ_{once} stating that every different session of the sender should claim a different coin as payment. The interaction between each party and the blockchain, and between the two parties is via the public channel controlled by the adversary - except for the masked signing functionality, where we rely on the specification $\mathcal{P}_{\mathcal{AS}}$ to ensure the corresponding properties.

5. Reduction of subsumed theories

The main observation that underlies our reduction is the following relation between the theories $\mathcal{E}_{\mathsf{ag}}$ and $\mathcal{E}_{\mathsf{xor}}$:

Lemma 1. *For any equational theory $\mathcal{E}$ and terms u, v, we have $u =_{\mathcal{E} \cup \mathcal{E}_{\mathsf{ag}}} v \implies u =_{\mathcal{E} \cup \mathcal{E}_{\mathsf{xor}}} v$.*

Lemma 1 is true because, for any equation $l = r \in \mathcal{E}_{\mathsf{ag}}$, we have $l =_{\mathsf{xor}} r$, as can be checked directly from Figure 6. This motivates the following notion:

Definition 5. We say that an equational theory $\mathcal{E}_0$ *is subsumed by* $\mathcal{E}_1$, denoted by $\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$, iff for any terms u, v, we have $u =_{\mathcal{E}_0} v \implies u =_{\mathcal{E}_1} v$.

We show that, if $\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$, security verification modulo $\mathcal{E}_0$ can be reduced to security verification modulo $\mathcal{E}_1$. We instantiate this to reduce security verification from $\mathcal{E} \cup \mathcal{E}_{\mathsf{ag}}$ to $\mathcal{E} \cup \mathcal{E}_{\mathsf{xor}}$, and we use this for the verification of our proposed protocol with Tamarin. For $i \in \{0, 1\}$, we let $\mathcal{I}_i$ be the intruder theory corresponding to $\mathcal{E}_i$.

5.1. Trace correspondence and non-reachability

Corollary 1. *If $\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$, then $\forall t.\ (t{\downarrow_{\mathcal{I}_0}}){\downarrow_{\mathcal{I}_1}} = t{\downarrow_{\mathcal{I}_1}}$.*

Proof. By definition, we have $t{\downarrow_{\mathcal{I}_0}} =_{\mathcal{E}_0} t$ and, from Definition 5, we deduce $t{\downarrow_{\mathcal{I}_0}} =_{\mathcal{E}_1} t$. Taking the normal form with respect to $\mathcal{I}_1$ we obtain $(t{\downarrow_{\mathcal{I}_0}}){\downarrow_{\mathcal{I}_1}} = t{\downarrow_{\mathcal{I}_1}}$. □

Corollary 1 allows to show that the normal form with respect to $\mathcal{I}_1$ of any $\mathcal{I}_0$-trace produces a valid $\mathcal{I}_1$-trace.

Proposition 2. *Assume that $\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$. Then, for any set of rules $\mathcal{P}$ and any $tr \in traces_{\mathcal{I}_0}(\mathcal{P})$, there is $tr' \in traces_{\mathcal{I}_1}(\mathcal{P})$ s.t. $|tr| = |tr'|$ and $facts(tr') = facts(tr){\downarrow_{\mathcal{I}_1}}$. We denote such a trace tr' by $tr{\downarrow_{\mathcal{I}_1}}$.*

Normalisation with respect to $\mathcal{I}_1$ may change trace properties. However, we will show that some relevant classes of properties can be carried between the two worlds of $\mathcal{E}_0$ and $\mathcal{E}_1$. For a sequence of variables X, we let $\forall X$ be the universal and $\exists X$ be the existential quantification of every variable in X.

Definition 6. We say that Φ is a *positive formula* if it does not contain negations, implications or quantifiers.

Definition 7. A trace property is a:

- *non-reachability property* if it is of the form $\forall X_0.\Phi_0 \Rightarrow \neg\exists X_1.\Phi_1$;
- *reachability property* if it is of the form $\forall X_0.\Phi_0 \Rightarrow \exists X_1.\Phi_1$.

where Φ_0 and Φ_1 are positive formulas.

Example 1. *A typical non-reachability property is secrecy, e.g. $\forall i, x.\mathsf{Secret}(x)@i \Rightarrow \neg\exists j.\mathsf{K}(x)@j$. A typical reachability property is authentication, e.g. $\forall i, x.\mathsf{Recv}(x)@i \Rightarrow \exists j.\mathsf{Sent}(x)@j$.*

An attack wrt $\mathcal{E}_0$ against a non-reachability property is shown by a trace containing a sequence of terms $t_1, \ldots, t_n$ - e.g. $\mathsf{Secret}(t), \mathsf{K}(t)$ expressing that the adversary knows a secret. According to Proposition 2, there is a corresponding $\mathcal{E}_1$-trace that contains the sequence of terms $t_1{\downarrow_{\mathcal{I}_1}}, \ldots, t_n{\downarrow_{\mathcal{I}_n}}$ in the same sequence of facts, leading to the same attack. Conversely, if we cannot find attacks with respect to $\mathcal{E}_1$, then there are none wrt $\mathcal{E}_0$.

Theorem 1. *For any set of rules $\mathcal{P}$ and non-reachability property Φ, if $\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$, then $(\mathcal{P}, \mathcal{E}_1) \models \Phi \Rightarrow (\mathcal{P}, \mathcal{E}_0) \models \Phi$.*

5.2. Handling reachability properties

The challenge in extending Theorem 1 to reachability is that $\mathcal{E}_1$ may enable new deduction steps, so we can have a fact reachable modulo $\mathcal{E}_1$ but unreachable modulo $\mathcal{E}_0$. Take the example of $\mathcal{E}_{\mathsf{ag}}$ (playing the role of $\mathcal{E}_0$) and $\mathcal{E}_{\mathsf{xor}}$ (playing the role of $\mathcal{E}_1$). If a trace contains $F(a \circ a \circ a)$ but does not contain $F(a)$, then it does not satisfy $true \Rightarrow F(a)$ with respect to $\mathcal{E}_{\mathsf{ag}}$. On the other hand, if we consider equalities modulo $\mathcal{E}_{\mathsf{xor}}$, then this trace contains $F(a)$ and therefore it satisfies $true \Rightarrow F(a)$. A reachability property may also be of the form $\Phi \Rightarrow u = v$. In a trace we may have $u\sigma =_{\mathcal{E}_{\mathsf{xor}}} v\sigma$ and the property satisfied modulo $\mathcal{E}_{\mathsf{xor}}$, but $u\sigma \neq_{\mathcal{E}_{\mathsf{ag}}} v\sigma$ and the property violated modulo $\mathcal{E}_{\mathsf{ag}}$, since the converse of Lemma 1 is not true in general.

Example 2. *Consider the Tamarin deduction rules $\mathcal{D}$ and the set of rules $\mathcal{P} = \{\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2\}$, where*

$$\begin{array}{ll} R_0: & [\mathsf{Fr}(x)] \text{—}[\ \mathsf{Reg}(x)\ ]\!\!\rightarrow [F_1(x), F_2(x), F_2(x \circ x \circ x)] \\ R_i: & [F_i(sk), \mathsf{Fr}(n), \mathsf{Fr}(r), s = \mathsf{sign}(n, sk, r)] \\ (i \in \{1,2\}) & \qquad \text{—}[\ \mathsf{Signed}_i(n, sk, r)\ ]\!\!\rightarrow [\mathsf{Out}(s)] \end{array}$$

and the reachability formulas

$$\Phi_i : \mathsf{Signed}_i(x, y, z) \Rightarrow \mathsf{Reg}(y) \qquad (i \in \{1,2\})$$

$$\begin{array}{rllll} \textit{We have:} & (\mathcal{D} \cup \mathcal{P}, \mathcal{E}_{\mathsf{xor}}) & \models \Phi_1 & (\mathcal{D} \cup \mathcal{P}, \mathcal{E}_{\mathsf{xor}}) & \models \Phi_2 \\ & (\mathcal{D} \cup \mathcal{P}, \mathcal{E}_{\mathsf{ag}}) & \models \Phi_1 & (\mathcal{D} \cup \mathcal{P}, \mathcal{E}_{\mathsf{ag}}) & \not\models \Phi_2 \end{array}$$

We will solve the these problems by restricting the terms that are subject to reachability constraints in formulas. For illustration, we consider the intruder theory $\mathcal{I}_{\mathsf{xor}}$ from [35] for representing $\mathcal{E}_{\mathsf{xor}}$:

$$\begin{array}{ll} (1)\ x \circ x \to \mathtt{1} & (3)\ x \circ \mathtt{1} \to x \\ (2)\ x \circ x \circ y \to y & (4)\ \mathtt{i}(x) \to x \end{array}$$

where rule (4) is our addition to handle the symbol $\mathtt{i}$. To represent $\mathcal{E}_{\mathsf{ag}}$, we consider the theory from [65] that we denote by $\mathcal{I}_{\mathsf{ag}}$. For a fact symbol F of arity n, we let $pos(F) = \{1, \ldots, n\}$ be the set of positions in F. We call a pair of the form (F, i), for $i \in pos(F)$, a *fact position*.

Definition 8. A fact position (F, i) *is $(\mathcal{I}_0, \mathcal{I}_1)$-complete in a set of rules $\mathcal{P}$* iff for any $F(t_1, \ldots, t_n)$ occurring in $traces_{\mathcal{I}_0}(\mathcal{P})$ we have that t_i is in $\mathcal{I}_1$-normal form.

Example 3. *We have that $(F, 1)$ is $(\mathcal{I}_0, \mathcal{I}_1)$-complete wrt the rule R:$[\mathsf{Fr}(x)]$—$[\ F(x)\ ]\!\!\rightarrow [\mathsf{Out}(x)]$, for any $\mathcal{I}_0, \mathcal{I}_1$. Indeed, for any fact $F(t)$ occurring in $traces_{\mathcal{I}_0}(\{R\})$, we have that t is a name and therefore is in normal form.*

Definition 9. Let $\mathcal{P}$ be a set of rules and Φ be a positive formula. A term t with $\mathsf{var}(t) \subseteq \mathsf{var}(\Phi)$ is $(\mathcal{I}_0, \mathcal{I}_1)$-*complete* in $(\Phi, \mathcal{P})$ iff, for every $tr \in traces_{\mathcal{I}_0}(\mathcal{P})$ and substitution σ s.t. $tr \models_{\mathcal{I}_0} \Phi\sigma$, we have that $t\sigma$ is in normal form wrt $\mathcal{I}_0 \cup \mathcal{I}_1$.

The $(\mathcal{I}_0, \mathcal{I}_1)$-completeness of a term t in $(\Phi, \mathcal{P})$ can be derived from the $(\mathcal{I}_0, \mathcal{I}_1)$-completeness of fact positions where the variables of t occur in Φ, as in the next example.

Example 4. *Continuing Example 2, let Ψ_i be $\mathsf{Signed}_i(x, y, z)@\ell$. Then y is $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete in $(\Psi_1, \mathcal{P})$. This follows since $(\mathsf{Signed}_1, 2)$ is $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete for $\mathcal{P}$. Indeed, since all arguments of Reg are names, it follows that all second arguments of Signed_1 are also names and therefore in normal form.*

On the other hand, we may have $tr \models_{\mathsf{ag}} \mathsf{Signed}_2(n, t \circ t \circ t, r)@\ell\sigma$ in a trace of $\mathcal{P}$, via the rule R_2. The term $t \circ t \circ t$ is not in $\mathcal{I}_{\mathsf{xor}}$-normal form, showing that y is not $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete in $(\Psi_2, \mathcal{P})$.

For a formula Φ, we say that a fact position of Φ is *trivial* if it contains a variable that does not occur anywhere else in Φ, except in quantifiers.

Definition 10. A reachability property $\Phi : \forall X_0.\Phi_0 \Rightarrow \exists X_1.\Phi_1$ is *$(\mathcal{I}_0, \mathcal{I}_1)$-complete for $\mathcal{P}$* iff

1) for any term t that occurs as an argument of a term atom in Φ_1, we have
   a) either $\mathsf{var}(t) \subseteq X_0$ and t is $(\mathcal{I}_0, \mathcal{I}_1)$-complete in $(\Phi_0, \mathcal{P})$
   b) or else t occurs in a trivial fact position of Φ.
2) any non-trivial fact position in Φ_1 is $(\mathcal{I}_0, \mathcal{I}_1)$-complete in $\mathcal{P}$.

Intuitively, the point 1) of Definition 10 will ensure that any term that is subject to a reachability constraint in Φ_1 is in normal form wrt $\mathcal{I}_1$ once Φ_0 is satisfied. On the other hand, the point 2) ensures that any term from a trace that may match facts in Φ_1 is also in normal form wrt $\mathcal{I}_1$.

Example 5. *Continuing Example 2 and Example 4, the formula Φ_2 is not $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete for $\mathcal{P}$, since y is not $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete for $(\Psi_2, \mathcal{P})$. On the other hand, the formula Φ_1 is $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete, since y is $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete for $(\Psi_1, \mathcal{P})$. Furthermore, $(\mathsf{Reg}, 1)$ is also $(\mathcal{I}_{\mathsf{ag}}, \mathcal{I}_{\mathsf{xor}})$-complete, as required.*

Example 6. *The formula $F(x) @ i \wedge G(x) @ j \Rightarrow i \prec j$ is always $(\mathcal{I}_0, \mathcal{I}_1)$-complete, since it does not contain any term to the right of the implication (but only timepoints). The formula $F(x) \Rightarrow G(y) \vee G(c)$ also is, if c is a constant that does not occur in $\mathcal{I}_1$, since the variable y only occurs once on the right hand side.*

Theorem 2. *For any set of rules $\mathcal{P}$ and reachability property Φ that is $(\mathcal{I}_0, \mathcal{I}_1)$-complete for $\mathcal{P}$, if $\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$, then we have $(\mathcal{P}, \mathcal{E}_1) \models \Phi \Rightarrow (\mathcal{P}, \mathcal{E}_0) \models \Phi$.*

It is convenient to allow restrictions Ψ in protocol specifications. Since restrictions occur as preconditions, and not as conclusions of the verification query, we need a definition of completeness that is dual to Definition 10.

Definition 11. A reachability property $\forall X_0.\Psi_0 \Rightarrow \exists X_1.\Psi_1$ is a *$(\mathcal{I}_0, \mathcal{I}_1)$-complete restriction for $\mathcal{P}$* if the following holds for Ψ_0: it does not contain term equality atoms; all arguments of its fact atoms are variables; and any of its fact positions is $(\mathcal{I}_0, \mathcal{I}_1)$-complete in $\mathcal{P}$.

To prove the following theorem, we fix an attack trace wrt $\mathcal{E}_0$ where Ψ holds, and show that Ψ also holds in the corresponding $\mathcal{E}_1$-trace, leading to an attack wrt $\mathcal{E}_1$. To ease the proof, we introduce a notion of subsumption for intruder theories as well:

Definition 12. We say that an intruder theory $\mathcal{I}_0$ *is subsumed by* $\mathcal{I}_1$, denoted by $\mathcal{I}_0 \sqsubseteq \mathcal{I}_1$, if any term that

is in normal form with respect to $\mathcal{I}_1$ is also in normal form with respect to $\mathcal{I}_0$.

Theorem 3. *Let $\mathcal{P}$ be any set of rules, Ψ, Φ be trace formulas and $\mathcal{E}_0, \mathcal{E}_1$ be equational theories. Assume that:*

- *$\mathcal{E}_0 \sqsubseteq \mathcal{E}_1$ and $\mathcal{I}_0 \sqsubseteq \mathcal{I}_1$;*
- *Ψ is a conjunction of $(\mathcal{I}_0, \mathcal{I}_1)$-complete restrictions for $\mathcal{P}$;*
- *Φ is either an $(\mathcal{I}_0, \mathcal{I}_1)$-complete reachability property for $\mathcal{P}$, or a non-reachability property.*

Then we have $(\mathcal{P}, \Psi, \mathcal{E}_1) \models \Phi \implies (\mathcal{P}, \Psi, \mathcal{E}_0) \models \Phi$.

Application to abelian groups. For any equational theory $\mathcal{E}$, Lemma 1 shows that $\mathcal{E} \cup \mathcal{E}_{\mathsf{ag}} \sqsubseteq \mathcal{E} \cup \mathcal{E}_{\mathsf{xor}}$. Assume that $\mathcal{E} \cup \mathcal{E}_{\mathsf{ag}}$ is represented by $\mathcal{I} \cup \mathcal{I}_{\mathsf{ag}}$ and $\mathcal{E} \cup \mathcal{E}_{\mathsf{xor}}$ by $\mathcal{I} \cup \mathcal{I}_{\mathsf{xor}}$, for some intruder theory $\mathcal{I}$. We also have:

Lemma 2. *For any intruder theory $\mathcal{I}$, $\mathcal{I} \cup \mathcal{I}_{\mathsf{ag}} \sqsubseteq \mathcal{I} \cup \mathcal{I}_{\mathsf{xor}}$.*

Therefore, from Theorem 3, we derive:

Corollary 2. *Assume $\mathcal{P}, \Psi$ and Φ are as in Theorem 3, using $(\mathcal{E} \cup \mathcal{E}_{\mathsf{ag}}, \mathcal{E} \cup \mathcal{E}_{\mathsf{xor}})$ in place of $(\mathcal{E}_0, \mathcal{E}_1)$, and $(\mathcal{I} \cup \mathcal{I}_{\mathsf{ag}}, \mathcal{I} \cup \mathcal{I}_{\mathsf{xor}})$ in place of $(\mathcal{I}_0, \mathcal{I}_1)$. Then we have $(\mathcal{P}, \Psi, \mathcal{E} \cup \mathcal{E}_{\mathsf{xor}}) \models \Phi \implies (\mathcal{P}, \Psi, \mathcal{E} \cup \mathcal{E}_{\mathsf{ag}}) \models \Phi$.*

We note that our proofs only show the soundness, but not the completeness of the reduction from ag to xor. The lack of completeness means that, if an attack is found modulo xor against a protocol supposed to run with ag, one should check the attack trace and verify if it is a real attack against the protocol.

6. Verification of the proposed protocol

Consider the specifications $\mathcal{P}_{\mathcal{S}}, \mathcal{P}_{\mathcal{R}}, \mathcal{P}_{\mathcal{R}}$ and Ψ_{once} introduced in Section 4.4 and the trace properties from Section 4.3. We let:

$$\mathcal{E}_{\mathsf{zkcp}} = \mathcal{E}^{+}_{\mathsf{ag}} \cup \mathcal{E}^{*}_{\mathsf{ag}} \cup \mathcal{E}_{\exp} \cup \mathcal{E}_0 \quad \mathcal{I}_{\mathsf{zkcp}} = \mathcal{I}^{+}_{\mathsf{ag}} \cup \mathcal{I}^{*}_{\exp} \cup \mathcal{I}_0$$
$$\mathcal{E}^{\mathsf{xor}}_{\mathsf{zkcp}} = \mathcal{E}^{+}_{\mathsf{xor}} \cup \mathcal{E}^{*}_{\mathsf{ag}} \cup \mathcal{E}_{\exp} \cup \mathcal{E}_0 \quad \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}} = \mathcal{I}^{+}_{\mathsf{xor}} \cup \mathcal{I}^{*}_{\exp} \cup \mathcal{I}_0$$

$$\begin{array}{lll} \mathsf{Sec}_{\mathcal{S}}: & (\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{I}}, \Psi_{\mathsf{once}}, \mathcal{E}_{\mathsf{zkcp}}) & \models \ \Phi^1_{\mathcal{S}} \wedge \Phi^2_{\mathcal{S}} \wedge \Phi^3_{\mathcal{S}} \\ \mathsf{Sec}_{\mathcal{R}}: & (\mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\mathcal{I}}, \Psi_{\mathsf{once}}, \mathcal{E}_{\mathsf{zkcp}}) & \models \ \Phi_{\mathcal{R}} \end{array}$$

where:

- $\mathcal{E}_{\mathsf{zkcp}}$ is as in Figure 7, with $\mathcal{E}_0 = \mathcal{E}_{\mathsf{enc}} \cup \mathcal{E}_{\mathsf{sign}} \cup \mathcal{E}_{\mathsf{zk}}$;
- $\mathcal{I}_0$ is orienting all equations from $\mathcal{E}_0$ to the right;
- $\mathcal{I}^{*}_{\exp}$ is the theory representing $\mathcal{E}^{*}_{\mathsf{ag}} \cup \mathcal{E}_{\exp}$ as in [65];
- $\mathcal{I}_{\mathsf{ag}}$ is the theory representing $\mathcal{E}^{+}_{\mathsf{ag}}$.
- $\mathcal{I}_{\mathsf{xor}}$ is the theory representing $\mathcal{E}^{+}_{\mathsf{xor}}$.

To formally prove security for our protocol, we have to prove the statements $\mathsf{Sec}_{\mathcal{S}}$ and $\mathsf{Sec}_{\mathcal{R}}$ from above. For these statements, we will apply Corollary 2 for reducing $\mathcal{E}_{\mathsf{zkcp}}$ to $\mathcal{E}^{\mathsf{xor}}_{\mathsf{zkcp}}$. For applying Corollary 2, we need to ensure that the formulas in $\mathsf{Sec}_{\mathcal{S}}$ and $\mathsf{Sec}_{\mathcal{R}}$ are $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-complete with respect to $\mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{I}}$ and $\mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\mathcal{I}}$. First, we note that Ψ_{once} is an $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-complete restriction, since Once only applies to tuples of serial numbers, honest public keys and public constants in $\mathcal{P}_{\mathcal{S}}$ or $\mathcal{P}_{\mathcal{I}}$. The security properties $\Phi^1_{\mathcal{S}}$ and $\Phi^2_{\mathcal{S}}$ are also $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-complete. This is immediate for $\Phi^2_{\mathcal{S}}$, as it only contains timepoint atoms on the right-hand side. For $\Phi^1_{\mathcal{S}}$, the term atom $\mathsf{Claim}(pk, w, sn)$ does not violate $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-completeness since w is always instantiated to a name and pk and sn are variables that do not occur anywhere else in $\Phi^2_{\mathcal{S}}$, i.e. their corresponding positions are trivial.

6.1. Formula transformation for completeness

The formulas $\Phi^3_{\mathcal{S}}$ and $\Phi_{\mathcal{R}}$ are not $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-complete: the equality $pk' = pk$ violates it because of the term pk'. Here, pk' represents a potentially adversarial public key occurring on the blockchain, and it may be e.g. of the form $pk(x + x)$, i.e. not in $\mathcal{I}_{\mathsf{xor}}$-normal form. In $\Phi_{\mathcal{R}}$, the equality $\mathsf{f}(w) = x$ also violates completeness, because the term w is provided by the adversary and may have arbitrary structure. We show how to transform these formulas into complete ones while still ensuring the desired security properties.

Pushing out corrupt keys. To solve the first problem, related to the atom $pk' = pk$, we make a case distinction according to whether pk' is corrupt or honest. Let $\Phi_X \in \{\Phi^3_{\mathcal{S}}, \Phi_{\mathcal{R}}\}$. Recall that Φ_X is of the form $\Upsilon(pk, pk') \Rightarrow pk' = pk \vee \Gamma$ where pk is the key of the honest party for which we verify the property, and the formula Γ states the expected outcome in case $pk' \neq pk$. Since any key is either honest or corrupt, Φ_X is then equivalent to the conjunction of:

$$\begin{array}{lrcl} \Phi^{\mathsf{cor}}_X: & \Upsilon(pk, pk') \wedge \mathsf{Corrupt}(pk') & \Rightarrow & \Gamma \\ \Phi^{\mathsf{hon}}_X: & \Upsilon(pk, pk') \wedge \mathsf{Honest}(pk') & \Rightarrow & pk' = pk \vee \Gamma \end{array}$$

The atom $pk' = pk$ in Φ^{hon}_X does not pose a problem now since honest keys are based on fresh names. We are left to consider the formula Γ. For $\Phi^3_{\mathcal{S}}$, Γ is $\ell = \mathtt{time}$, where ℓ is instantiated to a constant denoting the type of transaction on the blockchain. It follows that $\Phi^{3,\mathsf{hon}}_{\mathcal{S}}$ and $\Phi^{3,\mathsf{cor}}_{\mathcal{S}}$ are $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-complete. For $\Phi_{\mathcal{R}}$, Γ is $\mathsf{f}(w) = x$, which can violate completeness.

Rearranging term equalities. To solve the problem of the equality $\mathsf{f}(w) = x$, we replace it with another one that will also provide the receiver with a valid witness, but where both terms can be ensured to be in normal form. Recall that the zero-knowledge proof π, for which the receiver checks $\mathsf{ver}(\pi, c, \rho, x)$, ensures the following relation between the terms c, ρ, x:

$$\exists \beta.\ c = \mathsf{enc}(w, k) \ \wedge \ x = \mathsf{f}(w) \ \wedge \ k = \mathsf{H}(\beta) \ \wedge \ \rho = g^{\beta}$$

To obtain the witness w by decryption, the receiver computes a candidate β by computing $s' - s$, where s' is obtained from the masked signing functionality and s is the signature recorded on the blockchain ledger. It follows that the receiver obtains a valid witness if $s' = s + \beta$, where β is the exponent in the term ρ for which it verified the zero-knowledge proof. For $z \in \{\mathsf{hon}, \mathsf{cor}\}$, recall that:

$$\begin{array}{rl} \Phi^z_{\mathcal{R}}: & \Upsilon \wedge \Omega^z_0 \Rightarrow \boxed{\mathsf{f}(w) = x} \vee \Omega^z_1 \\ \Upsilon: & \mathsf{Pay}(pk, sn, \underline{x}) \ \wedge \mathsf{Spend}(sn, \ell, s, sn', pk') \\ & \qquad \wedge \mathsf{End}(pk, sn, \underline{w}) \end{array}$$

for some Ω^z_0, Ω^z_1. From the previous observations, we have that the formula $\Phi^z_{\mathcal{R}}$ is equivalent to

$$\begin{array}{rl} \Phi'^z_{\mathcal{R}}: & \Upsilon' \wedge \Omega^z_0 \Rightarrow \boxed{s' = s + \beta} \vee \Omega^z_1 \\ \Upsilon': & \mathsf{Pay}'(pk, sn, \underline{g^{\beta}}) \ \wedge \mathsf{Spend}(sn, \ell, \underline{s}, sn', pk') \\ & \qquad \wedge \mathsf{End}'(pk, sn, \underline{s'}) \end{array}$$

where we record in the fact $\mathsf{Pay}'(pk, sn, g^{\beta})$ the term $\rho = g^{\beta}$ instead of x, and in the fact $\mathsf{End}'(pk, sn, s')$ the term s' instead of w.

Constraining the algebraic structure. The advantage of the equality $s' = s + \beta$ is that we can add algebraic constraints over its terms. The term β is supposed to be a fresh random value, and this can be easily checked in the exponent as we will show. The terms s, s' also have a restricted structure enforced by the blockchain and the adaptor signing functionality. To show $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$-completeness, it is then sufficient to ensure that the terms ρ and $pk_{\mathcal{S}}$ (the public key of the sender) provided by the adversary are of the form g^t and $\mathsf{pub}(t)$ for some term t that does not interfere with the equational theory. In the symbolic model, we rely on a *private* function symbol fr and a rule $Q_{\mathsf{fr}} : [\mathsf{Fr}(t)] \Rightarrow [\mathsf{Out}(\mathsf{fr}(t))]$. Private functions cannot be applied by the adversary. It follows that for any term $\mathsf{fr}(t)$ we can ensure that t is a name. For a term ρ provided by the adversary, we will require for ρ to be of the form $g^{\mathsf{fr}(t)}$, and similarly for the public key, by considering the equational theory $\mathcal{E}_{\mathsf{fr}}$ and verification checks $\mathcal{V}_{\mathsf{fr}}$ presented below. We let $\mathcal{P}^{\mathcal{R},\mathsf{fr}}_{\mathsf{zkcp}}$ be the set of rules $\mathcal{P}^{\mathcal{R}}_{\mathsf{zkcp}}$ where we add the checks $\mathcal{V}_{\mathsf{fr}}$ to the receiver rule $\mathsf{Setup}_{\mathcal{R}}$, and the action facts $\mathsf{Pay}'(pk, sn, \rho)$ to the rule $\mathsf{Exchange}^1_{\mathcal{R}}$, and $\mathsf{End}'(pk, sn, s')$ to the rule $\mathsf{Settle}_{\mathcal{R}}$; moreover we add Q_{fr} to the set of rules.

$$\overbrace{\begin{array}{l}\mathsf{ver\_exp}(g^{\mathsf{fr}(x)}) = \mathsf{ok}\\ \mathsf{ver\_key}(\mathsf{pub}(\mathsf{fr}(x))) = \mathsf{ok}\end{array}}^{\mathcal{E}_{\mathsf{fr}}} \quad \overbrace{\begin{array}{l}\mathsf{ver\_exp}(\rho) = \mathsf{ok}\\ \mathsf{ver\_key}(pk_{\mathcal{S}}) = \mathsf{ok}\end{array}}^{\mathcal{V}_{\mathsf{fr}}}$$

We let $\mathcal{I}^{\mathsf{fr}}_{\mathsf{zkcp}}$ and $\mathcal{I}^{\mathsf{xor},\mathsf{fr}}_{\mathsf{zkcp}}$ be $\mathcal{I}_{\mathsf{zkcp}}$ and $\mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}}$ augmented with rules from $\mathcal{E}_{\mathsf{fr}}$ oriented from left to right.

6.2. Ensuring random exponents and keys

To justify the freshness constraints from the symbolic model, we show how to ensure that the terms we model as fresh are random. For the term ρ provided by $\mathcal{S}$, we will have $\rho = g^{\beta}$ for which $\mathcal{S}$ will know the secret β but it will have no ability to control its actual value. More precisely, we will have $\beta = \beta_0 + r$ where β_0 is chosen by $\mathcal{S}$ and r is indistinguishable from the output of a random function. For generating such an r, we rely on publicly *verifiable random functions* (VRF) [57], that have been successfully deployed in the context of blockchains [40], [43]. We assume a VRF-based party (e.g. a smart contract) that allows anyone to ask for fresh randomness. After such a request is recorded on the blockchain, fresh randomness is generated and publicly recorded on the ledger. Any party has a guarantee that the randomness is freshly generated after the request. Such a smart contract is already deployed on the Ethereum blockchain [22]. Then we can have the following initial protocol between $\mathcal{S}$ and $\mathcal{R}$ in our contingent payment protocol:

- $\mathcal{S}$ generates β_0 and sends g^{β_0} to $\mathcal{R}$;
- $\mathcal{R}$ requests fresh randomness from the VRF, which is recorded on the blockchain as $r \in \mathbb{Z}_q$;
- $\mathcal{S}$ computes $\beta = \beta_0 + r \in \mathbb{Z}_q$ and $\mathcal{R}$ verifies that $\rho = (g^{\beta_0})^r$.

This guarantees to $\mathcal{R}$ that the exponent $\beta_0 + r$ in $\rho = (g^{\beta_0})^r = g^{\beta_0 + r}$ is independent from the choice of the adversary. This follows from the pseudo-randomness of r generated after the initially committed value g^{β_0}. For the public key, since Bitcoin relies on ECDSA signing keys, we have $\mathsf{pub}(x) = g^x$ where $x \in \mathbb{Z}_q$ is the corresponding private key. Therefore we can apply the same technique as above to ensure that the public key that the sender uses for receiving payment in the ZKCP protocol is close to uniformly random. The above construction can be made non-interactive by sending a commitment to g^{β_0} to the blockchain and triggering the smart contract without input from the receiver. The receiver would just need to consult the blockchain for verification, after receiving ρ and g^{β_0} from the sender. We note that the parties can also run an interactive coin tossing protocol for generating a random g^{β} without using smart contracts. The advantage of using the blockchain is that g^{β} can be generated once, non-interactively, and reused for any number of receivers.

6.3. Verification results

Consider the statements $\mathsf{Sec}_{\mathcal{S}}$ and $\mathsf{Sec}_{\mathcal{R}}$ modeling the security of our protocol. Let $\mathsf{Sec}^{\mathsf{fr}}_{\mathcal{R}}$ be $\mathsf{Sec}_{\mathcal{R}}$ where we use the set of rules $\mathcal{P}^{\mathcal{R},\mathsf{fr}}_{\mathsf{zkcp}}$ instead of $\mathcal{P}^{\mathcal{R}}_{\mathsf{zkcp}}$ and we add $\mathcal{E}_{\mathsf{fr}}$ to the equational theory. In the accompanying technical report [19], we prove a series of lemmas formalising the claims from Section 6.1. From these lemmas, we derive:

Corollary 3. *For the statements*

$$\overline{\mathsf{Sec}}_{\mathcal{S}} : (\mathcal{P}^{\mathcal{S}}_{\mathsf{zkcp}}, \Psi_{\mathsf{once}}, \mathcal{E}_{\mathsf{zkcp}}) \models \Phi^1_{\mathcal{S}} \wedge \Phi^2_{\mathcal{S}} \wedge \Phi^{3,\mathsf{hon}}_{\mathcal{S}} \wedge \Phi^{3,\mathsf{cor}}_{\mathcal{S}}$$
$$\overline{\mathsf{Sec}}_{\mathcal{R}} : (\mathcal{P}^{\mathcal{R},\mathsf{fr}}_{\mathsf{zkcp}}, \Psi_{\mathsf{once}}, \mathcal{E}_{\mathsf{zkcp}} \cup \mathcal{E}_{\mathsf{fr}}) \models \Phi'^{\mathsf{hon}}_{\mathcal{R}} \wedge \Phi'^{\mathsf{cor}}_{\mathcal{R}}$$

we have $\overline{\mathsf{Sec}}_{\mathcal{S}} \implies \mathsf{Sec}_{\mathcal{S}}$ *and* $\overline{\mathsf{Sec}}_{\mathcal{R}} \implies \mathsf{Sec}^{\mathsf{fr}}_{\mathcal{R}}$.

Corollary 4. *1)* $\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{fr}}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor},\mathsf{fr}}_{\mathsf{zkcp}}$ *are intruder theories with* $\mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}} \sqsubseteq \mathcal{I}_{\mathsf{zkcp}}$ *and* $\mathcal{I}^{\mathsf{xor},\mathsf{fr}}_{\mathsf{zkcp}} \sqsubseteq \mathcal{I}^{\mathsf{fr}}_{\mathsf{zkcp}}$.

2) All restrictions and trace properties in $\overline{\mathsf{Sec}}_{\mathcal{S}}$ *are* $(\mathcal{I}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor}}_{\mathsf{zkcp}})$*-complete for* $\mathcal{P}^{\mathcal{S}}_{\mathsf{zkcp}}$.

3) All restrictions and trace properties in $\overline{\mathsf{Sec}}_{\mathcal{R}}$ *are* $(\mathcal{I}^{\mathsf{fr}}_{\mathsf{zkcp}}, \mathcal{I}^{\mathsf{xor},\mathsf{fr}}_{\mathsf{zkcp}})$*-complete for* $\mathcal{P}^{\mathcal{R},\mathsf{fr}}_{\mathsf{zkcp}}$.

Let $\overline{\mathsf{Sec}}^{\mathsf{xor}}_{\mathcal{S}}$ and $\overline{\mathsf{Sec}}^{\mathsf{xor}}_{\mathcal{R}}$ be the statements $\overline{\mathsf{Sec}}_{\mathcal{S}}$ and $\overline{\mathsf{Sec}}_{\mathcal{R}}$ where we replace the equational theory $\mathcal{E}_{\mathsf{zkcp}}$ with $\mathcal{E}^{\mathsf{xor}}_{\mathsf{zkcp}}$. From the Tamarin code provided online [1].

Proposition 3. *The statements* $\overline{\mathsf{Sec}}^{\mathsf{xor}}_{\mathcal{S}}$ *and* $\overline{\mathsf{Sec}}^{\mathsf{xor}}_{\mathcal{R}}$ *are true.*

By Corollary 4, we can apply Corollary 2 to deduce $\overline{\mathsf{Sec}}^{\mathsf{xor}}_{\mathcal{S}} \Rightarrow \overline{\mathsf{Sec}}_{\mathcal{S}}$ and $\overline{\mathsf{Sec}}^{\mathsf{xor}}_{\mathcal{R}} \Rightarrow \overline{\mathsf{Sec}}_{\mathcal{R}}$. Putting this together with Proposition 3 and Corollary 3, we can conclude the security proof for our proposed protocol:

Theorem 4. *The statements* $\mathsf{Sec}_{\mathcal{S}}$ *and* $\mathsf{Sec}^{\mathsf{fr}}_{\mathcal{R}}$ *are true.*

7. Conclusion and future work

This paper makes two main contributions: it proposes a new ZKCP protocol that improves the state of the art of fair exchange relying on blockchains, and a new method that enables security verification modulo abelian groups with automated tools. There is scope for improvement in both directions. First, the protocol needs to be proved secure in a stronger cryptographic model to get more assurance. Depending on the application, a specialised version of ZKCP may be more practical than the generic protocol we propose. For example, when the data to be exchanged is an ECDSA signature (e.g. of a contract),

[60] proposes a more efficient interactive zk proof to be plugged into [71] instead of the non-interactive zk proof.

The soundness of our reduction allows to perform security proofs with respect to one theory and conclude the security of the protocol with respect to another. It would be interesting to explore to what extent such a reduction is complete, in order to also have a correspondence of attacks. More generally, our results can form the foundation for a verification procedure modulo abelian groups, without any restriction on the class of properties. Indeed, the notion of $(\mathcal{I}_0, \mathcal{I}_1)$-completeness is related to the finite variant property [28], used in Tamarin to handle equational reasoning. A similar approach as in Tamarin could be applied to transform a specification $\mathcal{P}$ into a set of variants $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with respect to which any desired formula, or its variants, would be $(\mathcal{I}_0, \mathcal{I}_1)$-complete.

References

[1] Additional material: specifications in Tamarin. https://github.com/sbursuc/ZKCP-specifications-in-Tamarin.

[2] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *Advances in Cryptology - EUROCRYPT*, volume 2332 of *LNCS*, pages 83–107, 2002.

[3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via Bitcoin deposits. In *Financial Cryptography and Data Security Workshops*, volume 8438 of *LNCS*, pages 105–121. Springer, 2014.

[4] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on Bitcoin. In *IEEE S&P*, pages 443–458, 2014.

[5] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[6] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology - ASIACRYPT 2021*, volume 13091 of *LNCS*, pages 635–664, 2021.

[7] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE S&P*, pages 202–215. IEEE Computer Society, 2008.

[8] Waclaw Banasik, Stefan Dziembowski, and Daniel Malinowski. Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In *Computer Security - ESORICS 2016*, LNCS, pages 261–280, 2016.

[9] David A. Basin, Sebastian Mödersheim, and Luca Viganò. Algebraic intruder deductions. In *LPAR 2005*, volume 3835 of *LNCS*, pages 549–564. Springer, 2005.

[10] Amos Beimel. Secret-sharing schemes: A survey. In *Coding and Cryptology - Third International Workshop*, volume 6639 of *LNCS*, pages 11–46. Springer, 2011.

[11] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE S&P*, pages 459–474. IEEE Computer Society, 2014.

[12] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security Symposium*, 2014.

[13] Iddo Bentov and Ranjit Kumaresan. How to use Bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014*, volume 8617 of *LNCS*, pages 421–439. Springer, 2014.

[14] Bruno Blanchet. Mechanizing game-based proofs of security protocols. In *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series – D: Information and Communication Security*, pages 1–25. IOS Press, May 2012.

[15] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Found. Trends Priv. Secur.*, 1(1-2):1–135, 2016.

[16] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology - CRYPTO*, volume 1880 of *LNCS*, pages 236–254. Springer, 2000.

[17] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In *Public Key Cryptography - PKC 2006*, LNCS, pages 229–240, 2006.

[18] Sergiu Bursuc and Steve Kremer. Contingent payments on a public ledger: Models and reductions for automated verification. In *Computer Security - ESORICS*, LNCS, pages 361–382, 2019.

[19] Sergiu Bursuc and Sjouke Mauw. Contingent payments from two-party signing and verification for abelian groups. Technical report, IACR, 2022.

[20] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *Advances in Cryptology - CRYPTO*, volume 1880 of *LNCS*, pages 93–111. Springer, 2000.

[21] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *ACM CCS*, pages 229–243. ACM, 2017.

[22] Chainlink. Chainlink VRF: On-chain verifiable randomness. https://blog.chain.link/chainlink-vrf-on-chain-verifiable-randomness/.

[23] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. In *IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 261–270. IEEE Computer Society, 2003.

[24] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *ACM CCS*, pages 719–728. ACM, 2017.

[25] Ştefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *IEEE Computer Security Foundations Symposium*, pages 322–336. IEEE Computer Society, 2010.

[26] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Symposium on Theory of Computing*, pages 364–369. ACM, 1986.

[27] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *J. Cryptology*, 30(4):1157–1186, 2017.

[28] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In Jürgen Giesl, editor, *Term Rewriting and Applications*, volume 3467 of *LNCS*, pages 294–307. Springer, 2005.

[29] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *IEEE Symposium on Logic in Computer Science (LICS 2003)*, page 271. IEEE Computer Society, 2003.

[30] Hubert Comon-Lundh and Ralf Treinen. Easy intruder deductions. In *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *LNCS*, pages 225–242. Springer, 2003.

[31] Poulami Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. Fastkitten: Practical smart contracts on bitcoin. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 801–818, USA, 2019. USENIX Association.

[32] Stéphanie Delaune, Steve Kremer, and Daniel Pasaila. Security protocols, constraint systems, and group theories. In *Automated Reasoning - 6th International Joint Conference, IJCAR*, volume 7364 of *LNCS*, pages 164–178. Springer, 2012.

[33] Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis for monoidal equational theories. *Inf. Comput.*, 206(2-4):312–351, 2008.

[34] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier and MIT Press, 1990.

[35] Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, and Ralf Sasse. Automated unbounded verification of stateful cryptographic protocols with exclusive OR. In *IEEE Computer Security Foundations Symposium*, pages 359–373. IEEE Computer Society, 2018.

[36] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, pages 967–984. ACM, 2018.

[37] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. Optiswap: Fast optimistic fair exchange. In *Asia CCS*, page 543–557. ACM, 2020.

[38] Lloyd Fournier. One-time verifiably encrypted signatures a.k.a. adaptor signatures, 2019. https://github.com/LLFourn/one-time-VES.

[39] Georg Fuchsbauer. WI is not enough: Zero-knowledge contingent (service) payments revisited. In *CCS*, pages 49–62. ACM, 2019.

[40] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *IEEE EuroS&P*, pages 88–102. IEEE, 2021.

[41] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[42] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In Lie et al. [52], pages 1179–1194.

[43] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Symposium on Operating Systems Principles*, SOSP '17, page 51–68. ACM, 2017.

[44] Steven Goldfeder, Joseph Bonneau, Rosario Gennaro, and Arvind Narayanan. Escrow protocols for cryptocurrencies: How to buy physical goods using Bitcoin. In *Financial Cryptography and Data Security*, volume 10322 of *LNCS*, pages 321–339. Springer, 2017.

[45] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Computer Security Foundations Workshop*, pages 24–34. IEEE Computer Society, 2000.

[46] Andreas V Hess, Sebastian A Mödersheim, and Achim D Brucker. Stateful protocol composition. In *European Symposium on Research in Computer Security*, pages 427–446. Springer, 2018.

[47] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[48] Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (hkdf). Technical report, IETF, 2010. RFC 5869.

[49] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with XOR to the XOR-free case in the horn theory based approach. In *CCS*, pages 129–138. ACM, 2008.

[50] Ralf Küsters and Tomasz Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *Computer Security Foundations Symposium*, pages 157–171. IEEE Computer Society, 2009.

[51] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for the equational theory of abelian groups with distributive encryption. *Inf. Comput.*, 205(4):581–623, 2007.

[52] David Lie, Mohammad Mannan, Michael Backes, and Xiao Feng Wang, editors. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018.

[53] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In Lie et al. [52], pages 1837–1854.

[54] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2019.

[55] Gregory Maxwell. The first successful zero-knowledge contingent payment. https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/, 2016.

[56] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[57] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *Annual Symposium on Foundations of Computer Science*, pages 120–130. IEEE Computer Society, 1999.

[58] Pedro Moreno-Sanchez and Aniket Kate. Scriptless scripts with ecdsa. https://lists.linuxfoundation.org/pipermail/lightning-dev/attachments/20180426/fe978423/attachment-0001.pdf.

[59] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at https://bitcoin.org/bitcoin.pdf.

[60] Ky Nguyen, Miguel Ambrona, and Masayuki Abe. WI is almost enough: Contingent payment all over again. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 641–656, New York, NY, USA, 2020. Association for Computing Machinery.

[61] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-199-02, Darmstadt University of Technology, Department of Computer Science, 1999.

[62] Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. Non-interactive zero-knowledge for blockchain: A survey. *IEEE Access*, 8:227945–227961, 2020.

[63] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT*, volume 10211 of *LNCS*, pages 643–673, 2017.

[64] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *Advances in Cryptology - CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 52–72. Springer, 1987.

[65] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *25th IEEE Computer Security Foundations Symposium, (CSF'12)*, pages 78–94. IEEE Computer Society, 2012.

[66] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A^2L: Anonymous atomic locks for scalability in payment channel hubs. In *S&P*, pages 1834–1851. IEEE, 2021.

[67] Sri Aravinda Krishnan Thyagarajan and Giulio Malavolta. Lockable signatures for blockchains: Scriptless scripts for all signatures. In *S&P*, pages 937–954. IEEE, 2021.

[68] Alwen Tiu, Rajeev Goré, and Jeremy E. Dawson. A proof theoretic analysis of intruder theories. *Log. Methods Comput. Sci.*, 6(3), 2010.

[69] Florian Tramèr, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *Euro S&P*, pages 19–34. IEEE, 2017.

[70] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014. Available at https://gavwood.com/paper.pdf.

[71] Bitcoin wiki: Zero Knowledge Contingent Payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment, 2011.